Label Hierarchy Inference in Property Graph Databases

Bachelor Thesis

by
Fabian Klopfer

Faculty of Sciences
Department of Computer and Information Science

1st Reviewer: Prof. Dr. Michael Grossniklaus
2nd Reviewer: Prof. Dr. Tatjana Petrov

Konstanz, 2020

Abstract:

A lot of data contains implicit hierarchical structures, e.g. type hierarchies. The property graph model among others employed in some graph databases provides no tools to capture those internally. In this thesis we derive such hierarchies automatically. First a survey is conducted to find the most promising approaches that cluster a data set hierarchically. In the next step various features and vectors thereof are experimented with to extend the methodology to graphs, capturing the structure as well as possible. We found that there is not one specific feature vector that works well for all data sets and forms of representation in a graph, but rather needs to be constructed adaptively depending on the way data is modelled. Finally some extensions of a specific algorithm that was used during experimentation — namely Cobweb — are discussed as well as the use case of cardinality estimation in property graph databases leveraging the hierarchy as an associative multi-level histogram.

Contents

CHAPTER 1

Introduction

"Thus my central theme is that complexity frequently takes the form of hierarchy and that hierarchic systems have some common properties, independent of their specific content. Hierarchy, I shall argue, is one of the central structural schemes, that the architect of complexity uses."

- Herbert A. Simon,
Nobel Laureate and ACM Turing award winner,
The Sciences of the Artificial, 1968 [62].

Many things in everyday live are structured hierarchically. Biologists use a taxonomy to classify all sorts of creatures into sub-groups like bacteria, arachea and eukaryota. Computer scientists use taxonomies to reflect type hierarchies and inheritance in object oriented programming, and type hierarchies are used by mathematicians in type theory e.g. in Russell's Principia Mathematica to define a type system [71, 69]. Further Eleanor Rosch, a famous cognitive psychologist, assumes the existence of taxonomies during the categorizations of stimuli in the human brain [60]. Corter and Gluck call categorization "one of the most basic cognitive functions" and assume that it is performed leveraging the "hierarchies of natural categories" [17].

A lot of data contains implicit patterns, knowledge and structures [51] [13] that are not reflected explicitly in the data model. Extracting those pieces of knowledge is the task of knowledge discovery and data mining [61]. In many data models there tends to be no or only little information about the hierarchies that are implicitly present in the data. For example in relational databases, data is stored in relations or in more simple terms in tables. Thus the data model is aware of the different types of relations and how they refer to each other but not of the existing sub-types in a relation and between relations. Other data models are designed to include such hierarchies but require the specification of information by the user rather than automatically extracting relevant information from data.

When the property graph model, employed in some graph databases we find the same problem: There is no notion of something equivalent to type hierarchies. One may use this model to construct and describe such situations but those again need external intervention.
Consider the example in Table 1. In this data set we have nodes representing businesses. There exists only one label for all of them, namely 'Business'. Community or owner-assigned tags assigned are represented as a node property. By visual inspection one can see that those tags induce sub-types like restaurant, shopping or cafe with further refinements to the cooking style of restaurants for example. A visualization of this is shown in Figure 1.

This kind of information may be useful for cardinality estimation during query optimization but also for data exploration and the specification of more concise queries. The extraction requires either the specification of this hierarchy by the user assuming that it is already known beforehand or it is to be extracted automatically. Especially when there is a lot of data it is non-trivial to extract such structures by hand, even when only one property of a node provides

Node.name	Node.tags
Fernando's	restaurant, Italian
Arche	restaurant, Vietnamese
Zum Elefanten	restaurant, Thai
Campus Cafe	cafe, WiFi
Endlicht	cafe, late-night
Pano	cafe, breakfast
Lago	shopping, mall
Seerhein Center	shopping, cheap
Seepark	Shopping, expensive

Table 1 *A fictive example of business and user-defined associated tags.*

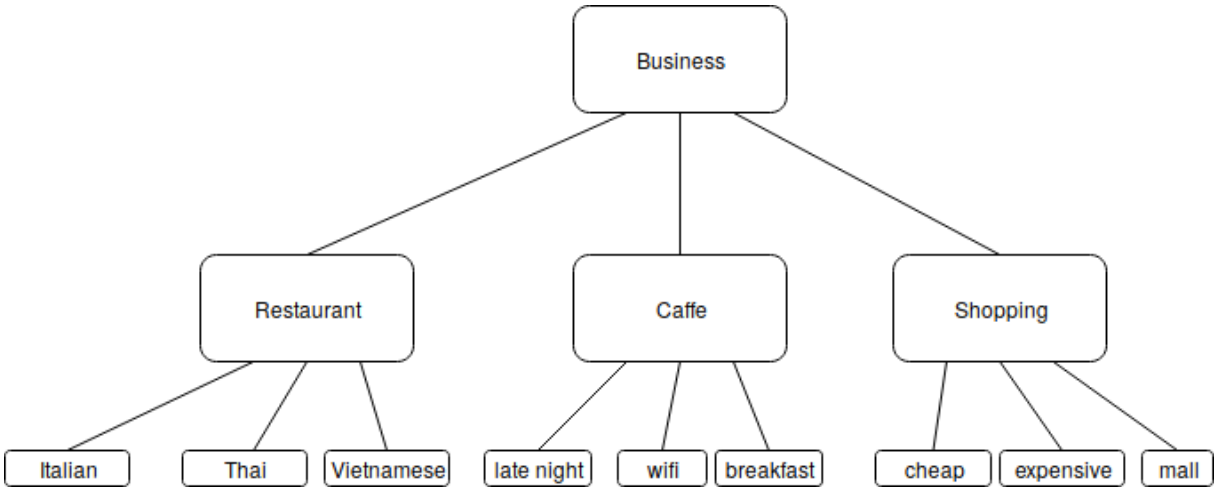

Figure 1 *The taxonomy that is implicit in the tags of the example in Table 1*

all necessary information. Given the fact that not all data sets contain community or owner defined tags but rather a mixture of many properties, structural and other semantic features like the characteristic set (see Section 3.5) that span the space of sub-types, specification by hand is infeasible in many cases.

In order to derive a taxonomy from data one needs two things: groups of similar instances and a hierarchy incorporating the former. Existing algorithmic approaches to find groups of similar instances are called clustering algorithms.

The contributions we make in this thesis are:

- A review of related literature on clustering data in a hierarchical fashion.
- The description and analysis of some of those methods.
- The combination of 'flat' clustering algorithms with hierarchical approaches to improve performance.
- A survey using the above results to extract a predefined hierarchy from synthetic data using only one attribute containing a set of tags in order to evaluate the results with the simplest possible feature vector.
- The implementation of one algorithms according to the kernel API of the Neo4j database.
- The Extension of the feature vector to capture information present in the property graph model.
- The experimentation with different extended feature vectors for different data sets to proof or reject the concept and point out methodological problems.

In the first step, a survey is carried out to find a suitable method to construct a taxonomy. Therefore different algorithms are considered, combined, improved and evaluated using a pipeline and synthetic data.
In the second step the pipeline previously applied is extended to be applicable to the property graph model and a particular algorithm is then selected and implemented — namely Cobweb — in the Neo4j property graph database to deliver a proof of concept, evaluate the proposed methodology and experiment with different feature vectors.

The rest of the thesis is structured in the following manner:
Chapter 2 presents basic terminology and a methodological overview of clustering. In Chapter 3 we describe selected clustering and feature extraction algorithms. The proposed pipeline is explained in Chapter 4. In Chapter 5 we present and discuss the results. Finally Chapter 6 contains an outlook on future work and a summary of the thesis.

CHAPTER 2

Background

2.1 The Property Graph Model

A **Property Graph** is a 9-Tuple $G = (V, E, \lambda, P, T, L, f_P, f_T, f_L)$ with

- V the set of vertices.
- E the set of edges.
- $\lambda : (V \times V) \to E$ a function assigning a pair of nodes to an edge.
- L a set of strings used as labels.
- P a set of key-value pairs called properties.
- T a set of strings used as relationship types.
- $f_P : V \cup E \to 2^P$ a function that assigns a set of properties to a node or relationship.
- $f_T : E \to T$ a function that assigns a type to a relationship.
- $f_L : V \to 2^L$ a function that assigns a node a set of labels.

The property graph model reflects a directed, node-labeled and relationship-typed multi-graph G, where each node and relationship can hold a set of key-value pairs [5].
In a graph the edges are normally defined as $E \subseteq (V \times V)$, but in the property graph model edges have sets of properties and a type, which makes them objects on their own. An illustration of this model is shown in Figure 2 . Neo4j is a graph database employing the property graph

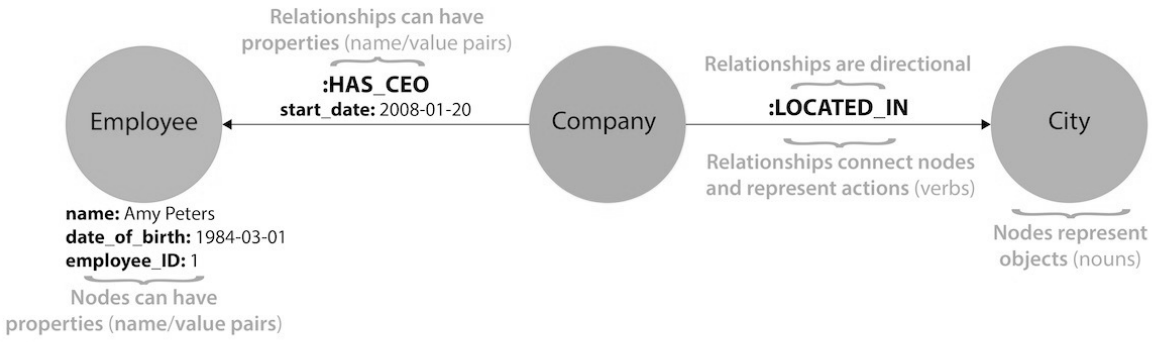

Figure 2 *A visualization of the property graph model*

model [59], which is used in the evaluation part of this thesis.

2.2 Cluster Analysis

Clustering or Cluster Analysis is defined as the automated process of splitting the set of observations into subsets, in order to group them such that objects in different subsets are different from another and objects within a subset are as similar [27].
According to Mirkin, "Clustering is a mathematical technique designed for revealing classification structures in the data collected on real-world phenomena" [49], with the purpose of analyzing structure in data, relate different aspects and assist in designing classification schemes.

Let O be the set of data objects (or equivalently data instances, data points, data sample, ...)
Let $O_0, O_1, \ldots, O_n \subseteq O$ with

- $\forall i \in \{0, 1, \ldots n\} : \cup_i O_i = O$
- $\forall i \in \{0, 1, \ldots n\} \forall j \in \{0, 1, \ldots n\} \setminus \{i\} : O_i \cap O_j = \emptyset$

Then $O_0, O_1, \ldots, O_n$ is a clustering of O.
Each subset O_i is called a cluster and clusters. For some approaches, e.g. fuzzy clustering, the constraint of disjointedness of the subsets may be relaxed.

Thus the goal of a clustering algorithm is to find a set of subsets that optimize an objective function based on the attributes and values A, V. Note that the clustering in general only imposes an order on the objects and not on the attributes, i.e. that the objects are grouped in some way is a necessity of clustering. The constraints defined on the attributes and values is imposed by the objective function of the respective algorithm. Looking at k-Means and bi-clustering * the former is grouping the attributes implicitly by the mean values of the cluster, whereas in bi-clustering each attribute is explicitly grouped by value.

Depending on the method used and the objective that is optimized for, the clustering differs between different algorithms. In order to compute any metric or to find similarities and differences or patterns in the set of objects, they must have some attributes. If no object has an attribute the only clusterings are the trivial ones:

- $O_0 \equiv O \wedge \forall i \in \{1, \ldots n\} : O_i \equiv \emptyset$
- $\forall i \in \{0, 1, \ldots |O|\} : |O_i| = 1 \wedge \sum_i |O_i| = |O|$

A class of clustering algorithms called hierarchical clustering algorithms construct sets of hierarchically ordered clusters [53]. The method has connections to the mathematical field of formal concept analysis which is out of the scope of the presented work. Fionn Murtagh provides excellent resources on the connection of hierarchical clustering to formal concept analysis [52, 53].

The second class of clustering algorithms is non-hierarchic and produces so called flat clusters, i.e. only one partition level. This distinction between hierarchic and non-hierarchic approaches is a narrow view on the wide field of clustering algorithms, a brief but broader overview is given in what follows.

2.2.1 Approaches

Different surveys list different taxonomies and categories of clustering algorithms. In what follows we are going to consider the ones discussed in Han et al. [27] and Jain et al. [32]. Han et al. compares the different methods by the representations used for data. Emphasizing that the proposed categories overlap, they define the following ones, depicted in Figure 3†:

*see Appendix A
†*Further approaches not used in this work are sketched in the appendix.*

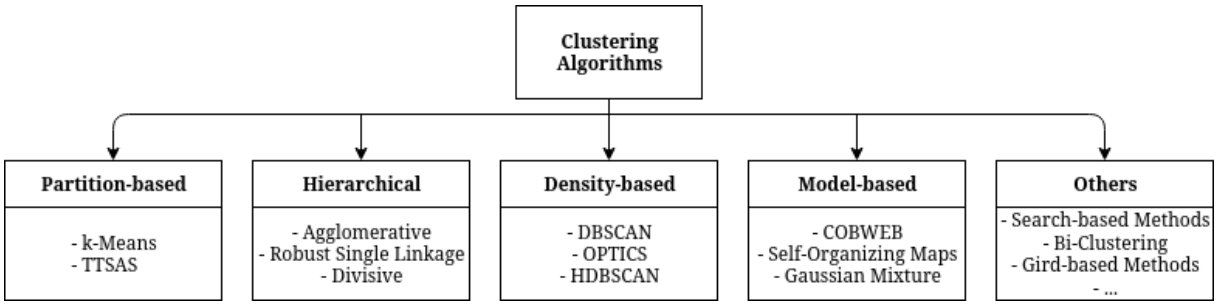

Figure 3 *A taxonomy of clustering algorithms and some examples for each class*

- **Partition-based methods:** Partition the objects into k disjoint groups. The input is partitioned only once producing a flat clustering. The requirement of disjoint groups may be relaxed for fuzzy clustering and related methods. Many partitioning algorithms use distance-based semantics, comparing the set of attributes of an object A_o element-wise, calculating the distance between those values with respect to a certain scale and metric e.g. Minkowski-distance for numeric attributes that may be interpreted geometrically as points in a space [66] or Jaccard-distance for a set of binary attributes [37]. Often partition-based methods use one or more prototypes to compare to when assigning a cluster to an object and tend to find rather spherically shaped clusters [27].

- **Hierarchical methods:** Merges (agglomerative) or splits (divisive) sets of objects recursively until all objects are assigned once in each level of the tree. Classic linkage-based approaches produce so called dendrograms [27, 53, 52] . For an example see Figure 4.

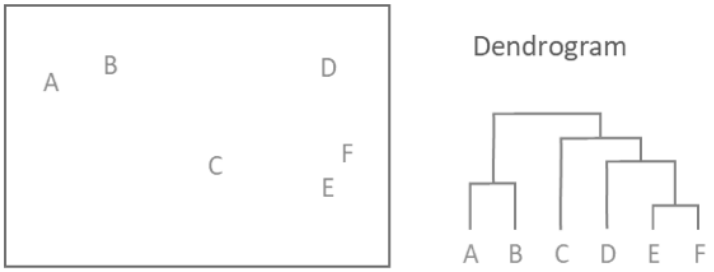

Figure 4 *An example dendrogram. [11]*

Hierarchical clustering can also be applied as a post-processing step of other methods in order to construct a hierarchy from flat clusters, which is one of the things that is applied in the later chapters.

- **Density-based methods** use a combination of distance and neighbourhood to identify dense regions which are recognized as clusters, separated by less dense regions. Density-based methods may recognize outliers (single objects in low density regions) and naturally assign a quality measure to each cluster — its density [45, 12, 19, 6]. Most density based methods produce flat clusters but there are extensions that append hierarchical clustering at the end to provide hierarchies (e.g. OPTICS and HDBSCAN), which is elaborated on in Chapter 3. Also these methods are able to find not only spherical but arbitrarily shaped clusters [19].

- **Model-based methods** are all approaches that use or construct a model to cluster instances. An example is the Gaussian Mixture Model that is the most common variant used with expectation maximization to estimate the mean and varience or in different terms the

center and radius of blob-shaped clusters [27]. In this category are also Self-Organizing Maps [36] and other neural network-based approaches, as all these assume a certain model how the neurons shall learn weights between layers [35].
Another class of methods that is model-oriented are the conceptual clustering algorithms, first introduced by Stepp and Michalski [48]. Conceptual clustering algorithms construct a description along with the clustering of objects. We will focus on this method in the latter part of Chapter 3.

2.2.2 Clustering as a Search

Clustering may be defined as a search for some set of subsets, dividing the input to satisfy some condition or optimizing for a certain metric function e.g. intra-cluster homogeneity and inter-cluster diversity [22].
Search-based methods improve *incrementally* with every iteration on a certain objective function. Search-based methods are generally all methods that optimize an objective function, but emphasize the nature of the problem as being a search for certain solution. Many clustering algorithms are also search-based, i.e. try to find an optimal solution in the space of all possible and valid solutions [32].
The search space is defined as the set of all possible partitions for flat clustering, which cardinality can be calculated using the Bell numbers [9] and in case of hierarchical clustering the set of all possible partitions of each cluster in the tree besides the bottom-most layer i.e. the layer that has single data instances as clusters. An example for the former is k-Means, improving the quality of the chosen centroids of the clusters with each iteration. An example for the latter is Cobweb, where in each iteration the constructed tree is altered in order to improve category utility. A more concise description of the algorithms is given in the next chapter.

2.3 Taxonomy

Taxonomies organize observations into hierarchical classification schemes, grouping a set of objects depending on their properties and are able to represent sub- and super-ordinations as well as inheritance. An example are biological taxonomies, grouping animals and plants into domains, kingdoms, phyla, classes, and so on [38, 27]. A Taxonomy can also be seen as a well-structured hierarchy of labels, associated with certain concepts. The term well-structured means here practically usable according to its application. For instance a taxonomy of animals where each hierarchically ordered cluster only contains one element less than the cluster above would be hierarchically ordered but not convenient in practice. A dendrogram is such a hierarchy that is not a usable taxonomy shown in Figure 4.

2.4 Probability Theory

In order to understand some aspects of the Cobweb algorithm the following definitions are required. A formally more complete introduction to probability theory is given in Appendix B.

