Universität Konstanz

Label Hierarchy Inference in Property Graph Databases

Fabian Klopfer

Databases and Information Systems Group

University of Konstanz, 10.03.2020

Outline

Motivation

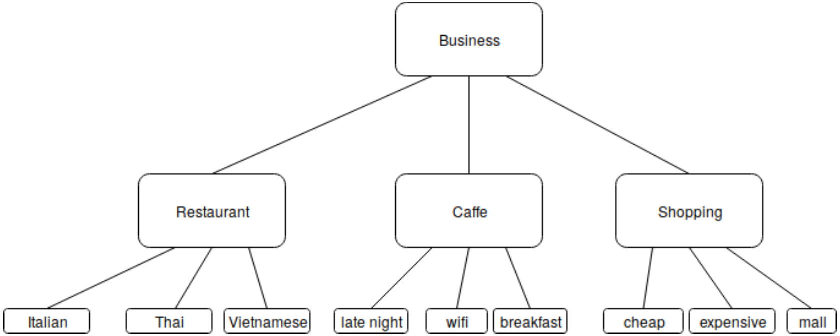
“Thus my central theme is that complexity frequently takes the form of hierarchy and that hierarchic systems have some common properties, independent of their specific content. Hierarchy, I shall argue, is one of the central structural schemes, that the architect of complexity uses.“

- Herbert A. Simon,
Nobel Laureate and ACM Turing award winner,
The Sciences of the Artificial, 1968 [1].

- Implicit hierarchical structure in many data sets
- Implicit in database models
- No explicit representation in the property graph model or Neo4J

Problem Statement

Given: Data set represented in the property graph model
Wanted: Hierarchy of labels reflecting the implicit hierarchical structure



Node.name	Node.tags
Fernando's	restaurant, italian
Arche	restaurant, vietnamese
Bangkok	restaurant, thai
CampusCafe	cafe, wifi
Endlicht	cafe, latenight
Pano	cafe, breakfast
Lago	shopping, mall
Seerhein Center	shopping, cheap
Seepark	Shopping, expensive

Overview

- Pre-Processing: Feature Generation & String Vectorization
- Post-Processing: Dendrogram flattening
- Steps vary between algorithms
- For some additional parameter inference necessary: Randomized search used here

Survey

Goal: Qualitative and quantitative evaluation of different approaches.

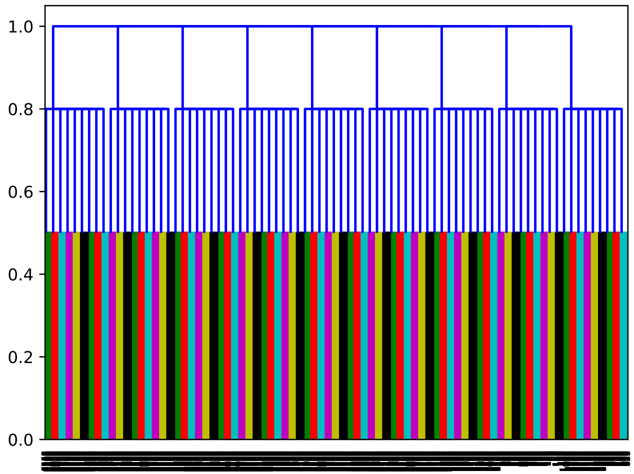
- Quality: Deviation from synthetic ground truth
- Quantity: Memory and run time complexity, benchmarked run time of state of the art implementations

Approaches considered in this thesis:

1. Agglomerative Clustering [2]
2. Two-Step: Non-Hierarchical and Agglomerative Hierarchical Clustering [3]
3. Conceptual Clustering [4]

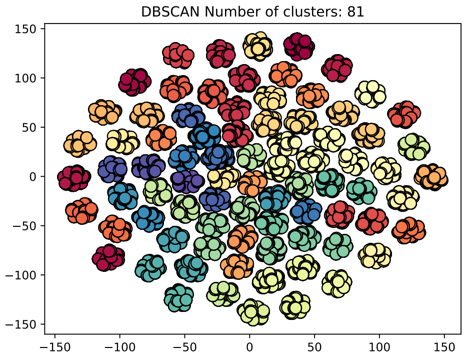
Hierarchical Agglomerative Clustering

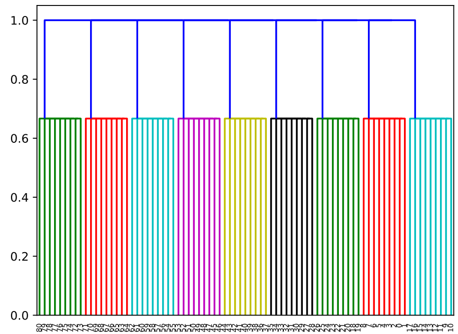
- Basic Idea:
  1. compute the pairwise distance matrix
  2. Merge the two clusters with the smallest cluster distance.
- Computational Complexity: $\mathcal{O}(n^3)$
- Space Complexity: $\mathcal{O}(n^2)$
- Extension with a computational complexity of $\mathcal{O}(n^2)$: Robust Single Linkage [5]