The **expected value** $E[X]$, also known as the mean value is defined as the average of all possible outcomes, weighted by the respective probability to occur. In the case of a discrete random variable:

$$\mu = E[X] = \sum_{x_i} x_i P(X = x_i)$$

In the case of a continuous random variable:

$$\mu = E[X] = \int_{\mathbb{R}} x_i f_X(x_i) dx_i$$

The **variance** may then be defined as

$$\sigma^2 = \mathrm{Var}[X] = E[X^2] - (E[X])^2$$

The **Gaussian distribution** is a continuous probability distribution with probability density function

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}e^{-\frac{(x-\mu)^2}{2\sigma^2}}}$$

Notice that the Gaussian distribution can be identified in terms of mean and variance.

CHAPTER 3

Algorithms

There are several detailed surveys and book chapters on the topic. Mirkin explores classification and clustering from a mathematical point of view [49], Han et al. focus on data mining as a field in their book, dedicating two chapters to clustering [27]. Further there are comprehensive surveys e.g. by Jain at al. [32], Berkhin [10] or Xu et al. [74].
The algorithms in what follows are by far no comprehensive list of all methods in the field of clustering; It is rather a broad overview of a minority of the methods that were considered during evaluation. The pseudo-code for all algorithms given in the appendix in Appendix C only provides non-optimized general-purpose single-threaded variants, which have been applied, refined and optimized in various ways by the scientific community [27].
In Chapter 4 we discuss how each of these methods is used to generate label hierarchies. For all methods described below, pseudo-code and a more sophisticated complexity analysis is available in the appendix Appendix C.

3.1 Hierarchical Clustering Algorithms

Model-free hierarchical clustering algorithms aim at extracting a hierarchy from data, that is a partition of the input and partitions thereof up to a certain level. Depending on how they operate, hierarchical clustering methods may be divided into agglomerative and divisive methods [27]. Agglomerative methods start with each data instance being a cluster and merges in all subsequent steps 2 clusters at a time in a bottom-up fashion. Divisive methods start with all instances being in the same cluster and splits them consecutively top-down. We are going to focus on the agglomerative methods as there are computational challenges inherent to divisive clustering that agglomerative clustering does not have. There are $2^{n-1}-1$ possible ways to partition a set of n elements into 2 sets and in divisive clustering one needs to heuristically choose a partitioning. Backtracking could improve the performance but this is not scaleable and may end up in exponential run time [27].

3.1.1 Hierarchical Agglomerative Clustering

Agglomerative methods [63, 72, 70] aim at merging two clusters consecutively. Agglomerative methods require a distance measure for a given pair of clusters C_i, C_j. The most common measures are

- Minimum distance, also called Single Linkage:

$$d_{\min}(C_i, C_j) = \underset{e_1 \in C_i, e_2 \in C_j}{\arg\min} |e_1 - e_2|$$

- Maximum distance, also called Complete Linkage:

$$d_{\max}(C_i, C_j) = \underset{e_1 \in C_i, e_2 \in C_j}{\arg\max} |e_1 - e_2|$$

- Average distance, also called Average Linkage:

$$d_{\text{avg}}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{e_1 \in C_i, e_2 \in C_j} |e_1 - e_2|$$

A visualization of the distances is shown in Figure 5.

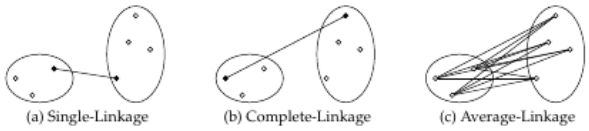

Figure 5 *Visualization of cluster distance measures*

The agglomerative algorithm with single linkage cluster distance proceeds as follows:

1. Initialize all data objects as own clusters
2. Compute the cluster distance matrix by computing the chosen distance function pairwise for all clusters
3. Remove and merge the two clusters with the minimal distance with respect to the chosen distance function. add the new cluster to the list of clusters
4. If there are more than 2 elements in the set of clusters continue with step 2. Alternatively one can specify the amount of clusters that are desired — as the parameter k in K-Medians as a stopping criterion.

Hierarchical Agglomerative Clustering has a computational complexity of $\mathcal{O}(n^3)$ and a space complexity $\mathcal{O}(n^2)$.

3.1.2 Robust Single Linkage

Robust Single Linkage [15] is a generalization of Hierarchical Agglomerative Clustering, where the method of Wishart [72] is used. Wishard developed a distance function that is more robust to noise, merging more than two clusters at a time. Robust Single Linkage considers the local density of the merged points and only merges dense regions at the lower levels. Intuitively this works as follows:

1. For each object in the data set compute $r_k(x_i)$the largest lower bound (infimum) with respect to r such that k other data objects have a lower distance than r to the current object, i.e. the smallest ball radius in the distance matrix of radius r which includes k or more data objects.
2. For $r = 0$ to ∞ construct $G_r = (V_r, E_r)$ with α a given parameter, $V_r = \{x_i \in X | r_k(x_i) \leq r\}$ and $E_r = \{i \neq j : (x_i, x_j) \in X \times X | ||x_i - x_j|| \leq \alpha r\}$, i.e. all data objects $o \in O$ which have more than k objects in a radius smaller than r are a vertex of the graph and all vertices whose distance is less than $\alpha \cdot r$ are connected by an edge.
3. The connected components of G_r form the clusters of O at level $r_{\max} - r$ in the hierarchy where $r_{\max}$ is the first value where there is only one connected component or in other words, the supremum or lowest upper bound for r of the clusters that have only one connected component.
4. Go to step 2 until there is only one connected component.

The Ball- or ϵ-neighbourhood $B(o_i, \epsilon)$ or N_ϵ of an object $o_i \in O$ is defined as

$$N_\epsilon(o_i) = \{o_j \in O \wedge i \neq j | d(o_i, o_j) \leq \epsilon\}$$

α scales weather an edge is added to the graph or not. Thus it may be regarded as the inverse density: High values for α yield sparse graphs (connecting less vertices by edges), while a low value means connecting all edges. Thus convergence of the procedure can be easily controlled by varying α.
Agglomerative Clustering with single linkage distance uses $\alpha = 1$ and $k = 2$, which was generalized by Wishart to use $\alpha = 1$ and $k \geq 2$.
Robust Single Linkage has a computational complexity of $\mathcal{O}(n^2)$ and a space complexity $\mathcal{O}(n^2)$.

3.2 Partition-based Clustering

Partition-based methods produce a flat clustering, i.e. a single partitioning of the input set O — in contrast to hierarchical methods, that produce nested partitionings. Partition-based approaches have thus the advantage of reduced run time, especially on data sets where computing a dendrogram would be prohibitive, but come with the disadvantage of an additional parameters [27, 66].

3.2.1 K-Means

The k-Means algorithm [42, 44] is one of the most commonly used algorithms for clustering [33]. K-Means chooses random initial cluster centroids, assigns the other objects to the cluster of the closest centers and then updates the centers iteratively until convergence. The intuitive idea behind k-Means is the following:

1. Initialize k random objects as initial centroids $c_1, \ldots, c_k$
2. Assign all other data objects to their closest centroid
3. update the centroids to the mean of all objects in the cluster for each feature
4. If the objective function does not change return the clusters else go to step 2

k-Means has a computational complexity of $\mathcal{O}(n)$ and a space complexity $\mathcal{O}(n)$.

3.2.2 TTSAS

The two-threshold sequential algorithmic scheme is an extension of the basic sequential algorithmic scheme, which is a generalization of ISODATA [7, 65]. To avoid specifying the number of clusters explicitly the two-threshold sequential algorithmic schema introduces two threshold parameters: Θ_1 the maximal difference of an object to a cluster so that the object is part of the cluster and Θ_2 the minimal difference of an object from all other clusters so that the object forms a new cluster. All objects with distances between Θ_1 and Θ_2 are considered to be in a "grey region", i.e. there is currently not enough information to decide cluster membership. Those objects' cluster assignment is postponed until a later point in time.
Intuitively TTSAS works as follows:

1. while there are uncategorized objects in the data set, iterate over all objects O

2. in the inner loop, if there is no change so far, if the current object o_i is unclassified and if it is the first element that is examined in the current outer loop create a new cluster consisting of this element only
3. in the inner loop else if the current object o_i is unclassified then find the minimal distance from the object o_i to a cluster C_k
    a) if the distance is below Θ_1 then add the object o_i to the cluster C_k
    b) if the distance is above threshold Θ_2, i.e. it is farther away from all clusters than Θ_2 then create a new cluster consisting of o_i only

TTSAS guarantees convergence by the first if statement: If the current object is the first that is examined in the while loop, there has been no change in this while loop so far and it is unclassified create a new cluster. For example if all objects are very far from each other but not too far i.e. with k the current number of clusters $\forall o_i \in O : \Theta_1 \leq \min_{j\in\{0,\dots,k-1\}}(d(o_i, C_j) \leq \Theta_2$, then there will only be one cluster created per outer (while) loop and it will take n outer loops to terminate, in each of which all n elements are assessed. TTSAS has a computational worst case complexity of $\mathcal{O}(n^2)$, a best case complexity of $\mathcal{O}(n)$ and a space complexity $\mathcal{O}(n)$.

3.3 Density-based Clustering

The main strategy of most density-based approaches to clustering is to find dense regions in the space of input data objects, that is separated by sparser regions. As standard hierarchical agglomerative clustering (see Section 3.1.1) and partitional clustering (see Section 3.2) methods do not use the notion of density and regions they are only able to discover relatively spherical clusters: Only considering the distance between certain points prohibits the algorithms from discovering structures that consist of many points that are not close in terms of distance but connected to each other by dense regions. Thus density-based algorithms are superior to the previous ones when data shows non-spherical patterns.

3.3.1 DBSCAN

DBSCAN [19] stands for Density-based Spatial Clustering of Applications with Noise. It was developed to meet the needs of class identification in large-scale spatial databases:

- minimal requirements of domain knowledge for input parameter determination
- discovery of arbitarily shaped clusters
- run time and space efficiency

The basic idea here is that clusters have a certain within-cluster density that is significantly higher than on the outside of the cluster. This is naively formalized by demanding that each object in the cluster has at least a minimum number of other data objects in its neighbourhood, i.e. the neighbourhood density needs to be above a certain threshold. The actual formalization handles boarder points of a cluster differently and needs some terms to be defined.

The Ball- or ϵ-neighbourhood $B(o_i, \epsilon)$ is defined as in Section 3.1.2.

An object $o_i \in O$ is directly density-reachable from another object $o_j \in O \setminus \{o_i\}$ if with ϵ the neighbourhood radius and MinObjects the density threshold

1. $o_i \in N_\epsilon(o_j)$
2. $|N_\epsilon(o_j)| \geq$MinObjects i.e. o_j is a core point and o_i is in the neighbourhood of o_j

An object $o_i \in O$ is density-reachable from another object $o_j \in O \setminus \{o_i\}$ if, with ϵ the neighbourhood radius and MinObjects the density threshold, there is a chain of directly density-reachable objects such that the chain starts at o_j and ends in o_i. Notably a border object is density-reachable from a core object but not the other way around. An object $o_i \in O$ is density-connected to another object $o_j \in O \setminus \{o_i\}$ if, with ϵ the neighbourhood radius and MinObjects the density threshold, there exists an object $o_k \in O \setminus \{o_i, o_j\}$ from which both o_i, o_j are density-reachable.

A cluster in DBSCAN C is a non-empty subset of the set of input objects O where

1. $\forall o_i, o_j \in O : o_i \in C \wedge o_j \text{density-reachable} o_i \Rightarrow o_j \in C$ (Maximality)
2. $\forall o_i, o_j \in C : o_i \text{density-connected} o_j$ (Connectivity)

Noise in DBSCAN is defined as

$$\{o_l \in O | \forall i \in \mathbb{N} : o_l \notin C_i\}$$

The intuitive idea of DBSCAN is the following:

1. Select an arbitrary object $o_i \in O$ and find all objects that are density-reachable from o_i, so all objects that are (eventually recursively) density-connected with o_i
2. if there are more than MinObjects that are density reachable from o_i i.e. o_i is a core point, a new cluster is created including all density-reachable objects. If there are no density-reachable objects, then o_i is either noise or a boarder object. In both cases, continue with the next element

DBSCAN has a computational complexity of $\mathcal{O}(n \log(n))$ and a space complexity of $\mathcal{O}(n^2)$.

3.3.2 OPTICS

OPTICS is an extension of the DBSCAN algorithm. The acronym stands for Ordering Points To Identify the Clustering Structure. Intuitively OPTICS works like DBSCAN, with different values for ϵ and stores the results using two values: The core-distance and the reachability-distance.

The core-distance of an object o_i is defined as the smallest radius for which o_i is a core point. If no such radius exists the core-distance is undefined, else it is given by

$$inf(\{\epsilon | MinObjects \leq |N_\epsilon(o_i)|\})$$

The reachability-distance of an object o_i with respect to another object $o_j \in O \setminus \{o_i\}$, is defined as the maximum of the distance between o_i, o_j and the core-distance of o_j. This means: o_j is a core object and o_i is directly density-reachable from o_j and the reachability distance is just the stronger of the conditions on ϵ and $d(o_i, o_j)$ such that both conditions are met. It is undefined if no radius exists with which o_j would be a core object. Else it is given by

$$\max(\text{core-distance}(o_j), d(o_i, o_j))$$

The differences to DBSCAN are

1. OPTICS [6] writes the results in a certain order to a file. The order is defined by the selection of elements in the outer loop but especially by ordering, traversing and writing out the objects in reachability-distance order.
2. in the expand cluster method, the seeds are ordered by reachability-distance and inserted such that the sorting remains correct.
3. Clustering have to be extracted from the ordered file in an extra step.
4. the different clusters are written to the file in order with their subset relationship, i.e. the clusters with the smallest ϵ values are written out first and are contained in certain other clusters with larger ϵ values.

The authors specify OPTICS' run time complexity as $\mathcal{O}(n \log(n))$ and its space complexity as $\mathcal{O}(n^2)$.

3.3.3 HDBSCAN

HDBSCAN [12] stands for Hierarchical DBSCAN is an extension of the OPTICS algorithm. It relies on a so called mutual reachability graph in order to construct a hierarchy.

The Mutual reachability distance between two objects $o_i, o_j \in O, i \neq j$ with d_{core} the core distance and respect to a minimal amount of objects necessary in the neighbourhood to be a core object MinObjects is defined as

$$d_{\text{mreach}}(o_i, o_j) = \max(d_{\text{core}}(o_i), d_{\text{core}}(o_J), d(o_i, o_j))$$

The Mutual Reachability Graph is then defined as $G_{\text{MinObjects}} = (V_{\text{MinObjects}}, E_{\text{MinObjects}})$ with $V_{\text{MinObjects}} = O$ and $E_{\text{MinObjects}} = \forall i, j \in \{0, \ldots, |O| - 1\} : (o_i, o_j)$ with weight $d_{\text{mreach}}(o_i, o_j)$.

Intuitively the method

1. compute the core distances for all data objects
2. build the mutual reachability graph
3. compute a minimum spanning tree of $G_{\text{MinObjects}}$
4. add a self edge to each node v_{o_i} in the minimal spanning tree with weight $d_{\text{core}}(o_i)$
5. remove iteratively the edges from the highest weight to the lowest. The resulting graph at each iteration is the clustering at hierarchy level i with i the loop variable.

HDBSCAN has a run time complexity of $\mathcal{O}(n^2)$ and a space complexity of $\mathcal{O}n^2$.

3.4 Model-based Clustering – Conceptual Clustering

Model-based approaches rely on a infer parameters for a pre-defined or build up a model of the data. An example here for is Gaussian mixture model and the expectation maximization algorithm: It models numerical data fitting a user-specified number of Gaussian distributions to the data. This has a couple of advantages:

1. for most model-based methods, probabilities of either cluster membership or uncertainty of the assignment are naturally provided
2. there is some notion of concept description: In Gaussian mixture modelling, the description are the mean and the standard deviation, in self organizing maps, feature maps are constructed that directly map from the feature space into the class labels when aggregating them [75], conceptual clustering methods provide concept descriptions in terms of feature occurrences or from a Bayesian point of view beliefs about the underlying value distribution per feature [22].

We are only going to look at one model-based approach — namely conceptual clustering — as the other methods are out of the scope due to some limitations. Note however that there are many model-based approaches like mixture modeling, self-organizing maps and others. For instance the limitations of SOMs are that they are not directly interpretable.

Conceptual Clustering was first defined by Michalski [47] as "an algorithm [...] in which entities are assembled into classes described by single conjunctive concepts. Thus the approach produces clusters with their descriptions." [47]. A conceptual clustering method takes a set of objects — consisting of describing features — and returns a scheme for classification of those without the

need for supervision. The most popular extension of this framework is the Cobweb framework, developed by Fisher in 1987 [22].

3.4.1 Cobweb

Fisher describes COBWEB as a conceptual clustering algorithm that is designed to maximize inference ability, is incremental (sometimes also referred to as an online algorithm) and computationally economical [22]. To achieve this, Fisher takes the viewpoint of clustering as a learning task for concepts [50] and applies the search paradigm to the task. He describes incremental learning along two dimension: search direction and search control.

As COBWEB shall produce a hierarchy of clusters along with a concept description there are three search problems present according to Fisher [23]:

1. Searching the space of Characterizations: According to Mitchell [50] the space of characterizations may be ordered by generality. Thus one can either search this space (search direction) bottom-up, starting with a very specific hypothesis and generalize incrementally or search top-down, starting with the most general hypothesis and discriminating incrementally or one can search in both directions, converging to some solution that is in the best case neither too specific nor too general. Mitchell called this the version space strategy [50].
2. Searching the space of Aggregations: This is the search through the space of all possible partitionings in all levels of the hierarchy. One can start with the smallest possible partitions and the largest number of partitions and then proceed by selectively merging two or more partitions or the other way round with only one partition consisting of all elements, splitting in subsequent search steps.
3. Searching the space of Hierarchies: This is the space of orderings of the partitions with the constraint that the partitions must be ordered by the subset relation. Here one may build the hierarchy from the smallest partitions to the largest or the other way round.

In the three-fold description above all of the three search problems have something in common: there is always a top-down, a bottom-up and a flexible strategy. Both the bottom-up and the top-down approaches guarantee convergence by their nature while with more flexible strategy requires the search control to guarantee for convergence. Further the searches are embedded: The search for characterizations defines the scope of objects for the search through all aggregations: considering a general concept description implies many objects that fit to the characterization, broadening the space of possible aggregations. The same applies regarding the search for aggregations and the search for hierarchies: More possible aggregations allow for more possible hierarchies. Thus search control is common to all three search tiers at the same time, as well as search direction is highly dependent between the tiers.

Cobweb uses a heuristic evaluation measure called Category Utility introduced by Gluck and Corter [25]. According to the authors "We propose that a category is useful to the extent that it can be expected to improve our ability to (a) accurately predict the values of features for members of that category and (b) efficiently communicate information to others about the features of instances of that category." [25]. The category utility is defined with P a probability measure as defined in Section 2.4 as

$$\frac{1}{n}\sum_{k=1}^{n} P(C_k)(\sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2)$$

Gluck and Corter prove that the category utility may be understood as the increase in the expected number of correctly guessed attribute values given the partition over a guess with no

information about the partition. In terms of information theory the category utility is a measure for the expected reduction of unceratinty given access to the information that the instance o_i is in category C_k [17]. Fisher also shows that the category utility may be reformulated to emphasize the optimization of the intra-class similarity, inter-class dissimilarity tradeoff with $P(A_i = V_{ij}|C_k)$ the measure for intra-class similarity and $P(C_k|A_i = V_{ij})$* as measure for inter-class dissimilarity:

$$\sum_{k=1}^{n} \sum_{i} \sum_{j} P(A_i = V_{ij} P(C_k|A_i = V_{ij}) P(A_i = V_{ij}|C_k) = \sum_{k=1}^{n} P(C_k) \sum_{i} \sum_{j} P(A_i = V_{ij}|C_k)^2$$

This means the more common attribute-value pairs in a class or cluster or concept, the higher is $P(A_i = V_{ij}|C_k)$ for a certain class and the most common attribute value pair and the less classes share the same attribute value pairs, the higher is $P(C_k|A_i = V_{ij})$ for a certain attribute and the class having $A_i = V_{ij}$. This equals the first part of the CU and may be interpreted as expected number of attribute values that can be correctly guessed for class C_k.

The characterization of concepts in Cobweb is simple: For each concept, store the number of data objects that are member of the concept, with the attribute and value counts in order to compute the probabilities.

Attribute	ValueType	Value	Probability	Occurences
Labels	Nominal	Forum	0.7403	57
	Nominal	Person	0.0779	6
	Nominal	Tag	0.0909	7
	Nominal	Company	0.0390	3
	Nominal	Organisation	0.0779	6
	Nominal	University	0.0390	3
	Nominal	Place	0.0130	1
	Nominal	City	0.0130	1

Table 2 *SNB Labels Only: ConceptNode I11 P(node) = 0.10829817158931083 Count 77*

The search operators are

- Classifying the object to an existing class, i.e. updating the counts of the concept and adding the object as child of the concept and leaf in the hierarchy
- Creating a new class: Update the counts of the super-concept, create a new node as class and add the object as leaf of the newly created class
- Merge the classes into one: Merge the two best fitting classes into one newly created super-class
- Split a concept into its children: Remove the concept and append its' children to the super concept

The former two operators end the current iteration: The iteration stops when an object is added to an existing concept or a new concept is created where it is added to. The latter two operators enable the algorithm to search the space of hierarchies, clusters and characterisations in both ways, i.e. to apply the version space strategy.
The control strategy is thus hill climbing, optimizing the category utility. The convergence is guaranteed, as if a split occurred a merge will always create the previous state which had a smaller category utility (as else the split wouldn't have happened) and the other way around. Subsequent splits have to end at the root and subsequent merges stop improving when a too dissimilar concept is merged which is guaranteed to be the case as the root is reached after

* *The less classes share the same attribute value pairs, the higher is this probability*

Algorithm 1: COBWEB

Input: An object $o_i \in O$
Parameters: node the node currently visiting
Output: A hierarchy of clusters with concept descriptions

```
begin
    updateCounts(node, o_i);
    if node is a leaf then
        node.add_child(o_i);
    else
        host, host_cu = find the child of node, that best hosts o_i;
        new_class_cu = probeCreateNewClass(o_i);
        merge_cu = probeMerge(host, o_i);
        split_cu = probeSplit(host);
        max_cu = max(new_class_cu, merge_cu, split_cu, host_cu);
        if max_cu == new_class_cu then
            createNewClass(o_i);
        else if max_cu == merge_cu then
            merged_node = merge(host, o_i);
            COBWEB(o_i, merged_node);
        else if max_cu == split_cu then
            split(host);
            COBWEB(o_i, node);
        else
            COBWEB(o_i, host)
```

merging all previous nodes.

The algorithm is incremental, i.e. one instance is processed at a time as opposed with all other algorithms so far, that assumed that all data is available at once and thus do not support subsequent updates when objects are added to the database. This also introduces a positional bias: The result of the algorithm is sensitive to input ordering [24].

The run time complexity of Cobweb is $\mathcal{O}(n \cdot \log(n) \cdot b^2 AV)$, where b is the average branching factor, A is the number of distinct attributes and V is the number of per attribute distinct values. The space complexity is $\mathcal{O}(n)$ which may be optimized to introduce a factor $\frac{1}{b}$ when using pointers to the data instances instead of nodes as leafs.

3.4.2 Extensions of Cobweb

The above algorithms provides support for only attribute sets of the same size, only categorical values and is sensitive to input orderings. There are several extensions of the presented algorithm that overcome those weaknesses.

Cobweb/3 introduces a version of the category utility that is able to also deal with numeric values by modelling them with a Gaussian distribution, thus replacing the innermost summation of the category utility with the integral of the equation for a Gaussian distribution. This introduces another bias (assuming Gaussian distribution) but may be easily avoided using more general solutions. What is also added is a divisor for the number of attributes present in the set to

account for differences in the number of attributes, e.g. missing values or inhomogeneous data sets.
The revised category utility is defined as

$$\frac{1}{n}\sum_{k=1}^{n} P(C_k)(\sum_{i}^{I}\sum_{j}\frac{1}{I}P(A_i = V_{ij}|C_k)^2 - \sum_{i}^{I}\sum_{j}\frac{1}{I}P(A_i = V_{ij})^2)$$

for discrete values

$$\frac{1}{n}\sum_{k=1}^{n} P(C_k)(\sum_{i}^{I}\frac{\frac{1}{\sigma_{ik}}}{I} - \sum_{i}\frac{\frac{1}{\sigma_{ip}}}{I})$$

for continuous values.

Trestle is another extension by MacLellan et al., that is motivated by human learning. It especially uses an unbiased estimator and an altered version of the A*-algorithm to rename non-matching attribute names to the best matching with respect to the catagory utility [43].

3.5 Feature Extraction

In order to make a clustering algorithm graph-aware the easiest way is to introduce new features based on the graph structures. The methods below are a brief summary of graph structure-related feature extraction and selection methods.

3.5.1 Characteristic Sets

Neumann and Moerkotte estimate cardinalities in RDF databases. RDF stands for Resource Description Format and is used in the ontology web language (OWL) to describe resources on websites. RDF basically defines triplets in the subject predicate object format. Standard histograms do often not capture the implicit structure of a database including correlations between predicates. As many triplets are used to describe the same objects, correlation is often visible between triplet predicates as the same predicate sets often describe similar objects. Neumann and Moerkotte show that the use of such predicate sets — called characteristic sets — increase carnality estimation significantly for triplet stores [54].

3.5.2 Recursive Feature Extraction

Henderson et al. [28] consider the question how to extract good features from nodes. They propose a new methods that computes regional features, that shall capture "behavioural" information on large graphs and show that this procedure is indeed effective on several graph mining tasks.
The base features Henderson et al. propose are local and ego net features, which they call together neighbourhood features.

Local features are described as measures of the node degree, as well as weighted and directed variants of the node degree depending on the graph kind.

Ego net features are all features extracted from the direct neighbours of the focused node. Examples are the neighbour's degree, with-in ego net edges, weighted and directed versions of

those, e.g. the number of incoming and outgoing edges to and from the ego net.

Regional features are then defined as all recursive aggregates of neighbourhood features. Aggregates can be the mean, the sum, maximum, minimum and the variance. As the number of regional features is essentially unlimited, the authors employ logarithmic binning as a correlation-related pruning technique.

A subsequent paper by Henderson's group [29] proposes to group the so constructed feature vector by applying clustering in order to derive roles or graph-structure based classes from those feature vectors and show that those learned roles transfer across graphs concerning network activity classification.

CHAPTER 4

Label Inference

4.1 Problem Statement

When focusing on data objects or nodes in terms of the property graph model there is often an implicit hierarchy or taxonomy of types. Such an implicit type taxonomy may provide further insights about data as it may e.g. help in estimating cardinalities during query optimization, when keeping further statistics about the latent class distribution and may also reveal additional information about the type of instances, like how to categorize an instance, is it rare or special from a certain point view, is it an outlier, ... In practice the categories are just informal strings, there is no notion of this structure in the current representation of the property graph model. In order to leverage this implicit taxonomy it needs to be made explicit.

For example businesses may be further categorized in e.g. restaurants, spas, shopping malls, etc. One may further subdivide the restaurants by additional properties and features like cooking style (Italian, Asian, burgers, ...) or by location (Italian restaurants in New York, Asian restaurants in Manhattan). An example tag distribution is depicted in Table 3:

Node.name	Node.tags
Fernando's	restaurant, Italian
Arche	restaurant, Vietnamese
Zum Elefanten	restaurant, Thai
Campus Cafe	cafe, WiFi
Endlicht	cafe, late-night
Pano	cafe, breakfast
Lago	shopping, mall
Seerhein Center	shopping, cheap
Seepark	Shopping, expensive

Table 3 *A fictive example of business and user-defined associated tags.*

Which may be visualized as taxonomy like in Figure 6.

Problem: Given a Property Graph $G = (V, E, \lambda, P, T, L, f_P, f_T, f_L)$, we aim at extracting a taxonomy inducing a set of meaningful labels with the following extracted information:

- the user-defined labels of the nodes V, L, f_l and types of the relationships E, T, f_T,
- the properties of the nodes and relationship V, E, P, f_P,
- per node structural features of the underlying graph that is the node degree, the average neighbour degree, the number of nodes incoming to the ego net and the number of nodes outgoing from the ego net and

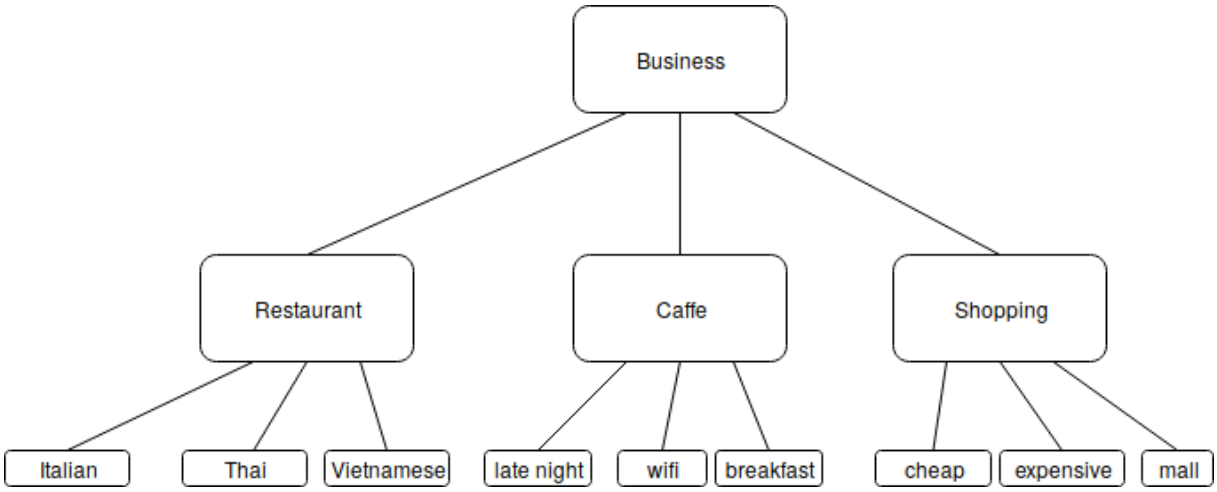

Figure 6 *The taxonomy that is implicit in the tags of the example in Table 3*

- the characteristic set of a node, that is the set of all relation types associated with this node.

4.2 Proposed Solution

Finding classes or in other terms subsets of the input is what is addressed by clustering methods. Extracting a hierarchy of classes is what hierarchical clustering copes with (see Section 3.1).

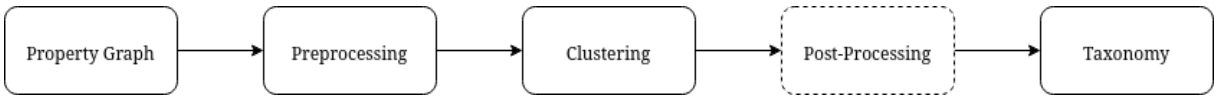

Figure 7 *The pipeline that is used to generate the taxonomy.*

4.2.1 Pre-Processing

4.2.1.1 Encoding Sets of Tags as Vectors

Algorithm 2: Vectorize tags

Input: Set of Objects O, set of sets of Attributes $A = \{A_{o_0}, A_{o_1}, \dots\}$
Parameters: No Parameters
Output: A matrix M of shape $|O| \times |A|$ with $m_{o,a} \in \{0, 1\}$
begin
  $M \rightarrow zeros(|O|, |A|)$;
  for *Object o in O* **do**
    for *Attribute a in A* **do**
      if $a \in A_o$ **then**
        $M[o, a] = 1$;

After loading the data it needs to be converted into a representation that the clustering algo-

rithms work on. Most clustering algorithms take a feature vector as input, that is a matrix of real numbers on some metric space. The only exception to this is conceptual clustering Section 3.4. Character strings need to be transformed to comply with this requirement. Text Vectorization, also referred to as One-Hot encoding builds a matrix representation, given a set of string attributes per object.
Consider the example given in Table 3. The vectorized representation of the example in table Table 3 is given in Table 4

Node.name	rest	ital	viet	thai	cafe	wifi	late	brea	shop	mall	chea	expe
Fernando's	1	1	0	0	0	0	0	0	0	0	0	0
Arche	1	0	1	0	0	0	0	0	0	0	0	0
ZumElefanten	1	0	0	1	0	0	0	0	0	0	0	0
CampusCafe	0	0	0	0	1	1	0	0	0	0	0	0
Endlicht	0	0	0	0	1	0	1	0	0	0	0	0
Pano	0	0	0	0	1	0	0	1	0	0	0	0
Lago	0	0	0	0	0	0	0	0	1	1	0	0
Seerhein Center	0	0	0	0	0	0	0	0	1	0	1	0
Seepark	0	0	0	0	0	0	0	0	1	0	0	1

Table 4 *A vector representation of the example from Table 3 corresponding to I*

4.2.1.2 Feature Vector Extension

Here we take the full complexity of available information into account: Categorical values as before, numerical values, graph structure and a few label-based semantic features using a graph database (Neo4j) employing the property graph model. As we focus on a label hierarchy we want to extract structural features from the nodes (compare with Section 2.1: Nodes have labels, relationships have a type). When no properties are present at all then structural features are the information that can be extracted. Following the approach of Henderson et al. and Neumann et al. described in Section 3.5 the features that are extracted are

- the degree of the node,
- the average neighbour degree,
- the amount of relationships that are outgoing from the ego net,
- the amount of relationships that are incoming to the ego net and
- the characteristic set.

As the proposed method shall be a proof of concepts, the extracted features are neither complete nor exhaustive. Additionally weighted, directed and recursively aggregated versions of the above should also be considered. Note that more semantic or concept-based features can be extracted, like aggregated neighbour property concepts or regarding relationships what node property types are connected by a certain edge concept. Also less features overall may provide better results as the priority of the more important features is reduced by adding more features in an unweighted feature vector.

4.2.2 Clustering

The methods described in Section 3.1.1, Section 3.1.2, Section 3.3.3 and Section 3.4.1 already produce hierarchies as outputs, so those methods may be applied to the pre-processed data

without further clustering steps. All other approaches produce only "flat" clusters. Similarly to HDBSCAN, one can apply hierarchical clustering to overcome weaknesses of both approaches: The resource requirements of hierarchical clustering algorithms are less when it is applied to the input, that was clustered before with a more efficient clustering method. In order to do this, we first apply a non-hierarchical algorithm to the input data set. Then for each cluster take the intersection of the set of tags of all objects in the cluster in order to obtain a class description for all clusters. On those newly formed objects perform hierarchical clustering to generate a hierarchy. This comes with the obvious cost of loss of information, and is especially difficult when data is noisy. Problems and ways to overcome those are discussed in Chapter 5.
Further many clustering algorithms have parameters that need to be optimized for the data set, e.g. for k-Means the number of clusters needs to be known in advance.

Applying the proposed algorithms to the extended feature vector yields something similar to Henderson's role extraction [29], but with additional information: Not only the role is analyzed but also the characteristics of the object itself. What is captured as concepts in the hierarchy or taxonomy constructed depends on the concrete feature vector that is chosen: The labels a node has and what labels occur together, the distribution over the properties a node has, what type the relations are (characteristic set), how it is embedded into the graph in terms of structure.
Returning to the road network example this would classify the crossings by the street types and their properties, meaning that a motor way junction (a few wide long streets crossing) should end up in a different concept than a country road crossing (a few long but not too wide streets crossing).
From a theoretical point of view the proposed method is able to deal with object-oriented data, triplets like in RDF and a mixture of both models, that is the property graph model. It is able to summarize information and extract probabilistic concepts from relations, components and thereof constructed objects and graphs in a parameter-free and unsupervised manner.

4.2.3 Post-Processing: Taxonomy Extraction

Applying hierarchical agglomerative clustering on the results of the previously applied algorithm yields a hierarchically ordered set of subsets of O but with only one merge per tree depth level: This is a characteristic trait of a dendrogram and the most significant difference to other trees and tree-like structures. Instead of consisting of only $\log_k n$ levels it consists of n levels. Thus the dendrogram can be flattened to get the corresponding "well-formed" tree of clusters - a taxonomy.
The output of agglomerative clustering is a linkage matrix, a matrix of shape $3 \times |merges|$, where column 0 contains the first merged cluster, column 1 the second merged cluster and column 2 the distance of the merged clusters, representing the hierarchically ordered set of subsets. What the flattening intuitively does is aggregating merges where clusters have been merged consecutively and with the same distance.
An example input and output pair is shown in Figure 8.