Two-step clustering

- Basic Idea: Apply a “flat“ clustering algorithm and combine it with subsequent hierarchical clustering.
- Partition-based: k-Means, TTSAS [6, 7]
- Density-based: DBSCAN, OPTICS, HDBSCAN [3, 8, 9]
- requires to infer parameters of the flat clustering algorithms





Conceptual clustering

- Basic Idea: Build a hierarchy of label sets/concepts with descriptions by integrating instances iteratively
- When integrating choose one of five operations at each level, optimizing value-predictiveness [10–13]

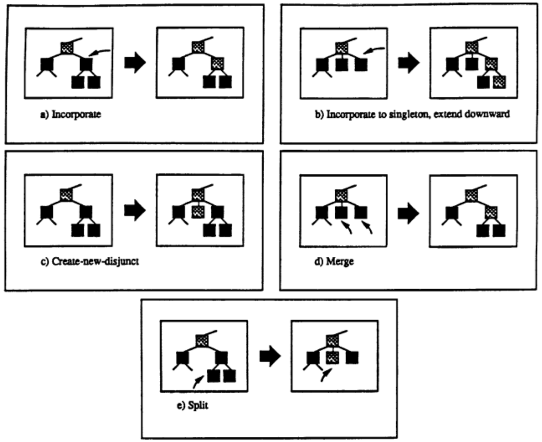


Figure 4. COBWEB's learning operators.

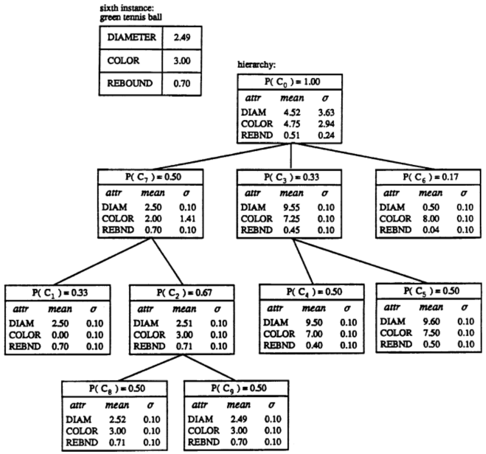


Figure 12. COBWEB hierarchy after the sixth instance of a ball.

Summary Algorithms

Algorithm	Runtime Complexity	Space Complexity
Single Linkage	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
Robust Single Linkage	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
k-Means	$\mathcal{O}(n \cdot k \cdot d \cdot i)$	$\mathcal{O}(n)$
TTSAS	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
DBSCAN	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n^2)$
OPTICS	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n^2)$
HDBSCAN	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
COBWEB	$\mathcal{O}(n \cdot \log(n) \cdot b^2 AV)$	$\mathcal{O}(n)$

Experimental Evaluation Setup I

- A synthetic data set is used
- noise $\equiv$ take a node and remove $\vee$ rename a label.

```
"id":24,"labels":"l2, l22",
"id":25,"labels":"l22",
"id":26,"labels":"l1"
```

- Run the pipeline on each of the noise variations and algorithm:
  - no noise
  - 5% noise
  - 10% noise
  - 20% noise
  - 33% noise

Experimental Evaluation Setup II

- . . . and for each of the sample sizes

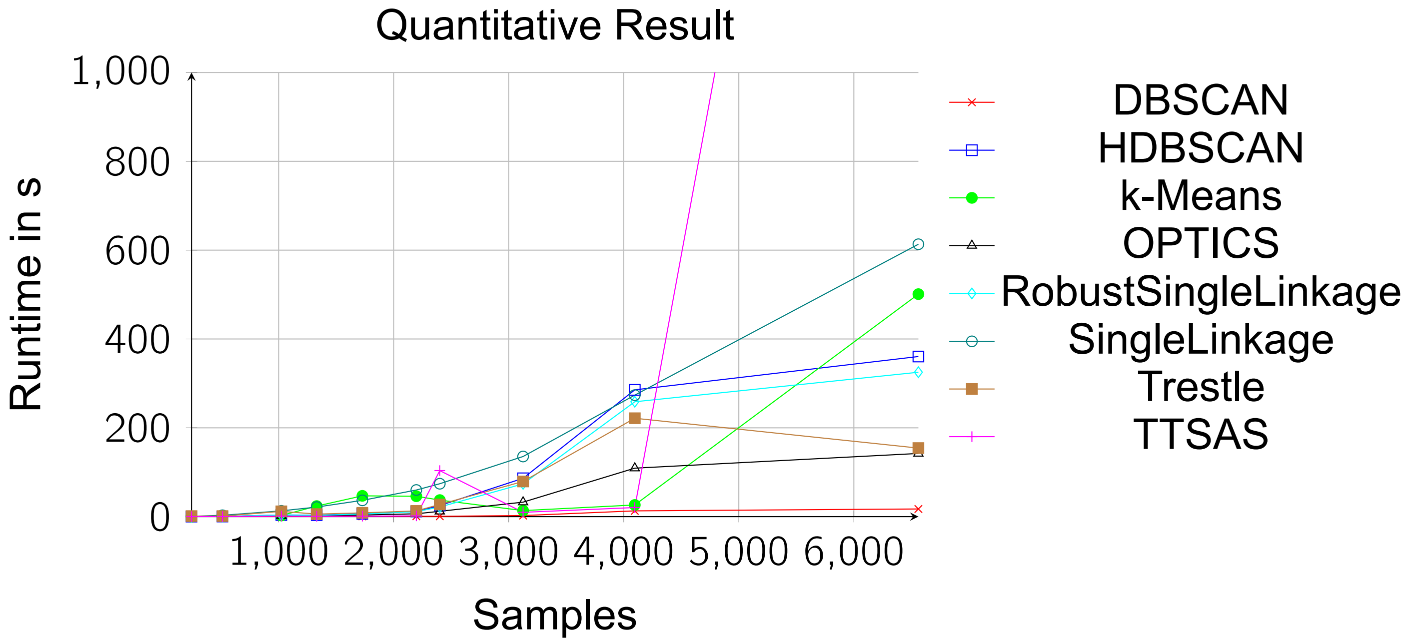
Size	Width	Depth
243	3	5
512	8	3
1024	4	5
1331	11	3
1728	12	3
2197	13	3
2401	7	4
3125	5	5
4096	4	6
6561	9	4