Algorithm 3: Taxonomy Extraction from a Dendrogram

```
Input: linkage matrix IM
Parameters: None
Output: A taxonomy of O
begin
    previousRow ← [−1, −1, −1];
    taxonomy ← new Map;
    cluster ← new Set;
    distance ← -1;
    consecutive = False;
    for row in IM do
        if (row[0] == previousRow[0] or row[1] == previousRow[1]) and row[2] ==
         previousRow[2] then
            if taxonomy is not empty and not consecutive then
                taxonomy.removeLast();
                cluster.add(previousRow[0], previousRow[1]);
            distance ← row[2];
            cluster.add(row[0], row[1]);
            consecutive ← True;
        if consecutive then
            taxonomy.add(clusters, distance);
            cluster.clear();
            distance = -1;
        consecutive = False;
        taxonomy.add({row[0], row[1]}, row[2])
    return cluster
```

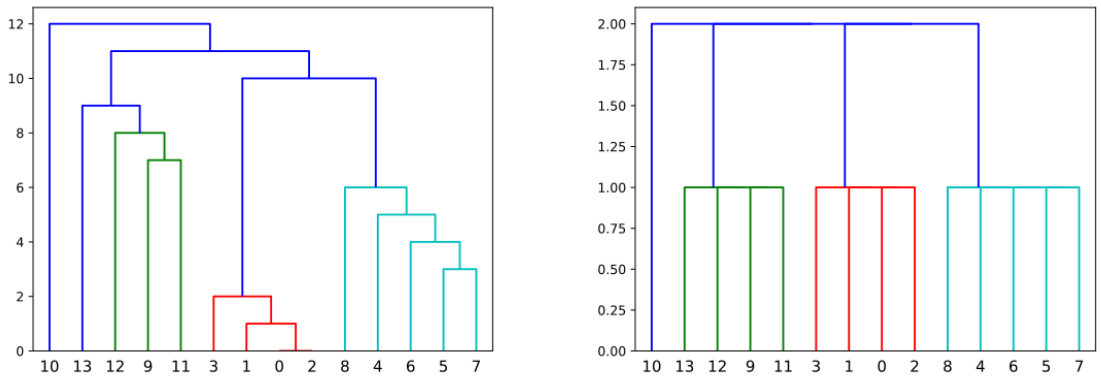

Figure 8 *The Input and output of the taxonomy extraction heuristic*

CHAPTER 5

Evaluation

As briefly mentioned in Chapter 2, different algorithms yield different clusterings as well as the same clustering with different features and parameters yields different clusterings. Thus in order to find a suitable taxonomy extraction method for the property graph model the following steps are subsequently performed:

1. Algorithm Selection: Find a suitable clustering algorithm that produces a taxonomy using a minimal feature vector and different parameters to cover the space of algorithms and parameters as extensively as feasible.
2. Feature Vector Extension: Find a suitable feature vector that also leverages the graph structure.

In the subsequent sections, the methods and steps taken is described for each of the two steps. Each step has its own requirements and objectives that are addressed in the subsequent sections, but both follow the same pipeline depicted in Figure 7.

5.1 Tag-based clustering

Tag-based clustering was mainly applied in order to probe a range of different algorithms. The selection of a fitting algorithm has three main requirements:

- Memory usage: Databases need a certain amount of memory to work, so the clustering algorithm must not use all memory to cluster data. That is the memory complexity shall be as low as possible
- Run time: The algorithm needs to have a decent run time complexity or must provide possible optimizations to scale it to large and very large data sets (i.e. millions of nodes, tens of millions of edges).
- Adaptive to nominal and numeric data: Most databases support a couple of data types, that may be summarized as either nominal (string, characters, Boolean, ...) or numeric (integer, long, floats, ...), i.e. the clustering algorithm needs to be able to support both kinds of data.
- Noise detection and tolerance: As there are frequently missing fields in data records the clustering algorithm should be tolerant to noise and detect noisy instances.

In order to evaluate the so extracted taxonomy, a way of measuring the deviation from the ground truth is needed. The Tree Edit Distance is a measure defined as the minimum cost sequence of edit operations to a tree T to transform it to another tree T' [64]. A computationally more efficient and less memory consuming implementation is APTED by Pawlik et al [56], which is used for evaluation. The visualization provides illustrations of the dendrogram, the taxonomy, the flat clusters and the resulting tree edit distance.

5.1.1 Setup

5.1.1.1 Data

Inspired by the example given in Section 4.1 a synthetic data set is generated, that constructs tag sets yielding a homogeneous taxonomy with a user-defined height and width when extracting the taxonomy by hand. The height is the number of levels in the hierarchy and the width is the number of children each inner node has. This allows to measure the extracted hierarchy against a simple ground truth.
Data sets are by nature incomplete and noisy due to measurements or human errors. So the data generator shall also be able to introduce noise, i.e. to add, remove or alter labels in the label set of an object in order to measure the robustness of the taxonomy extraction method. An example of the generated data is shown in Table 5.

Node.name	Node.labels
0	l0, l00
1	l0, l01
2	l0, l02
3	l1, l10
4	l1, l11
5	l1, l12
6	l2, l20
7	l2, l21
8	l2, l22

Table 5 *A sample of generated data for parameters height = 2, width = 3 and noise = 0*

Synthetic data provides fine grained control over noise and a ground truth to compare against. The parameters for height and width given in Table 6 are used for generation, where the height defines the number of levels in the resulting hierarchy and the width describes the number of children that each non-leaf node has. For each of them 0%, 5%, 10%, 20%, 33% noise is applied. This is done for each number of samples specified in the Table 6.

No. Samples	Width	Depth
243	3	5
512	8	3
1024	4	5
1331	11	3
1728	12	3
2197	13	3
2401	7	4
3125	5	5
4096	4	6
6561	9	4

Table 6 *The parameters used during evaluation in the synthetic data generator.*

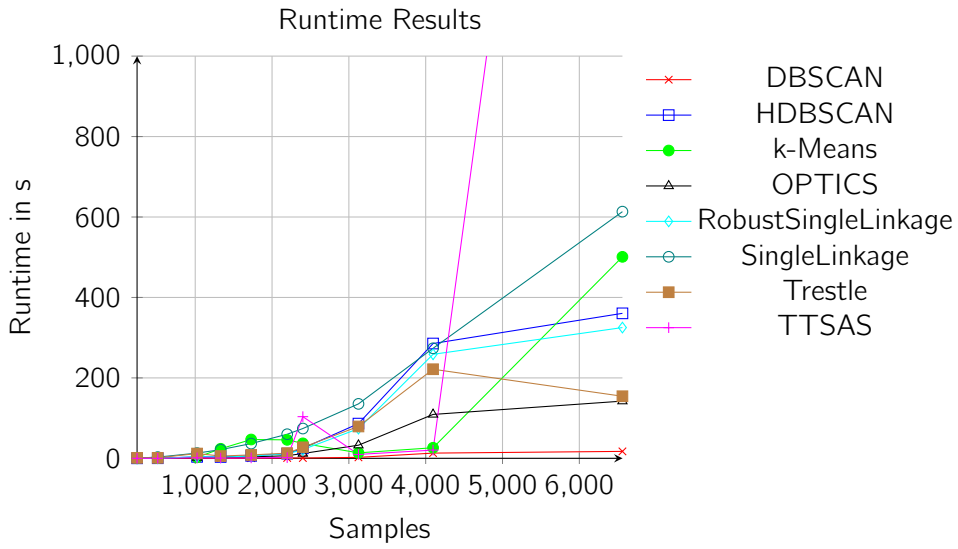

Figure 9 *Run time comparison of the different algorithms discussed.*

5.1.1.2 Implementation

In order to probe as many algorithms as possible and in a representative way, state of the art implementations of scikit-learn [57] and PyClustering have been used. Overall 16 different clustering algorithms have been tried, where only 8 of them are described in this thesis for the sake of brevity. During the implementation and experimentation the author found and fixed a bug in the widely used scikit-learn package. As most of the algorithms require parameters a parameter optimization technique was applied namely random search, with the silhouette coefficient as optimization criterion. In order to use the random search — whose implementation is using the sklearn API — with PyClustering, a wrapper around the algorithms of the latter was implemented. For conceptual clustering the TRESTLE algorithm from the concept formation package was used, that is an extension of Cobweb that tries to rename variables and restructure data to maximize the category utility when fitting it to the root node.

The pipeline was then executed on a machine with two (dual slot) 64-bit AMD EPYC 7531 16 core processors, clocked at 2.4GHz with 512GiB 2666MHz DDR4 RAM, an Intel 660P NVMe SSD running 64-bit Linux 4.15 (Ubuntu). Regarding Python the Anaconda distribution, version 1.7.2 implementing Python 3.7.3 was used, along with scikit-learn 0.22 and PyClustering 0.9.2.

5.1.2 Results

The run time comparison of the algorithms described in Chapter 3 is shown in Figure 9.
First of all some of the search spaces of the parameter optimization got limited due to performance issues (e.g. when an algorithm ran for 10 hours+ with a sample size lower than 5000). More concretely there were two such algorithms namely OPTICS and TTSAS. From an algorithmic point of view the repeated neighbourhood queries for different ϵ values impose a high linear factor, especially when choosing an epsilon value close to 1 when using Jaccard distance. Thus the maximal considered neighbourhood region was $\epsilon = 0.5$. Similarly for TTSAS, if the thresholds are too far away from each other, both too low or too high (too high means the higher threshold is greater than one and the lower close to 1), the run time degenerates to cubic and the results are rather poor. Thus the thresholds distributions were uniform between $[0.4, 0.75]$ and $[0.75, 1]$. Single Linkage has the highest run time, but sometimes intersects other algorithms, which is surprising given the fact that it has the highest computational complexity of all the

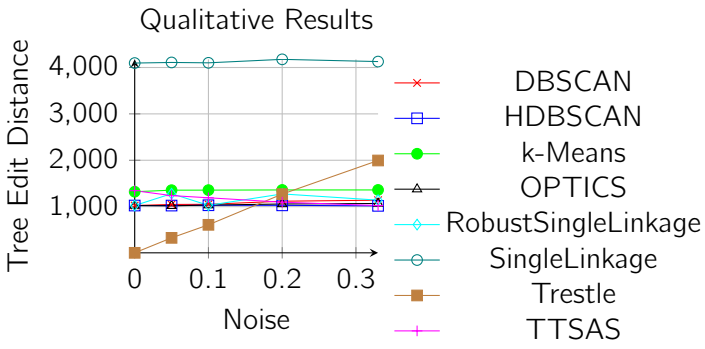

Figure 10 *Tree Edit Distance per noise level for 4096 samples*

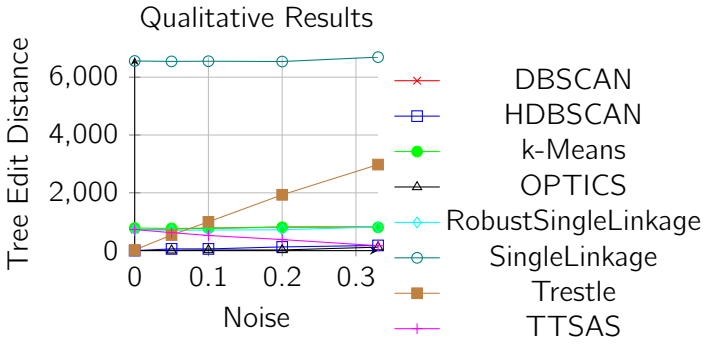

Figure 11 *Tree Edit Distance per noise level for 6561 samples*

methods. When investigating the implementation, it becomes clear that single linkage has the most mature and optimized implementation using Cython (that is C bindings) resulting in faster execution times for some corner cases. Robust Single Linkage performs better in most cases and one can see, that Robust Sinlge Linkage also bounds HDBSCAN as its most computationally intensive part. TTSAS and k-Means perform well for small sample sizes, but do not scale too well to larger ones. In case of TTSAS the experiment with 6561 samples resulted in a run time of approximately 3400s which is probably due to sub-optimal thresholds. OPTICS is always lower bounded by DBSCAN and upper bounded by HDBSCAN. DBSCAN is the most basic variant of the considered density-based approached, thus has to lowest run time for all larger sample sizes. Trestle performs similar to HDBSCAN and Robust Single Linkage.

As the resulting tree edit distance correlates with the tree size and depth (in a larger tree there are more possible edits and more possible non-matching branches when inferring a tree), there are overall 3 visualizations, showing the tree edit distance depending on the introduced noise:

- Figure 10 for a 4096 samples so for a hierarchy with depth 6 and width 4
- Figure 11 for 6561 samples that is for a hierarchy with depth 4 and width 9
- Figure 12 shows the averaged values, i.e. $\frac{\text{Tree Edit Distance}}{|\text{samples}|}$ as function of noise independent of depth

One can clearly see, that single linkage is by design not able to group objects that have equivalent attributes in one cluster in a single step, but rather in subsequent steps, resulting in a very high tree edit distance. Robust Single Linkage provides great improvements over classic Single Linkage, as it allows merging multiple objects at once and only merges multiple objects when appropriate. The density-based methods are quite similar when it comes to the tree edit distance, i.e. very robust to noise, but only in rare cases exact. k-Means is always worse than the density-based approach in terms of the tree edit distance, like TTSAS even tough TTSAS is able to reach similar TED scores as the density-based approach. Trestle is the only algorithms whose

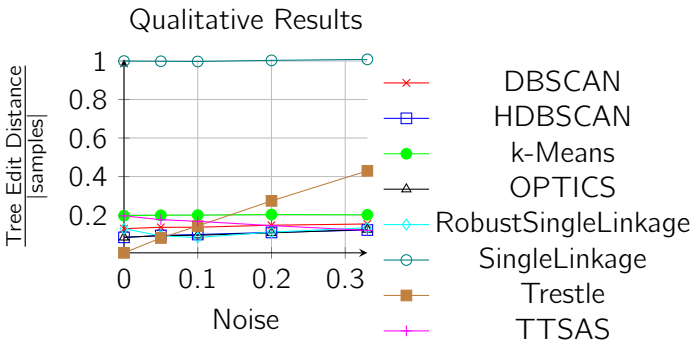

Figure 12 *Tree Edit Distance per noise level divided by sample size*

performance decreases with the introduced noise and the only algorithm that improves with the height of the ground truth hierarchy. This is especially noticable when comparing Figure 10 with Figure 11, where the point of intersection between Trestle and the density-based approaches is at a higher noise level compared due to 2 more steps in the hierarchy as compared to the latter figure where this intersections happens almost at 0 nosie. This is due to the fact that it is the only algorithm able to change the depth of the inferred hierarchy through out the whole processing step.

5.1.3 Discussion

The results with respect to time only focus on the clustering time itself. When looking at multi-phase approaches and robust single linkage, we have to keep in mind that those algorithms have parameters that need to be inferred to obtain meaningful results. Depending on the parameter space and the inference method this may take a dominant amount of time as each iteration in the inference process must use at least some representative sample of the data and execute the full algorithm on that. For example a randomized search samples parameters from used-defined distributions a pre-defined number of times and executes the algorithm on the whole data set to evaluate the results with respect to a user defined measure. This may result in factors of more than 100 for medium and large parameter spaces with non-trivial restrictions of the parameters. However note, that also the inference of parameters is of course highly dependent on an appropriate choice of the parameter distribution or restricted space (depending on inference methodology). That is the presented run times are not guaranteed to be globally optimal, but rather the local optimum found by the randomized search.
Additionally the vectorization of categorical attributes that is necessary for all algorithms besides Trestle — or more generally conceptual clustering approaches — uses additional space, that is equivalent to the number of distinct values of all attributes, what may grow relatively large. Although this amount of memory may be freed when the distance matrix is computed, one needs the memory for both the objects including vectorized attributes and the distance matrix. For large data sets this is infeasible and provides little possibilities for optimization. One would be explicit swapping when iterating over the elements during distance computation, so that at any time only a part of the matrix would be in main memory as well as only a couple of objects at a time. This may be achieved using a database and an implementation that does I/O similarly to the two-way merge sort algorithm.
Another issue is that of noisy data sets. When there is too much noise, the intersection of the attributes in a cluster may be empty, resulting in unusable object descriptions and non-sense hierarchies that get extracted. Additionally there is the need to flatten the dendrogram that is produced during Single Linkage Clustering, which can only unify subsequent merges but not restructure the merges in general.

Some of the so far evaluated approaches may be applied to the graph-aware pipeline, when integrating them more closely with the memory optimization facilities of a database in mind and further refinements, like introducing a fuzzy intersection to describe cluster representatives in the multi-phase approach.

5.2 Graph-aware clustering of Nodes

Conceptual clustering provides a lot of useful traits for this application:

- Parameter-free: As Cobweb does not need additional parameters, there is no need to tune those i.e. the algorithm fits to the data by its' own means in theory.
- Memory: It is incremental and thus uses in the most inefficient implementation $\mathcal{O}(n)$ memory. Does not require vectorization or similar things.
- Computation: Generally Conceptual clustering is able to integrate a node in logarithmic time with respect to the number of already incorporated instances, but also requires $B + 4 \cdot 3B \cdot A \cdot 2V$ floating point computations, which may grow very large, depending on the number of attributes and values. Finding an objective function that is rank-preserving and uses less computations, reducing the number of occasions where the objective function is computed and a way to only select the most discriminating attributes provides means for optimization of the computational complexity.
- Probabilistic concept descriptions: The problem of an empty intersection is overcome by using probabilistic concept descriptions, that may also provide further helpful information in query optimization.
- Hierarchical: Producing not a dendrogram but rather a data driven hierarchy of variable branching factor, that is no post-processing is needed.
- Existing extension to mixed data without additional memory or computational complexity. In contrast to what McKusick et al. reports ("[...] summing together terms from both forms of the equation works well in domains with mixed data.") [46], mixed data processing was dominated by numeric values, thus a constant bias was added to the denominator of the equation for continuous values in Section 3.4.2.

For this reason Cobweb is used for the evaluation of the graph-aware pipeline. Density-based algorithms in combination with robust single linkage may provide superior robustness and faster run times, but also require further adaptions which introduce additional degrees of freedom. As we want to experiment with different feature vectors this is undesirable here and we stick with the maybe sub-optimal but more convenient cobweb algorithm.

5.2.1 Setup

5.2.1.1 Data

The Linked Data Benchmark Council (LDBC) consists of a group of industrial and academic organizations that research on databases e.g. the Technische Universitaet Muenchen, Neo4j or the Vrije Universiteit Amsterdam. The LDBC social network benchmark aims at fulfilling the following requirements:

- Cover most demands that arise when managing data
- Modular structured, i.e. may be broken into pieces without large overhead
- Balanced selection of challenges

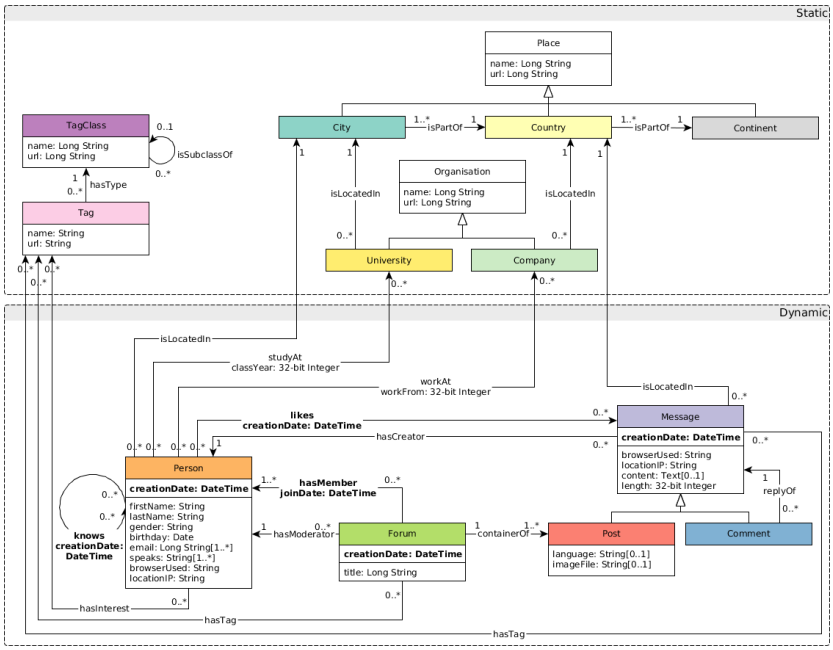

Figure 13 *The schema for the LDBC SNB benchmark*

- Modest implementation cost
- Reproducibility and documentation

The synthetic data set that is generateable using the software provided by the LDBC mimics a social network that follows the schema in Figure 13. To ensure realism the authors modeled data link distributions as found in real world social networks like Facebook and provided attribute values found in DBpedia. They also pay attention to regularities like homophily in the links of social networks, i.e. in a social context there tend to be more connections between people that behave similar and have similar interests. To ensure scaling ability of the generation process, it is implemented using the MapReduce paradigm. The LDBC SNB dataset is used to probe the pipeline as some sort of ground truth is available, even though this ground truth only considers node labels in the sense of a label in Neo4j rather than similar instances. The cardinalities of instances in the database is a dominant factor for deriving such a hierarchy. For example consider a sample size of 10000. If 9950 of 10000 samples are instances with the node label message, the output of a taxonomy will mainly provide characteristics of nodes with those labels while nodes with other labels are a minority and may even be classified as noise. Additionally the yelp data set is used as an example for a not graph but rather "object-oriented" or property-oriented data set. One experiment uses the New York Road Net data set provided by the 9th DIMACS shortest path challenge to demonstrate the pipeline on a less complex and spatially graph-oriented data set [4].

5.2.1.2 Pre-Processing

Cobweb is able to deal with character strings by design, while all other clustering algorithms introduced in Chapter 3 require dictionary vectorization as pre-processing step. A number of different combinations of the following features has been used:

- the labels of the nodes V, L, f_l
- the properties of the nodes
- per node structural features of the underlying graph that is the node degree, the average neighbour degree, the number of nodes incoming to the ego net and the number of nodes

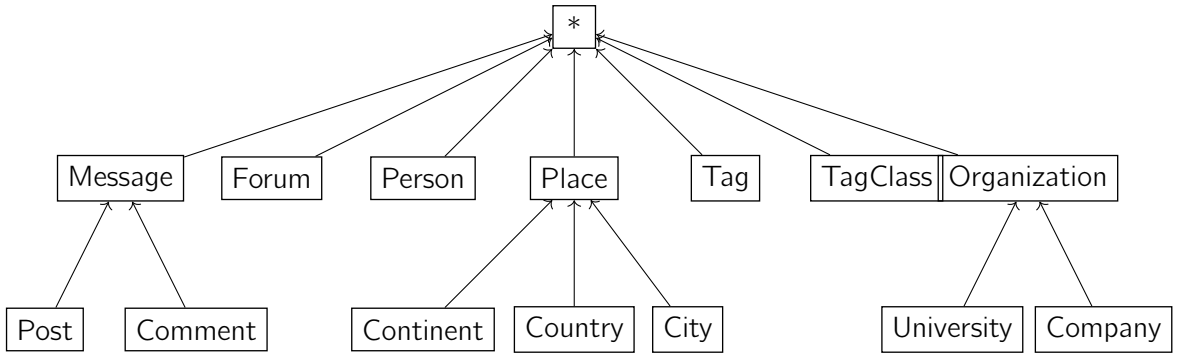

Figure 14 *Ground truth label hierarchy for the LDBC SNB data set*

outgoing from the ego net

- the characteristic set of a node, that is the set of all relation types associated with this node

The node labels have always been used. For the latter three features, all variations (i.e. the labels plus one element of the power set) have been tried.

5.2.1.3 Implementation

In order to integrate the proposed method with Neo4j and potentially move it to a deeper level in the architecture of the database it was implemented as a Cypher procedure so that it complies with the internal API of Neo4j and can be called from the query language Cypher. The implementation of Cobweb was done by the author from scratch as well as all feature extraction traversals of the graph, besides the node degree.
The implementation of Cobweb uses the adjusted category utility from [46] and an unbiased and numerically stable algorithm for computing mean and variance, namely Chan's Algorithm [14] as proposed by MacLellan et al [43]. For testing purposes, an annotation-based database setup and import framework was written, provided by Manuel Hotz and refined by the author.
In order to boost the computational performance a multi-threaded version was implemented but yielded high contention that cancelled the concurrency gains in Cobweb. There is further space for optimization regarding memory access patterns, caching and storing intermediate values to avoid the computation of the same value over and over again.

The pipeline was again executed on a machine with two (dual slot) 64-bit AMD EPYC 7531 16 core processors, clocked at 2.4GHz with 512GiB 2666MHz DDR4 RAM, an Intel 660P NVMe SSD running 64-bit Linux 4.15 (Ubuntu). Regarding Java the OpenJDK distribution, version 11.0.4 was used, along with Neo4j 3.5.3.

5.2.2 Results

In Figure 14 the ground truth label hierarchy is shown. The trees visualized in the following section are the label hierarchies that were inferred with generated abstract labels. That means the labels in the trees shown as result have no semantic meaning and are solely there to reference the table describing the same concept as the node. Also the root of the tree is always called "Root" as it always contains an aggregation of all values present in the feature vector for all incorporated nodes. The tables for all concepts shown in the trees can be found in the appendix. An exemplar table is shown in Table 9 As the trees get relatively large due to saving each instance

as an extra node, the results are cut at the second level of the tree which is mostly sub-optimal. This is a similar problem as the one of single linkage, that is further addressed in extensions of the Cobweb algorithm e.g. in [24, 68] and others. We shall visually inspect differences, as due to the above mentioned difficulties, the tree edit distance may be closer to the ground truth even tough the learned concepts aren't. Also some values in the concept description tables are not visualized, as the tables get very large (more than 100 rows, some values wider than what fits on A4 paper). Many of the trees contain the desired concepts, but at levels that are not always the same, i.e. there is no level in the tree that contains optimal concepts for all branches. For instance as the ratio of persons to messages is very low, the desired concept of persons may be at a very low level and the desired concept for messages is at a rather high level in the tree. That is concepts with high support are at a high level in the tree, while concepts with low support are at lower levels. For all the data sets a sample size of 2000 nodes and all relationships connecting to those nodes was used.

Labels Only When only taking the node labels into account on the LDBC SNB data set, the algorithm splits at the first level all Comments (which are also messages) and then all posts at the next level. This is also due to the cardinality of the label sets. The fact that both l0 and l10 conatin the label message is probably due to order effects described by Fisher in more detail [22]. All other labels are present at such low total probabilities, that the split that separates those eventually is at a deeper level in the tree. The tree is shown in Figure 15 and an example for a concept description table can be found in Table 7.

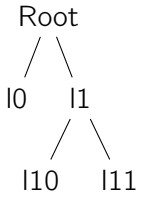

Figure 15 *Inferred Label Hierarchy with labels only*

Attribute	ValueType	Value	Probability	Occurences
Labels	Nominal	Message	1.0000	1289
	Nominal	Comment	1.0000	1289

Table 7 *Example concept description for node l0: P(node) = 0.6445, Count = 1289*

If the label distribution only consists of one equally probable label for all nodes there is no valuable information that can be used to cluster the data instances, thus the tree is made up of a single root node and no information is gained. An example here fore is the yelp data set with properties to store all information when it is clustered using only the label sets. That is for all nodes "Business" so there is only one concept containing all nodes.

Labels and Node Properties The first three levels of the generated tree is shown in Figure 16. When using the properties additionally to the set of labels, a sub-tree is appended at l0 splitting nodes that have labels message and comment further into messages of created with different browsers. The sub-tree starting at l1 is very similar to the one with only labels: It splits into nodes having the labels message and post and those having anything else but in contrast to

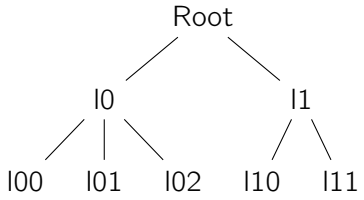

Figure 16 *Inferred Label Hierarchy for the LDBC SNB data set with labels and properties as the feature vector*

before still contains nodes having the labels message and post. The unclean split is due to the extra noise introduced with the presence of the properties. As there are a lot of properties, the predictiveness of some properties (e.g. the id, what browser was used, the creation date) weakens the effect of the label predictiveness and thus introduces additional noise.

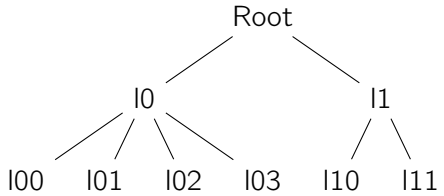

Figure 17 *Inferred Label Hierarchy for the property-oriented Yelp data set with labels and properties as the feature vector*

In the previous paragraph we noticed, that a data set with only one equi-probable label for all nodes contains no information. If a data set has node properties, adding those to the feature vector enables the use of clustering methods to extract concepts. The first three levels of the generated tree is shown in Figure 17. Depending on how predictive the properties are and in how many directions they are, the resulting hierarchy may be clean but is mostly rather messy: For instance the property-based Yelp data set contains attributes describing spatial features (the exact location including latitude, longitude, address, state, ...), time based features (the opening hours), numeric and nominal ones (user-defined categories, price ranges, an average review score and a review count for example). Optimizing for those in common yields a mixture of the above which introduces a lot of noise. Some semantically close instances are in different concept classes as their spatial position differs completely, while some spatially and semantically close instances are in distinct groups due to different tags assigned (covering traits like WiFi, Parking, GoodForChildren). Extensions of the algorithm to deal with Spatial and Date-based values, to find correlations in the attribute-value distribution and to evaluate only the most salient and discriminative values or weighting, cleaning and projection of properties before clustering may provide further refinements to improve the result using properties in the feature vectors.

Considering a road network data set, with no node labels and no properties, there is no information to cluster the nodes with, that is this feature vector yields no information for such data sets.

Labels and Characteristic Sets Taking the characteristic set into account results in more information than with labels only but less noise than with the properties for most cases. Applied to the LDBC SNB data set, the concepts are similar to the ones created with labels only with some additions: l0 is again the concept that contains nodes with labels message and comment, but additionally separates between nodes that have relationships to tags, nodes that do rather not and a class where approximately two thirds of the nodes have tags. The latter one is probably

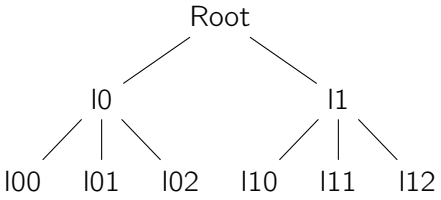

Figure 18 *Inferred Label Hierarchy for the SNB data set with labels and the characteristic set as the feature vector*

created erroneously due to ordering effects. In the sub-tree of l1 again nodes with labels Message and Post are split from the rest of the nodes. There are actually two sub-concepts for nodes with labels Message and Post: One where there are mostly tagged answer posts, and one where the posts are not in reply to something and barely have tags. The first three levels of the generated tree is shown in Figure 18.

Again if we have no relationships as in the property-oriented yelp data set we gain no information by adding the characteristic set (as it is an empty set). Another case where no relevant information is gained is the case of a road network: Every node has edges with the relationship type "ROAD". That is all characteristic sets are exactly the same containing one element — namely "ROAD".

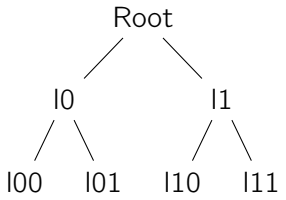

Figure 19 *Inferred Label Hierarchy for the SNB data set with labels and structural features as the feature vector*

Labels and Structural Features Here the nodes are structured as with just the labels, but additionally in the sub-tree l0 there is a distinction between lower mean node degree, lower number of outgoing edges, higher average neighbour degree and higher mean node degree, higher number of outgoing edges, lower average neighbour degree. On the sub-tree of l1 there is again the nodes having the labels message and post and those having anything else split. This time the split provides additional relevant information: nodes with labels "Message" and "Post" have mostly a low degree and a lower number of edges outgoing from the ego net but a high number of edges incoming to the ego net and a high average neighbour degree, whereas nodes with other labels have on average a higher degree, more outgoing than incoming edges to the ego net and a lower average neighbour degree. The first three levels of the generated tree is shown in Figure 19. Note however, that more complex structural features can be extracted in order to extract more complex meaning from the data and that a different bias term as mentioned in the first paragraph of Section 5.2 may change the result significantly. That is if numeric values dominate the result, the influence that the labels (nominal values) have decreases in comparison to the structural features which are numeric. This also applies for properties (mixed) and the characteristic set (nominal).

For the property oriented yelp data set we have again no meaningful information gain as it does not contain any relationships.

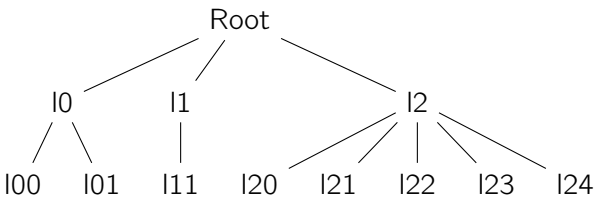

Figure 20 *Inferred Label Hierarchy for the New York road network data set with labels and structural features as the feature vector*

For the road network example the situation looks a bit different now: Crossings connecting many streets and those who only connect two streets are separated as well as streets in more or less dense networks. The sub-tree l0 contains all crossings having exactly 3 roads that are connected. It further sub-divides into crossings whose ego nets have between 17 (l03) and 23 (l01) incoming and outgoing roads. Sub-tree l1 contains all crossings that connect exactly 1 road (dead ends). Last but not least sub-tree l3 contains crossings connecting either 2 or 4 roads with 15 and 31 incoming and outgoing roads respectively. The first three levels of the generated tree is shown in Figure 20.

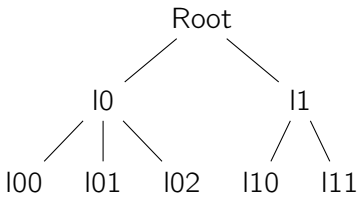

Figure 21 *Inferred Label Hierarchy for the SNB data set with labels, structural features and the characteristic set as feature vector*

Labels and mixtures of the other features When taking labels, structural features and the characteristic set as the feature vector the resulting hierarchy is the same, if only the structure of the hierarchy is considered. The first three levels of the generated tree is shown in Figure 21. Semantically at the sub-tree l1 nodes with the labels Message and comment are split as for most of the example hierarchies before. The split happening after l1 separates nodes with a degree of exactly 3 (std is 0.027) from higher degree nodes with a mean degree of 9. This indicates the number of tags of the comment: The higher the degree, the more tags are attached, which is also visible from significant differences in the probabilities of the characteristic set containing the HAS_TAG relationship (99.5% for the higher degree nodes versus 0.35% for the lower degree nodes). The sub-tree l0 separates again the rest of the nodes and the children sub-divide into nodes with labels Message and Post (l01, l02) and the rest (l00). The concepts l01 and l02 separate posts with a high degree and a low degree indicating more or less tags and replies as with comments. This time the probabilities are $3,8\%$, 0.21%, 0% versus 17%, 89.7%, 85.2% for likes, tags and replies respectively.

Taking the properties additionally into account introduces more noise, that is more features of for which the information gain is unknown and might be negative (that is adding the information to the feature vector actually interferes with the algorithm's ability to extract the relevant concepts with the correct attribute value distribution from the underlying data distribution). The first three levels of the generated tree is shown in Figure 22. Here l0 splits again between nodes with labels message comment but in the subsequent step the predictiveness of using a certain browser is maximized which is a characteristic of the data which does not carry any further information about structure and distribution of the data. In the other sub-tree again nodes with

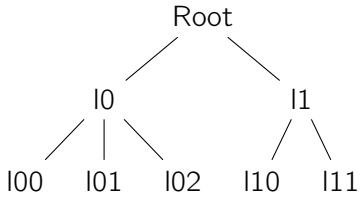

Figure 22 *Inferred Label Hierarchy for the SNB data set with labels, node properties, structural features and the characteristic set as feature vector*

the labels message and post are split from the nodes containing all other labels.

5.2.3 Discussion

None of the generated concepts or label hierarchies do match fully to the ground truth, but this is also not what is desired. Desired is to extract meaningful and representative concept classes that can be used to derive labels. For example the LDBC data set contains about 95% nodes with the label "Message" and only a very small amount of nodes with the label "Person". Therefore it is representative to split at the first steps into "Message and non-Message"-labled nodes as this is the major difference between the largest group of nodes in this data set and all the others.
The results clearly indicate that there is no single feature vector which can be used to extract meaningful concepts from all data sets. For data sets not leveraging graph structures, the characteristic set and structural features contain no information like in the non-transformed yelp data set. For those which do not have properties attached to the data set, taking the properties into account yields no information. Road network data like the New York road net data set contain often no labels, that is taking the labels into account in the feature vector adds no information.
On the other hand, adding the properties to the feature vector may introduce additional noise and actually disturb the derivation of meaningful concepts that represent the underlying distribution. Constructing the feature vector adaptively seems to be the only option resulting in a minimal and complete source of information on the data set. That is if there are no relations but properties, only the properties and eventually the labels when meaningful shall be used. On the other hand for data sets not containing properties, useful labels and more than one relationship type, structural features are the only usable feature that should be used and so on.
Also extensions to the algorithms to make the procedure detect significant correlations, deal with temporal and spatial data types (and possibly others requiring special semantics to yield useful aggregations not considered here) and the ability to only use those attributes with the highest information gain. In general it seems that both a Bayesian and an information theoretic extension of the conceptual clustering framework may counter some of the problems. Applied to the other clustering algorithms such extensions would require additional pre-processing, for instance a sampling procedure to estimate correlations and to calculate the information gain for certain attributes or groups of correlated attributes.
As the distributions that shall be learned are only constrained to model numeric values as Gaussians, there is no upper bound for the sample complexity, due to the no free lunch theorem. Further when loosening the Gaussian prior which has no concrete theoretical justification the space of possible distributions is unrestricted.

CHAPTER 6

Conclusion

6.1 Summary

In this thesis, a pipeline was designed in order to conduct a survey, comparing hierarchical agglomerative, density-based, partition-based and conceptual state-of-the-art clustering. More practically relevant problems with those approaches include parameter inference, space complexity and empty cluster descriptions (that is the set of all common attributes of the instances in the cluster) for noisy domains. Those could be overcome by introducing a more sophisticated parameter inference mechanism as randomized search e.g. stochastic variational inference [30], by implementing the computation of the distance matrix and the algorithms using it in a fashion similar to the two-way merge sort algorithm, i.e. only using a few buffer pages keeping in ram only what is necessary for a batch of iterations and by introducing a probabilistic concept description using a fuzzy intersections for determining the cluster description. A model-based hierarchical framework namely conceptual clustering provided superior abilities regarding some characteristics, especially that the algorithm is to some degree able to deal with mixed data, has linear space complexity, is incremental and produces hierarchies by design instead of the multi-phase approach requiring additional post-processing as for the other algorithm classes.
This approach was used in the second part, where a graph-aware version of the previous pipeline was introduced that leverages graph-based features of the graph. Different feature vectors were evaluated, concluding that finding the features extracted have to be chosen adaptively. For example it is desirable to look at the labels and structural features only in case if there are no properties and all relationships have the same type. In the case when there are no relationships one might want to consider the properties of the node and the label as feature vector (i.e. no structural features as they contain no information) and so on.

6.2 Future Work

Generally there are three categories of future work

1. Incorporate extensions to the conceptual clustering framework
2. Find better feature vectors and ways to construct them adaptively
3. Apply conceptual clustering or multi-phase clustering with adaptive feature vectors to other domains

There are a lot of extensions to the conceptual clustering framework. Cheeseman [16] integrates the Bayesian interpretation more concretely into the method. Gennari [24] reduces the number of nodes in the tree by storing pointers to the data instances instead of nodes and probabilistic concepts in the tree such that the leaves are rather the elements of a concept class than concepts themselves. Thompson et al. [67] takes structured domains into account and introduces internal links in the hierarchy. That is if an object from a different class is contained in the description, it may be linked and treated as entity in the hierarchy (that is a sub-concept is also part of the super-concept). The mathematical field of formal concept analysis provides a concise mathemat-

ical foundation for the framework, which Godin et al. [26] synthesised into the construction of the concepts using Gallois lattices. Xie et al. [73] suggests the usage of the of concept lattices for classification which is similar to what Godin et al. [26] proposes. Devaney et al. [18] provide an extension that focuses on the efficient selection of features. Fisher [21] provides a series of improvements, that may be applied to select only features that assist the optimization process to avoid noisy concept descriptions and methods to refine the tree after an instance or all instances have been incorporated like restructuring the hierarchy or cleaning some concept descriptions. Many other improvements can be thought of like the detection of correlated features after incorporating new instances. Even tough conceptual clustering has not received much attention since the 90s it remains a field of research.

Finding better feature vectors includes not only the invention of better features but also the selection of the significant features and pruning of unnecessary or grouping of highly correlated features. Depending on the application use case, the adaption strategy may be far from trivial. When the features are not human readable but generated as an intermediate or internal representation as the filters that are learned during the training of a neural net, the selection of features not only from the top level but through out the network with keeping at the same time a modest number of features is an example for a non-trivial adaption procedure.
In the database environment, many products already contain a profile that is used for cardinality optimization which can give concrete guidelines for the features to use or not to use.

Cardinality estimation for query optimization has been an ongoing research topic since the invention of databases. Many different methods have been invented and applied, most of which can be divided in 2 general approaches:

- Database Profile: Keep statistics about the value distributions to estimate cardinality (e.g. using general statistics about the relations, histograms, results of previous queries, ...)
- Sampling techniques: Execute the query with a sample of the considered relationships to estimate cardinalities

The first approach assumes uniformity and independence assumptions when using general statistics and tries to mitigate inaccuracies by those assumptions with histograms, which accuracy depends on the chosen bucket size and which are not able to capture correlations between different attributes. Those two approaches have also been adapted to graph-based data models. Neumann et al. propose characteristic sets, that is the set containing all relationship types, predicates, labels or however they are called in the implementation to estimate cardinalities [54]. Neo4j, the "leading" graph database according to it's homepage, uses a database profile with the following statistics according to the operations manual [1]:

1. The number of nodes having a certain label.
2. The number of relationships by type.
3. The number of relationships by type, ending or starting from a node with a specific label.
4. Selectivity per index.

Thus they make use of the uniformity and independence assumption and rely on indices to provide information about property value information per label [2]. Further the Query Planer of Neo4j can only use one index at a time and picks a sub-optimal index sometimes [3]. Those and other weaknesses in cardinality estimation may be overcome by explicitly storing taxonomies of the data implicated in the database. Taxonomies capture differences in value distributions for similar data instances and assign those to categories which may then be used not only for cardinality estimation but also for data exploration, profiling and analysis. A feature vector that is adapted according to the already existing database profile may greatly improve the results, capturing concrete attribute-value distributions for different compositions and correlations. As Cobweb is incremental this may be implemented as a dynamic (update-able) index which is then used by the query optimizer during cardinality estimation.

Another interesting application is that of using taxonomies of sub-graphs as self-organizing associative index, that segments the physical storage of the data base to optimize query times in a molecule database [41].

Other use cases include cognitive architectures [39] and the modelling of human learning behaviour [43].

Appendix

A Other Approaches to Clustering

Other approaches to clustering include:

- **Grid-based methods** construct a grid on the space to cluster that may be scaled to produce clusters in multiple resolutions. Most grid-based methods have a fast processing time as they are typically independent of the number of data objects but rather on the space the objects span.
- **Bi-clustering methods** sometimes also referred to as subspace clustering clusters the objects and one class of attributes - in other words rows and columns of a matrix - at the same time or in the terms of Formal Concept Analysis all attributes of in the same scaling context. It is closely related to formal concept analysis as described by[31] and may be thought of as grouping both the columns and the rows of a table concurrently. It is a very common technique in clustering gene expression data [58]. An example of is given in fig. 23.

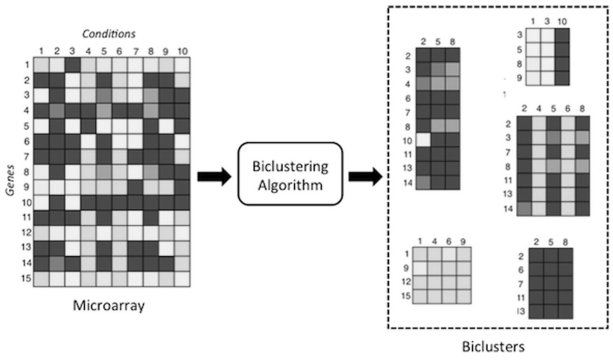

Figure 23 *An example of biclustering on gene expression data [58].*

- **Evolutionary or genetic methods** are search-based, searching the space by
    1. applying random alternations (mutation) or combinations of previous clusters to generate new clustering candidates.
    2. evaluating the so generated clustering candidates against the objective function.
    3. selecting a subset of the evaluated candidates, possibly introducing randomness and other tools to sample from different regions of the search space in order not to converge into a local minimum.
    4. iterate until there is convergence or stopping criteria are satisfied.

  All of these steps may be adapted to a certain setting. It is a common approach to combine evolutionary algorithms with other techniques e.g. self-organizing maps with evolutionary algorithms [40]. Generally neural networks and evolutionary algorithms or similar constructed Markov decision processes (reinforcement learning) are often combined into neural architecture search systems, using stochastic processes to select and vary the al-

ready evaluated networks to generate new ones with faster convergence rates to better AUC-scores[34].
Evolutionary algorithms may be seen as Markov chains, as the Markov property holds (each generation only depends on the last one) and each generative operator creates a new candidate solution with each candidate solution having a certain probability depending on the nature of the operation and the probabilities of all generateable candidate solutions of 1 per operator and generation [55].

B An Introduction to Probability Theory

Probability theory is essential to understand and implement probabilistic model-based approaches like COBWEB [22] and will be used in chapters 3 to 5.
There are many textbooks extensively defining the notations needed in probability theory [8, 20]. The following is a summary of the most important terms used in the latter chapters.

A **sample space** Ω is the set of all possible atomic results or outcomes of an experiment. An **event E** is a subset of the sample space, i.e. a set of elementary results or outcomes. As an example the a match day of a soccer league. Each match day 2n teams compete in n matches. The sample space would be all possible results $\Omega = \{(i,j)|\forall i,j \in \mathbf{N}\}$. An event would e.g. be the set of results of a match day or a partial result.

A **σ-algebra** on sample space Ω is a collection of events $\mathfrak{E} \subseteq 2^\Omega$ is a pair $(\Omega, \mathfrak{E})$ with

1. $\Omega \in \mathfrak{E}$
2. $E \in \Omega : E \in \mathfrak{E} \Rightarrow \Omega \setminus E \in \mathfrak{E}$
3. $E_0, E_1, \cdots \in \mathfrak{E} \Rightarrow E_0 \cup E_1 \cup \cdots \in \mathfrak{E}$

The pair $(\Omega, \mathfrak{E})$ is then called measurable space.

A **probability space** $\mathcal{P} = (\Omega, \mathfrak{E}, P)$ is a structure with

1. $(\Omega, \mathfrak{E})$ is a σ-Algebra
2. $\forall x \in \Omega : 0 \leq P(x) \leq 1$
3. $P(\Omega) = 1$
4. $\forall i \geq 0 \wedge i \neq j : E_i \cap E_j = \emptyset : P(\cup_i E_i) = \sum_i P(E_i)$

P is called the **probability measure**. A **discrete probability space** is a probability space $(\Omega, \mathfrak{E}, P)$ where

$$\exists E \subseteq \Omega : \forall e \in E : \{e\} \in \mathfrak{E} \wedge \sum_{e \in E} P(\{e\}) = 1$$

If the probability space is not discrete, it is called **continuous**.

A **measurable function** $f : \Omega_0 \to \Omega_1$ is a function that maps a sample space of a measurable space $(\Omega_0, \mathfrak{E}_0)$ to another sample space of another measurable space $(\Omega_1, \mathfrak{E}_1)$ with:

$$\forall E \in \mathfrak{E} : f^{-1}(E) = \{e|f(e) \in E\} \in \mathfrak{E}$$

A **random variable** $X : \Omega \to \mathbf{R}$ is a measurable function that maps a sample space to the real numbers.
The **Probability distribution** P_X of X is given by

$$P_X = P \circ X^{-1}$$

A **distribution function** F_X, also called probability mass function for discrete probability space or cumulative density function for continuous probability spaces of a random variable X is given by

$$x \in \mathbf{R} : F_X(x) = P(\{e \in \Omega | X(e) \leq x\})$$

For the discrete case it can be formulated as:

$$x \in \mathbf{R} : F_X(x) = \sum_{x_i \leq x} P_x(X = x_i)$$

In the continuous case with f the probability density function:

$$x \in \mathbf{R} : F_X(x) = \int^d f_X(x) dx$$

C Pseudo-Code for Chapter 3: Algorithms

C.1 Hierarchical Clustering

C.1.1 Hierarchical Agglomerative Clustering

Algorithm 4: Hierarchical Agglomerative Clustering

Input: A distance matrix dM of shape $|O| \times |O|$
Parameters: Distance function d
Output: A linkage matrix IM

```
begin
    List cluster → initializeObjectsAsOwnCluster(M);
    while cluster.size() > 2 do
        m1, m2 → argmin_{r1,r2∈dM} d(r1, r2);
        cluster → (cluster \ m1 \ m2) ∪ (m1 ∪ m2) dM →
          computePairwiseDistance(cluster);
    return cluster;
```

As we need to compute the pairwise (symmetric) distance matrix which is a square matrix the space complexity of this algorithm is in $\frac{1}{2}n^2 \in \mathcal{O}(n^2)$. The time complexity is $\mathcal{O}(n^3)$ for a naive implementation as in the pseudo-code above. In each iteration we need to scan the distance matrix for the minimum and update it after merging resulting in $2 \cdot \frac{1}{2}n^2$ steps per iteration. As in each step exactly 2 objects are merged, there are $n-1$ merges in total, resulting in a time complexity of $n-1 \cdot n^2 \in \mathcal{O}(n^3)$.

C.1.2 Robust Single Linkage

The space complexity is again the space needed to store the symmetric pairwise distance matrix $\frac{1}{2}n^2 \in \mathcal{O}(n^2)$. The time complexity here is a bit different: Computing the smallest radius for which the sphere around the object contains at least k points is $n \cdot (n-1)$ as for every object, the distance to all other points needs to be compared to r - in the most naive implementation against $0, 1 \ldots$ until a r is found that leads to $\{o_j | \, |B(o_i, r)| \geq k$. Thus the first for loop has a complexity of $n \cdot r \cdot (n-1) \in \mathcal{O}(n^2)$. The While loop constructs a graph that needs to traverse only the list of infimum r values for the vertices (i.e. n elements). Regarding the edges, the distance matrix

Algorithm 5: Robust Single Linkage Clustering

Input: A distance matrix $distM$ of shape $|O| \times |O|$
Parameters: k the number of neighbouring points, α the inverse density
Output: A linkage matrix IM

begin
  Stack Clustering = new Stack;
  for $o_i \in O$ **do**
    $r_k[o_i] = inf(\{r|k \leq |B(o_i, r)|\})$;
  $r \to 0$;
  while *Clustering.peek().size()*> 1 **do**
    Construct $G_r = (V_r, E_r)$ with $V_r = \{x_i \in X | r_k(x_i) \leq r\}$ and
     $E_r = \{i \neq j : (x_i, x_j) \in X \times X | \ ||x_i - x_j|| \leq \alpha r\}$;
    Clusterings.push(connectedComponents(G_r));
    $r++$;
  return Clustering;

needs to be traversed i.e. $\frac{1}{2}n^2$ elements. As α is user defined, the number of iterations is variable but is less than $(n-1)$ for $\alpha = 1, k = 2$ (compare with Agglomerative clustering). To conclude, for $\alpha \geq 1$ and $k \geq 2$ convergence is faster than single linkage as many points are merged in one step. For a more comprehensive discussion of the rates of convergence, see [15]

C.2 Partition-based Methods

C.2.1 K-Means

Algorithm 6: k-Means Clustering

density based spatial clustering **Input:** A Set of data objects O
Parameters: k the number of clusters
Output: A set of k clusters

begin
  for $i = 0, \ldots, k-1$ **do**
    Centroid $c_k \leftarrow$ Chose k objects at random;
  while *Objective function did not converge* **do**
    for $o_i \in O \setminus \{c_1, \ldots c_k\}$ **do**
      Centroid closestCentroid = $\{c_i \in \{c_1, \ldots, c_k\} | \text{argmin}_{c_i \in \{c_1, \ldots, c_k\}} d(o_i, c_i)\}$
       Cluster(o_i) = Cluster(closestCentroid);
    for $C \in$ *Clusters* **do**
      Update Centroids to the means of all objects in the cluster per feature;
  return Clusters;

The space complexity is simply the number of data objects $n \cdot k \in \mathcal{O}(n)$ for small k as the distance for each object to the cluster centroid must be stored instead of the full distance matrix. The time complexity is with I the number of iterations, k the number of clusters and n the number of objects $n \cdot k \cdot I \in \mathcal{O}(n)$ for small k and I, as in each iteration we need to find the closest centroid to the object for each object, thus $n \cdot k$.

C.2.2 TTSAS

Algorithm 7: Two-Threshold Sequential Algorithmic Scheme Clustering

Input: A Set of data objects O
Parameters: Θ_1 the maximal distance threshold, Θ_2 the minimal distance threshold
Output: A set of k clusters

```
begin
    for o_i ∈ O do
        categorized[o_i] = 0;
    k = 0 ;                                          // The number of clusters
    prev = 0 ;                                     // previous iteration's change
    curr = 0 ;                                      // current iteration's change
    exist = 0 ;                                                // overall change
    while ∃o_i ∈ O : categorized[o_i] == 0 do
        for i = 1 to N do
            if categorized[o_i] == 0 ∧ exist== 0 ∧ o_i is the first element in the new iteration
              of the while loop then
                k ← k + 1;
                C_m = {o_i};
                categorized[o_i] = 1;
                curr = curr + 1;
            else if categorized[o_i] == 0 then
                min_dist, min_dist_cluster = {d(o_i, C_i), C_i|argmin_{C_i∈{C_0,...,C_{k−1}}} d(o_i, C_i)};
                if min_dist < Θ_1 then
                    min_dist_cluster = min_dist_cluster ∪{o_i};
                    categorized[o_i] = 1;
                    curr = curr + 1;
                else if min_dist > Θ_2 then
                    k ← k + 1;
                    C_m = {o_i};
                    categorized[o_i] = 1;
                    curr = curr + 1;
            else if categorized[o_i] == 1 then
                curr = curr + 1;
        exists_change = |current_change - previous_change|;
        previous_change = current_change;
        current_change = 0;
    return clusters;
```

In algorithm7 the first condition guarantees that the algorithm terminates. If the current object is uncategorized and there was no change in the last iteration and the current object is the first that is inspected in this iteration of the while loop, make it a new cluster. This results in a worst-case complexity of $n \cdot 2nk$ with $|O| = n$ and k the final number of clusters after algorithm execution. in this case every object's minimal distance is between the thresholds for all clusters that are created and every point ends up in an own cluster, taking two iterations per element: The first where no change happens and a second where only one point is assigned as a new cluster and no further changes happen. For each element (n) do 2 complete iterations ($n \cdot 2n$), where for each element the distance to all so far existing clusters is queried ($n \cdot 2nk$), resulting in

a worst-case run time of $2n^2k \in \mathcal{O}(n^3)$ as in this case $k = n$. In the best case $k << n$ and only two passes [65] resulting in $2 \cdot n \cdot k \in \mathcal{O}(n)$ iterations. The space complexity is as in k-Means plus n to store the categorized vector.

C.3 Density-based Methods

C.3.1 DBSCAN

Algorithm 8: Expand Cluster method

Input: A Set of data objects O, o_i the currently expanded point, cluster[] the array storing the object to cluster mapping

Parameters: ϵ the radius of the neighbourhood, MinObjects the minimal number of neighbours of a core point, cluster_id the id of the cluster to be used

Output: A boolean indicating weather a new cluster was formed

```
begin
    seeds ← {o_j ∈ O \ {o_i} | d(o_i, o_j) ≤ ε};
    if seeds.size < MinObjects then
        return False;
    else
        for seed in seeds do
            cluster[seed] = cluster_id;
        while seeds ≠ ∅ do
            currentSeed ← seeds.pop();
            currentSeedNeigh = ← {o_j ∈ O \ {currentSeed} | d(o_i, o_j) ≤ ε};
            if currentSeedNeigh.size ≥ MinObjects then
                for i = 1, ..., currentSeedNeigh.size do
                    currentResult = currentSeedNeigh[i];
                    if cluster[currentResult] == -1 ∨ cluster[currentResult] == 0 ;
                     // i.e. currentResult is unclassified or noise
                     then
                        cluster[currentResult] = cluster_id;
                        if cluster[currentResult] == -1 ;          // unclassified
                         then
                            seeds.append(currentResult);
        return True;
```

The space complexity of DBSCAN is $\frac{1}{2}n^2$ as the pairwise distance matrix needs to be computed for neighbourhood queries. The run time complexity is n for the outer loop in the DBSCAN algorithm and $\log(n)$ for a neighbourhood query if the neighbourhood information is stored in an efficient data structure e.g. a R*-tree according to the authors [19]. In total this induces a run time complexity of $n \log(n) \in \mathcal{O}(n \log(n))$. If no such structure is used, a neighbourhood query needs to scan a row (or column) of the matrix which takes n elements to visit, resulting in $n \cdot n \in \mathcal{O}(n^2)$ [27].

Algorithm 9: Density-based Spatial Clustering of Applications with Noise

Input: A Set of data objects O

Parameters: ϵ the radius of the neighbourhood, MinObjects the minimal number of neighbours of a core point

Output: A set of k clusters

```
begin
    for o_i ∈ O do
        cluster[o_i] ← -1 ;            // All objects are unclassified at the start
    cluster_id = nextId(NOISE);
    for o_i ∈ O do
        if cluster[o_i] == -1 then
            if expand_cluster(O, o_i, cluster, cluster_id, ε, MinObjects) then
                cluster_id = nextInt(cluster_id);
```

C.3.2 OPTICS

Algorithm 10: Expand Cluster Order method

Input: A Set of data objects O, o_i the currently expanded point

Parameters: ϵ the radius of the neighbourhood, MinObjects the minimal number of neighbours of a core point, OrderedFile the file to write the cluster ordering to

Output: None

```
begin
    seeds ← {o_j ∈ O \ {o_i} | d(o_i, o_j) ≤ ε};
    o_i.processed ← 1;
    o_i.reachability_distance = UNDEFINED;
    o_i.setCoreDistance(seeds, ε, MinObjects);
    OrderedFile.write(o_i);
    if o_i.core_distance ≠ UNDEFINED then
        OrderSeeds.update(seeds, o_i);
        while OrderSeed ≠ ∅ do
            currentSeed ← seeds.next();
            seeds = ← {o_j ∈ O \ {currentSeed} | d(o_i, o_j) ≤ ε};
            currentSeed.preprocessed = 1;
            currentSeed.setCoreDistance(seeds, ε, MinObjects);
            OrderedFile.write(currentSeed);
            if currentSeed.core_distance ≠ UNDEFINED then
                OrderSeeds.update(seeds, currentSeed);
```

As the algorithm proceeds very similar to DBSCAN a lot of the complexity analysis also applies here: The space complexity is $\frac{1}{2}n^2$ for the pairwise distance matrix plus additionally the file that is written to, which size depends on the number of clusters but is generally bound by $3n$ (the object plus two distances per object) what yields $\frac{1}{2}n^2 + 3n \in \mathcal{O}(n^2)$. Regarding the computational complexity, the authors provide $\mathcal{O}(n^2)$ as run time without a special data structure and $\mathcal{O}(n \log(n))$ with a specialised data structure[6]. The additional overhead for finding the appropriate core distance may be solved with a one time pass over the respective row or column using dynamic programming, i.e. besides querying the neighbour hood to find the seeding neighbours

Algorithm 11: OrderSeeds.update method

Input: neighbours a set of neighbour objects, o_i the current center object
Parameters: None
Output: None

```
begin
    c_dist = o_i.core_distance;
    for o_j ∈ neighbours do
        if o_j.processed == 0 then
            new_r_dist = max(c_dist, o_i.dist(o_j));
            if o_j.reachability_distance == UNDEFINED then
                o_j.reachability_distance = new_r_dist;
                insert(o_j, new_r_dist);
            else
                if new_r_dist < o_j.reachability_distance then
                    o_j.reachability_distance = new_r_dist;
                    decrease(o_j, new_r_dist);
```

Algorithm 12: Ordering Points To Identify Cluster Structure

Input: A Set of data objects O
Parameters: ϵ the radius of the neighbourhood, MinObjects the minimal number of neighbours of a core point, OrderedFile a file to write to
Output: An Ordering of the clusters in the database with different densities

```
begin
    OrderedFile.open();
    for o_i ∈ O do
        o_i.processed ← 0;
    for o_i ∈ O do
        if processed[o_i] == 0 then
            expand_cluster_order(O, o_i, ε, MinObjects, OrderedFile);
    OrderedFile.close();
```

another pass is needed to find the core distance.

C.3.3 HDBSCAN

Algorithm 13: Hierarchical Density-based Spatial Clustering of Applications with Noise

Input: A Set of data objects O
Parameters: MinObjects the minimal number of neighbours of a core point
Output: A dendrogram

begin
  for $o_i \in O$ **do**
    compute the core distance wrt. MinObjects;
  Build the mutual reachability graph $G_{\text{MinObjects}}$;
  Build the minimum spanning tree of $G_{\text{MinObjects}}$;
  Add a self-edge for each vertex v_{o_i} in the minimal spanning tree with weight $d_{\text{core}}(o_i)$;
  Assign all objects the same label ; `// this forms the root of the dendrogram`
  while $G_{MinObjects}$ *has edges* **do**
    Set the dendrogram scale value of the current level in the hierarchy to the largest edge weight which is to be removed;
    Remove the largest edge(s);
    Assign a label to the connected component, that contained the end vertices of the removed edge(s);
    If the end vertices are not in a connected component assign the noise label to those;

The space requirements are the same as for DBSCAN: The pairwise distance matrix needs $\frac{1}{2}n^2 \in \mathcal{O}n^2$ space complexity. regarding the run time, one needs to compute the core-distance for each object which takes $n \cdot n \in \mathcal{O}(n^2)$. Constructing the minimum spanning tree using Prim's algorithm or Boruvka's algorithm is in $\mathcal{O}(n^2)$ worst-case run time complexity. Removing all edges iteratively requires maximally $\frac{1}{2}n^2$ iterations for a fully connected graph and when sorting the matrix before starting the removal procedure which takes $\mathcal{O}(n \log n)$ steps retrieving the maximum is a constant time operation. Overall we get $n^2 + n^2 + n \log n + \frac{1}{2}n^2 \in \mathcal{O}(n^2)$ steps as run time complexity.

C.4 Conceptual Clustering

C.4.1 Cobweb

With b the average branching factor, n the number of objects classified so far, A the number of all distinct attributes, V the average number of distinct values per attribute for categorical values. When incorporating a new object, at each level the node counts must be updated, the category utility must be computed for each child to find the host, split merge and creation must be probed. That is when computing the category utility takes $\mathcal{O}(bAV)$ steps, finding the host is $b \cdot bav$, adding the probes for merging (finding another host and computing the category utility for the operation means b+1 computations of the category utility, splitting and class creation is one time computing the category utility resulting in $(b+b+1+1+1) \cdot bAV = (2b+3) \cdot bAV = 2b^2AV + 3bAV \in \mathcal{O}(b^2AV)$. As the depth of the tree is $\log_B(n)$ and we need to incorporate every instance we get $n \cdot \log_b(n) \cdot \mathcal{O}(b^2AV) \in \mathcal{O}(n \cdot \log(n) \cdot b^2AV)$. In the most naive implementation each data object needs to be stored as a leaf and additional $\frac{n}{b}$ nodes are needed to build the tree,

resulting in a space complexity of $n + \frac{n}{b} \in \mathcal{O}(n)$.

Algorithm 14: COBWEB

Input: An object $o_i \in O$
Parameters: node the node currently visiting
Output: A hierarchy of clusters with concept descriptions

```
begin
    updateCounts(node, o_i);
    if node is a leaf then
        node.add_child(o_i);
    else
        host, host_cu = find the child of node, that best hosts o_i;
        new_class_cu = probeCreateNewClass(o_i);
        merge_cu = probeMerge(host, o_i);
        split_cu = probeSplit(host);
        max_cu = max(new_class_cu, merge_cu, split_cu, host_cu);
        if max_cu == new_class_cu then
            createNewClass(o_i);
        else if max_cu == merge_cu then
            merged_node = merge(host, o_i);
            COBWEB(o_i, merged_node);
        else if max_cu == split_cu then
            split(host);
            COBWEB(o_i, node);
        else
            COBWEB(o_i, host)
```

D Example Taxonomy Extraction from a Dendrogram

$C_{14} = C_0 \cup C_2$, with distance 1
$C_{15} = C_1 \cup C_{14}$, with distance 1
$C_{16} = C_3 \cup C_{15}$, with distance 1
$C_{17} = C_5 \cup C_7$, with distance 1
$C_{18} = C_6 \cup C_{17}$, with distance 1
$C_{19} = C_4 \cup C_{18}$, with distance 1
$C_{20} = C_8 \cup C_{19}$, with distance 1
$C_{21} = C_9 \cup C_{11}$, with distance 1
$C_{22} = C_{12} \cup C_{21}$, with distance 1
$C_{23} = C_{13} \cup C_{22}$, with distance 1
$C_{24} = C_{16} \cup C_{20}$, with distance 2
$C_{25} = C_{23} \cup C_{24}$, with distance 2
$C_{26} = C_{10} \cup C_{25}$, with distance 2

is transformed to

$C_{16} = C_0 \cup C_2 \cup C_1 \cup C_3$, with distance 1
$C_{20} = C_5 \cup C_7 \cup C_6 \cup C_4 \cup C_8$, with distance 1
$C_{23} = C_9 \cup C_{11} \cup C_{12} \cup C_{13}$, with distance 1
$C_{26} = C_{16} \cup C_{20} \cup C_{23} \cup C_{19}$, with distance 2

```
[
    [ 0.,  2.,  1.]
    [ 1., 14.,  1.]
    [ 3., 15.,  1.]
    [ 5.,  7.,  1.]
    [ 6., 17.,  1.]
    [ 4., 18.,  1.]
    [ 8., 19.,  1.]
    [ 9., 11.,  1.]
    [12., 21.,  1.]
    [13., 22.,  1.]
    [16., 20.,  2.]
    [23., 24.,  2.]
    [10., 25.,  2.]
 ]
```

```
[
    [[ 0.,  2.,  1.,  3.], 1]
    [[ 5.,  7.,  6.,  4.,  8.], 1]
    [[ 9., 11.,  12.,  13. ], 1]
    [[16.,  20.,  23.,  10.], 2]
]
```

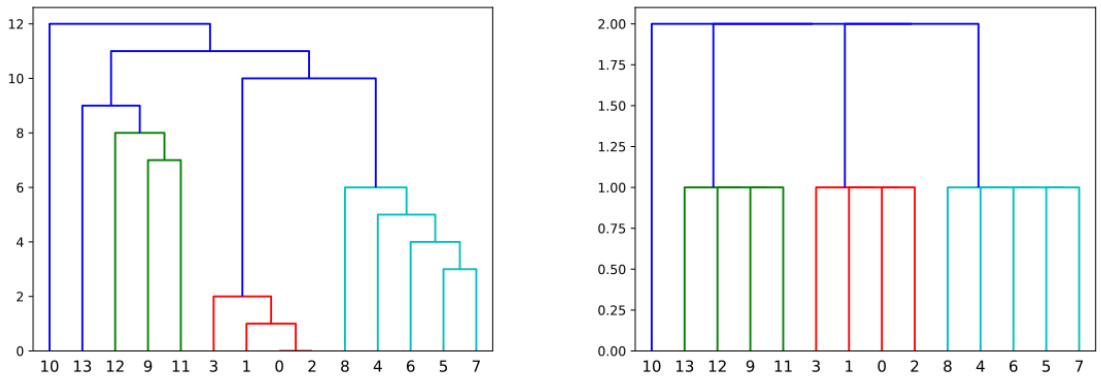

Figure 24 *The Input and output of the taxonomy extraction heuristic*

E Example Concept Descriptions

E.1 Labels only

Attribute	ValueType	Value	Probability	Occurrences
Labels	Nominal	Message	0.9615	1923
	Nominal	Comment	0.6445	1289
	Nominal	Post	0.3170	634
	Nominal	Forum	0.0285	57
	Nominal	Person	0.0030	6
	Nominal	Tag	0.0035	7
	Nominal	Company	0.0015	3
	Nominal	Organisation	0.0030	6
	Nominal	University	0.0015	3
	Nominal	Place	0.0005	1
	Nominal	City	0.0005	1

Table 8 *SNB Labels Only: ConceptNode l, P(node) = 1.0, Count 2000*

Attribute	ValueType	Value	Probability	Occurrences
Labels	Nominal	Message	1.0000	1289
	Nominal	Comment	1.0000	1289

Table 9 *SNB Labels Only: ConceptNode l0, P(node) = 0.6445, Count 1289*

Attribute	ValueType	Value	Probability	Occurrences
Labels	Nominal	Forum	0.0802	57
	Nominal	Message	0.8917	634
	Nominal	Post	0.8917	634
	Nominal	Person	0.0084	6
	Nominal	Tag	0.0098	7
	Nominal	Company	0.0042	3
	Nominal	Organisation	0.0084	6
	Nominal	University	0.0042	3
	Nominal	Place	0.0014	1
	Nominal	City	0.0014	1

Table 10 *SNB Labels Only: ConceptNode l1, P(node) = 0.3555, Count 711*

E.2 Labels, Structural Features and Characteristic Set

Attribute	ValueType	Value	Probability	Occurrences
Labels	Nominal	Message	1.0000	634
	Nominal	Post	1.0000	634

Table 11 *SNB Labels Only: ConceptNode l10, P(node) = 0.8917018284106891, Count 634*

Attribute	ValueType	Value	Probability	Occurrences
Labels	Nominal	Forum	0.7403	57
	Nominal	Person	0.0779	6
	Nominal	Tag	0.0909	7
	Nominal	Company	0.0390	3
	Nominal	Organisation	0.0779	6
	Nominal	University	0.0390	3
	Nominal	Place	0.0130	1
	Nominal	City	0.0130	1

Table 12 *SNB Labels Only: ConceptNode l11, P(node) = 0.10829817158931083, Count 77*

Attribute	ValueType	Value	Probability	Occurrences
RelationshipTypes	Nominal	IS_LOCATED_IN	0.9635	1927
	Nominal	CONTAINER_OF	0.3325	665
	Nominal	HAS_CREATOR	0.9615	1923
	Nominal	REPLY_OF	0.7150	1430
	Nominal	HAS_TAG	0.3275	655
	Nominal	LIKES	0.0430	86
	Nominal	HAS_MEMBER	0.0290	58
	Nominal	HAS_MODERATOR	0.0335	67
	Nominal	HAS_TYPE	0.0045	9
	Nominal	HAS_INTEREST	0.0045	9
	Nominal	KNOWS	0.0010	2
	Nominal	WORK_AT	0.0010	2
	Nominal	STUDY_AT	0.0010	2
	Nominal	IS_PART_OF	0.0005	1
EgoDegree	Numeric	mean= 9.191, std=74.672	1.0000	2000
EgoNetOutgoingEdges	Numeric	mean= 1527.115, std=5897.396	1.0000	2000
Labels	Nominal	Message	0.9605	1921
	Nominal	Post	0.3035	607
	Nominal	Comment	0.6570	1314
	Nominal	Forum	0.0320	64
	Nominal	Tag	0.0045	9
	Nominal	Organisation	0.0010	2
	Nominal	University	0.0010	2
	Nominal	Person	0.0015	3
	Nominal	Place	0.0005	1
	Nominal	City	0.0005	1
AverageNeighbourDegree	Numeric	mean= 42028.168, std=54456.000	1.0000	2000
EgoNetIncomingEdges	Numeric	mean= 158689.547, std=180363.031	1.0000	2000

Table 13 *LDBC SNB Lables, Structural Features & Characteristic Set: ConceptNode I, P(node) = 1.0, Count 2000*

Attribute	ValueType	Value	Probability	Occurrences
RelationshipTypes	Nominal	IS_LOCATED_IN	0.8936	613
	Nominal	CONTAINER_OF	0.9679	664
	Nominal	HAS_CREATOR	0.8878	609
	Nominal	REPLY_OF	0.1691	116
	Nominal	HAS_TAG	0.2843	195
	Nominal	HAS_MEMBER	0.0845	58
	Nominal	HAS_MODERATOR	0.0977	67
	Nominal	HAS_TYPE	0.0131	9
	Nominal	HAS_INTEREST	0.0131	9
	Nominal	LIKES	0.0641	44
	Nominal	KNOWS	0.0029	2
	Nominal	WORK_AT	0.0029	2
	Nominal	STUDY_AT	0.0029	2
	Nominal	IS_PART_OF	0.0015	1
EgoDegree	Numeric	mean= 16.792, std=126.929	1.0000	686
EgoNetOutgoingEdges	Numeric	mean= 2424.458, std=9137.503	1.0000	686
Labels	Nominal	Message	0.8848	607
	Nominal	Post	0.8834	606
	Nominal	Forum	0.0933	64
	Nominal	Tag	0.0131	9
	Nominal	Organisation	0.0029	2
	Nominal	University	0.0029	2
	Nominal	Person	0.0044	3
	Nominal	Place	0.0015	1
	Nominal	City	0.0015	1
	Nominal	Comment	0.0015	1
AverageNeighbourDegree	Numeric	mean= 41143.160, std=55915.066	1.0000	686
EgoNetIncomingEdges	Numeric	mean= 153544.609, std=179332.031	1.0000	686

Table 14 *LDBC SNB Lables, Structural Features & Characteristic Set: ConceptNode I0, P(node) = 0.343, Count 686*

Attribute	ValueType	Value	Probability	Occurrences
RelationshipTypes	Nominal	HAS_MEMBER	0.7342	58
	Nominal	CONTAINER_OF	0.7342	58
	Nominal	HAS_MODERATOR	0.8481	67
	Nominal	HAS_TAG	0.9114	72
	Nominal	HAS_TYPE	0.1139	9
	Nominal	HAS_INTEREST	0.1139	9
	Nominal	IS_LOCATED_IN	0.0759	6
	Nominal	KNOWS	0.0253	2
	Nominal	WORK_AT	0.0253	2
	Nominal	HAS_CREATOR	0.0253	2
	Nominal	LIKES	0.0253	2
	Nominal	STUDY_AT	0.0253	2
	Nominal	IS_PART_OF	0.0127	1
EgoDegree	Numeric	mean= 101.759, std=361.730	1.0000	79
EgoNetOutgoingEdges	Numeric	mean= 13588.544, std=20538.965	1.0000	79
Labels	Nominal	Forum	0.8101	64
	Nominal	Tag	0.1139	9
	Nominal	Organisation	0.0253	2
	Nominal	University	0.0253	2
	Nominal	Person	0.0380	3
	Nominal	Place	0.0127	1
	Nominal	City	0.0127	1
AverageNeighbourDegree	Numeric	mean= 1512.177, std=1285.417	1.0000	79
EgoNetIncomingEdges	Numeric	mean= 40872.121, std=56835.594	1.0000	79

Table 15 *LDBC SNB Lables, Structural Features & Characteristic Set: ConceptNode l00, P(node) = 0.1151603498542274, Count 79*

Attribute	ValueType	Value	Probability	Occurrences
RelationshipTypes	Nominal	IS_LOCATED_IN	1.0000	471
	Nominal	CONTAINER_OF	1.0000	471
	Nominal	HAS_CREATOR	1.0000	471
	Nominal	LIKES	0.0382	18
	Nominal	HAS_TAG	0.0021	1
EgoDegree	Numeric	mean= 3.040, std=0.197	1.0000	471
EgoNetOutgoingEdges	Numeric	mean= 352.968, std=434.770	1.0000	471
Labels	Nominal	Message	1.0000	471
	Nominal	Post	1.0000	471
AverageNeighbourDegree	Numeric	mean= 55171.863, std=61742.605	1.0000	471
EgoNetIncomingEdges	Numeric	mean= 167092.719, std=186981.578	1.0000	471

Table 16 *LDBC SNB Lables, Structural Features & Characteristic Set: ConceptNode l01, P(node) = 0.6865889212827988, Count 471*

Attribute	ValueType	Value	Probability	Occurrences
RelationshipTypes	Nominal	IS_LOCATED_IN	1.0000	136
	Nominal	CONTAINER_OF	0.9926	135
	Nominal	HAS_CREATOR	1.0000	136
	Nominal	LIKES	0.1765	24
	Nominal	REPLY_OF	0.8529	116
	Nominal	HAS_TAG	0.8971	122
EgoDegree	Numeric	mean= 15.059, std=20.142	1.0000	136
EgoNetOutgoingEdges	Numeric	mean= 3113.500, std=9365.829	1.0000	136
Labels	Nominal	Message	1.0000	136
	Nominal	Post	0.9926	135
	Nominal	Comment	0.0074	1
AverageNeighbourDegree	Numeric	mean= 15579.483, std=17879.170	1.0000	136
EgoNetIncomingEdges	Numeric	mean= 172073.391, std=175803.813	1.0000	136

Table 17 *LDBC SNB Lables, Structural Features & Characteristic Set: ConceptNode l02, P(node) = 0.19825072886297376, Count 136*

Attribute	ValueType	Value	Probability	Occurrences
RelationshipTypes	Nominal	IS_LOCATED_IN	1.0000	1314
	Nominal	REPLY_OF	1.0000	1314
	Nominal	CONTAINER_OF	0.0008	1
	Nominal	HAS_CREATOR	1.0000	1314
	Nominal	HAS_TAG	0.3501	460
	Nominal	LIKES	0.0320	42
EgoDegree	Numeric	mean= 5.223, std=5.472	1.0000	1314
EgoNetOutgoingEdges	Numeric	mean= 1058.639, std=2950.754	1.0000	1314
Labels	Nominal	Message	1.0000	1314
	Nominal	Post	0.0008	1
	Nominal	Comment	0.9992	1313
AverageNeighbourDegree	Numeric	mean= 42490.211, std=53672.684	1.0000	1314
EgoNetIncomingEdges	Numeric	mean= 161375.422, std=180840.781	1.0000	1314

Table 18 *LDBC SNB Lables, Structural Features & Characteristic Set: ConceptNode l1, P(node) = 0.657, Count 1314*

Attribute	ValueType	Value	Probability	Occurrences
RelationshipTypes	Nominal	REPLY_OF	1.0000	459
	Nominal	IS_LOCATED_IN	1.0000	459
	Nominal	HAS_CREATOR	1.0000	459
	Nominal	HAS_TAG	0.9956	457
	Nominal	LIKES	0.0915	42
EgoDegree	Numeric	mean= 9.351, std=7.713	1.0000	459
EgoNetOutgoingEdges	Numeric	mean= 1563.516, std=4805.696	1.0000	459
Labels	Nominal	Message	1.0000	459
	Nominal	Comment	1.0000	459
AverageNeighbourDegree	Numeric	mean= 20528.490, std=24791.783	1.0000	459
EgoNetIncomingEdges	Numeric	mean= 159520.031, std=177424.219	1.0000	459

Table 19 *LDBC SNB Lables, Structural Features & Characteristic Set: ConceptNode l10, P(node) = 0.3493150684931507, Count 459*

Attribute	ValueType	Value	Probability	Occurrences
RelationshipTypes	Nominal	REPLY_OF	1.0000	855
	Nominal	IS_LOCATED_IN	1.0000	855
	Nominal	HAS_CREATOR	1.0000	855
	Nominal	HAS_TAG	0.0035	3
	Nominal	CONTAINER_OF	0.0012	1
EgoDegree	Numeric	mean= 3.007, std=0.128	1.0000	855
EgoNetOutgoingEdges	Numeric	mean= 787.601, std=879.029	1.0000	855
Labels	Nominal	Message	1.0000	855
	Nominal	Comment	0.9988	854
	Nominal	Post	0.0012	1
AverageNeighbourDegree	Numeric	mean= 54280.191, std=60822.508	1.0000	855
EgoNetIncomingEdges	Numeric	mean= 162371.594, std=182640.844	1.0000	855

Table 20 *LDBC SNB Lables, Structural Features & Characteristic Set: ConceptNode l11, P(node) = 0.6506849315068494, Count 855*

Bibliography

[1] *11.8. Statistics and execution plans - Chapter 11. Performance*. Jan. 3, 2020. URL: https://neo4j.com/docs/operations-manual/current/performance/statistics-execution-plans/ (visited on 01/11/2020).

[2] *6.4. Index Values and Order - Chapter 6. Query tuning*. Jan. 3, 2020. URL: https://neo4j.com/docs/cypher-manual/3.5/query-tuning/cypher-index-values-order/ (visited on 01/11/2020).

[3] *6.5. Planner hints and the USING keyword - Chapter 6. Query tuning*. Jan. 3, 2020. URL: https://neo4j.com/docs/cypher-manual/3.5/query-tuning/using/ (visited on 01/11/2020).

[4] *9th DIMACS Implementation Challenge: Shortest Paths*. June 14, 2010. URL: https://users.diag.uniroma1.it/challenge9/download.shtml (visited on 01/12/2020).

[5] Renzo Angles. "The Property Graph Database Model." In: *AMW*. 2018.

[6] Mihael Ankerst et al. "OPTICS: Ordering Points To Identify the Clustering Structure". In: ACM Press, 1999, pp. 49–60.

[7] Geoffrey H Ball and David J Hall. "A clustering technique for summarizing multivariate data". In: *Behavioral science* 12.2 (1967), pp. 153–155.

[8] Michael Baron. *Probability and Statistics for Computer Scientists, Second Edition*. 2nd. Chapman & Hall/CRC, 2013.

[9] E. T. Bell. "Exponential Numbers". In: *The American Mathematical Monthly* 41.7 (1934), pp. 411–419. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2300300.

[10] Pavel Berkhin. "A survey of clustering data mining techniques". In: *Grouping multidimensional data*. Springer, 2006, pp. 25–71.

[11] Tim Bock. *What is a Dendrogram? How to use Dendrograms | Displayr*. 2018. URL: https://www.displayr.com/what-is-dendrogram/ (visited on 11/26/2019).

[12] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. "Density-based clustering based on hierarchical density estimates". In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer. 2013, pp. 160–172.

[13] Gunnar Carlsson. "Topology and data". In: *Bulletin of the American Mathematical Society* 46.2 (2009), pp. 255–308.

[14] Tony F. Chan, Gene H. Golub, and Randall J. LeVeque. "Algorithms for Computing the Sample Variance: Analysis and Recommendations". In: *The American Statistician* 37.3 (1983), pp. 242–247. ISSN: 00031305. URL: http://www.jstor.org/stable/2683386.

[15] Kamalika Chaudhuri and Sanjoy Dasgupta. "Rates of convergence for the cluster tree". In: *NIPS*. 2010.

[16] Peter Cheeseman et al. "Autoclass: A Bayesian classification system". In: *Machine learning proceedings 1988*. Elsevier, 1988, pp. 54–64.

[17] James E Corter and Mark A Gluck. "Explaining basic categories: Feature predictability and information." In: *Psychological Bulletin* 111.2 (1992), p. 291.

[18] Mark Devaney and Ashwin Ram. "Efficient feature selection in conceptual clustering". In: *ICML*. Vol. 97. 1997, pp. 92–97.

[19] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: AAAI Press, 1996, pp. 226–231.

[20] Ludwig Fahrmeir et al. *Statistik: Der weg zur datenanalyse*. Springer-Verlag, 2016.

[21] Doug Fisher. "Iterative optimization and simplification of hierarchical clusterings". In: *Journal of artificial intelligence research* 4 (1996), pp. 147–178.

[22] Douglas H. Fisher. "Knowledge acquisition via incremental conceptual clustering". In: *Machine Learning* 2.2 (Sept. 1987), pp. 139–172. ISSN: 1573-0565. DOI: 10.1007/BF00114265. URL: https://doi.org/10.1007/BF00114265.

[23] Douglas Fisher and Pat Langley. *Approaches to Conceptual Clustering.* Tech. rep. CALIFORNIA UNIV IRVINE DEPT OF INFORMATION and COMPUTER SCIENCE, 1985.

[24] John H Gennari, Pat Langley, and Doug Fisher. "Models of incremental concept formation". In: *Artificial intelligence* 40.1-3 (1989), pp. 11–61.

[25] M Gluck. "Information, uncertainty and the utility of categories". In: *Proc. of the Seventh Annual Conf. on Cognitive Science Society, 1985*. 1985.

[26] Robert Godin, Rokia Missaoui, and Hassan Alaoui. "INCREMENTAL CONCEPT FORMATION ALGORITHMS BASED ON GALOIS (CONCEPT) LATTICES". In: *Computational Intelligence* 11.2 (1995), pp. 246–267. DOI: 10.1111/j.1467-8640.1995.tb00031.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8640.1995.tb00031.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8640.1995.tb00031.x.

[27] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[28] Keith Henderson et al. "It's who you know: graph mining using recursive structural features". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, pp. 663–671.

[29] Keith Henderson et al. "Rolx: structural role extraction & mining in large graphs". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 1231–1239.

[30] Matthew D Hoffman et al. "Stochastic variational inference". In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 1303–1347.

[31] Dmitry I Ignatov, Sergei O Kuznetsov, and Jonas Poelmans. "Concept-based biclustering for internet advertisement". In: *2012 IEEE 12th International Conference on Data Mining Workshops*. IEEE. 2012, pp. 123–130.

[32] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. "Data clustering: a review". In: *ACM computing surveys (CSUR)* 31.3 (1999), pp. 264–323.

[33] Xin Jin and Jiawei Han. "K-Means Clustering". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 563–564. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_425. URL: https://doi.org/10.1007/978-0-387-30164-8_425.

[34] Kirthevasan Kandasamy et al. "Neural architecture search with bayesian optimisation and optimal transport". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2016–2025.

[35] Teuvo Kohonen. “Exploration of very large databases by self-organizing maps”. In: *Proceedings of International Conference on Neural Networks (ICNN'97)*. Vol. 1. IEEE. 1997, PL1–PL6.

[36] Teuvo Kohonen. “The self-organizing map”. In: *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480.

[37] Sven Kosub. “A note on the triangle inequality for the Jaccard distance”. In: *CoRR* abs/1612.02696 (2016). arXiv: 1612.02696. URL: http://arxiv.org/abs/1612.02696.

[38] Helmut Krcmar. “Informationsmanagement”. In: *Informationsmanagement*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 85–111. ISBN: 978-3-662-45863-1. DOI: 10.1007/978-3-662-45863-1_4. URL: https://doi.org/10.1007/978-3-662-45863-1_4.

[39] Pat Langley et al. *An integrated cognitive architecture for autonomous agents*. Tech. rep. CALIFORNIA UNIV IRVINE SCHOOL OF INFORMATION and COMPUTER SCIENCE, 1990.

[40] Gang Leng, Thomas Martin McGinnity, and Girijesh Prasad. “Design for self-organizing fuzzy neural networks based on genetic algorithms”. In: *IEEE Transactions on Fuzzy Systems* 14.6 (2006), pp. 755–766.

[41] Robert Levinson. “A self-organizing retrieval system for graphs.” In: *AAAI*. 1984, pp. 203–206.

[42] Stuart Lloyd. “Least squares quantization in PCM”. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[43] Christopher J MacLellan et al. “Trestle: A model of concept formation in structured domains”. In: *Advances in Cognitive Systems* 4 (2016), pp. 131–150.

[44] James MacQueen et al. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[45] Leland McInnes and John Healy. “Accelerated hierarchical density based clustering”. In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE. 2017, pp. 33–42.

[46] Kathleen McKusick and Kevin Thompson. “Cobweb/3: A portable implementation”. In: (1990).

[47] Ryszard Michalski. “Knowledge acquisition through conceptual clustering: A theoretical framework and algorithm for partitioning data into conjunctive concepts”. In: *International Journal of Policy Analysis and Information Systems* 4 (1980), pp. 219–243.

[48] Ryszard S Michalski and Robert E Stepp. “Learning from observation: Conceptual clustering”. In: *Machine learning*. Springer, 1983, pp. 331–363.

[49] Boris Mirkin. *Mathematical classification and clustering*. Vol. 11. Springer Science & Business Media, 2013.

[50] Tom M Mitchell. “Generalization as search”. In: *Artificial intelligence* 18.2 (1982), pp. 203–226.

[51] Tom Michael Mitchell. *The discipline of machine learning*. Vol. 9. Carnegie Mellon University, School of Computer Science, Machine Learning . . ., 2006.

[52] Fionn Murtagh. “Ultrametric and generalized ultrametric in computational logic and in data analysis”. In: *arXiv preprint arXiv:1008.3585* (2010).

[53] Fionn Murtagh and Pedro Contreras. “Hierarchical Clustering for Finding Symmetries and Other Patterns in Massive, High Dimensional Datasets”. In: *Data Mining: Foundations and Intelligent Paradigms* (2012), pp. 95–130. ISSN: 1868-4408. DOI: 10.1007/978-3-642-23166-7_5. URL: http://dx.doi.org/10.1007/978-3-642-23166-7_5.

[54] Thomas Neumann and Guido Moerkotte. "Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins". In: *2011 IEEE 27th International Conference on Data Engineering*. IEEE. 2011, pp. 984–994.

[55] Allen E. Nix and Michael D. Vose. "Modeling genetic algorithms with Markov chains". In: *Annals of Mathematics and Artificial Intelligence* 5.1 (Mar. 1992), pp. 79–88. ISSN: 1573-7470. DOI: 10.1007/BF01530781. URL: https://doi.org/10.1007/BF01530781.

[56] Mateusz Pawlik and Nikolaus Augsten. "Tree edit distance: Robust and memory-efficient". In: *Information Systems* 56 (2016), pp. 157–173.

[57] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[58] Beatriz Pontes, Raúl Giráldez, and Jesús S. Aguilar-Ruiz. "Biclustering on expression data: A review". In: *Journal of Biomedical Informatics* 57 (2015), pp. 163–180. ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2015.06.028. URL: http://www.sciencedirect.com/science/article/pii/S1532046415001380.

[59] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases*. " O'Reilly Media, Inc.", 2013.

[60] Eleanor Rosch et al. "Basic objects in natural categories". In: *Cognitive Psychology* 8 (1976), pp. 382–439.

[61] SIGKDD. *SIGKDD*. Dec. 11, 2019. URL: https://www.kdd.org/ (visited on 12/11/2019).

[62] Herbert A Simon. *The sciences of the artificial*. 1968.

[63] P. H. A. Sneath. "The Application of Computers to Taxonomy". In: *Microbiology* 17.1 (1957), pp. 201–226. ISSN: 1350-0872. DOI: https://doi.org/10.1099/00221287-17-1-201. URL: https://www.microbiologyresearch.org/content/journal/micro/10.1099/00221287-17-1-201.

[64] Kuo-Chung Tai. "The Tree-to-Tree Correction Problem". In: *J. ACM* 26.3 (July 1979), pp. 422–433. ISSN: 0004-5411. DOI: 10.1145/322139.322143. URL: http://doi.acm.org/10.1145/322139.322143.

[65] Sergios Theodoridis and Konstantinos Koutroumbas. "Chapter 12 - Clustering Algorithms I: Sequential Algorithms". In: *Pattern Recognition (Fourth Edition)*. Ed. by Sergios Theodoridis and Konstantinos Koutroumbas. Fourth Edition. Boston: Academic Press, 2009, pp. 627–652. ISBN: 978-1-59749-272-0. DOI: https://doi.org/10.1016/B978-1-59749-272-0.50014-1. URL: http://www.sciencedirect.com/science/article/pii/B9781597492720500141.

[66] Sergios Theodoridis and Konstantinos Koutroumbas. "Chapter 14 - Clustering Algorithms III: Schemes Based on Function Optimization". In: *Pattern Recognition (Fourth Edition)*. Ed. by Sergios Theodoridis and Konstantinos Koutroumbas. Fourth Edition. Boston: Academic Press, 2009, pp. 701–763. ISBN: 978-1-59749-272-0. DOI: https://doi.org/10.1016/B978-1-59749-272-0.50016-5. URL: http://www.sciencedirect.com/science/article/pii/B9781597492720500165.

[67] Kevin Thompson and Pat Langley. "Concept formation in structured domains". In: *Concept Formation*. Elsevier, 1991, pp. 127–161.

[68] Kevin Thompson and Pat Langley. "Incremental concept formation with composite objects". In: *Proceedings of the sixth international workshop on Machine learning*. Elsevier. 1989, pp. 371–374.

[69] Charles D. Tupper. "21 - Object and Object/Relational Databases". In: *Data Architecture*. Ed. by Charles D. Tupper. Boston: Morgan Kaufmann, 2011, pp. 369–383. ISBN: 978-0-12-385126-0. DOI: https://doi.org/10.1016/B978-0-12-385126-0.00021-8. URL: http://www.sciencedirect.com/science/article/pii/B9780123851260000218.

[70] Joe H Ward Jr. “Hierarchical grouping to optimize an objective function”. In: *Journal of the American statistical association* 58.301 (1963), pp. 236–244.

[71] Alfred North Whitehead and Bertrand Russell. *Principia mathematica*. Vol. 2. University Press, 1912.

[72] David Wishart. “256. Note: An algorithm for hierarchical classifications”. In: *Biometrics* (1969), pp. 165–170.

[73] Zhipeng Xie et al. “Concept lattice based composite classifiers for high predictability”. In: *Journal of Experimental & Theoretical Artificial Intelligence* 14.2-3 (2002), pp. 143–156.

[74] Rui Xu and Donald C Wunsch. “Survey of clustering algorithms”. In: (2005).

[75] Jason Yosinski et al. “Understanding Neural Networks Through Deep Visualization”. In: *Deep Learning Workshop, International Conference on Machine Learning (ICML)*. 2015.