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

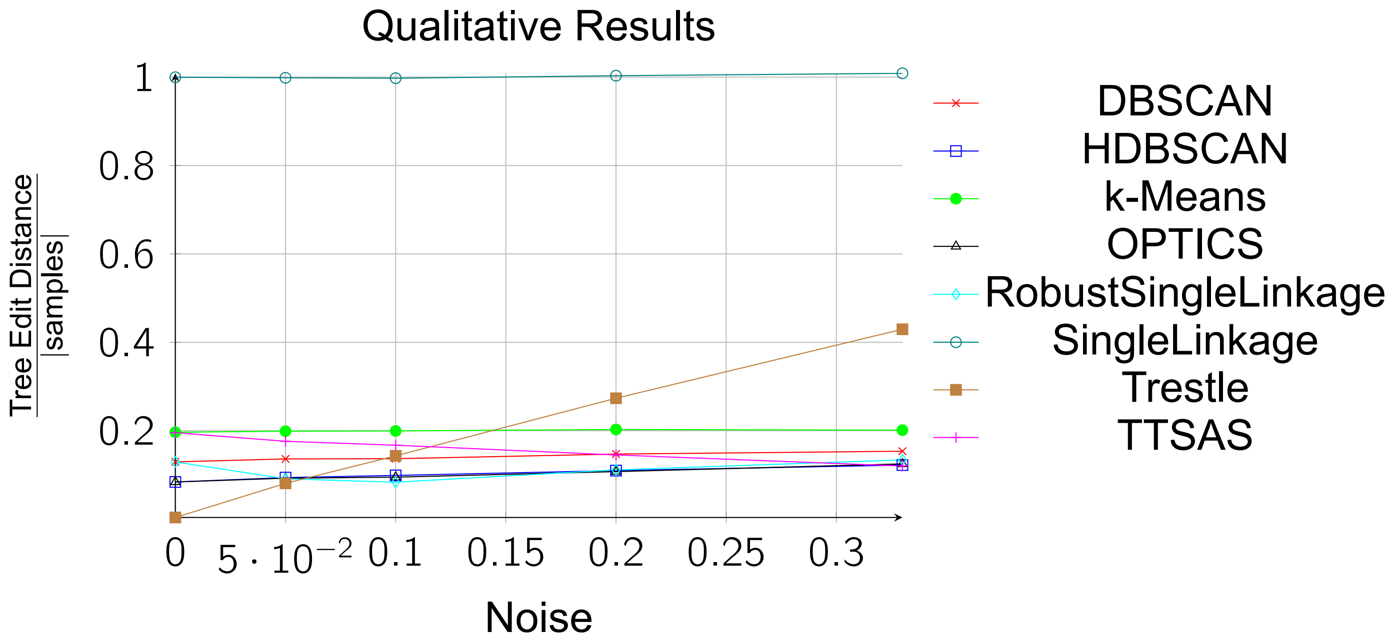
Experimental Evaluation Setup III

- Metric for how much resulting hierarchy deviates from perfect: Tree Edit Distance [14] [15]
- Implementation of the algorithms: SciPy [16], scikit-learn [17], PyClustering [18], concept-formation [19], hdbscan [3].
- For all algorithms having parameters: Use Random Search-based parameter optimization (32 Samples)

Survey Results I



Survey Results II



Discussion

- Performance of hierarchical agglomerative clustering can be improved by pre-clustering
- Parameters vary a lot between data sets and inference is very expensive
- TTSAS and k-Means run time highly dependent on parameters and initialization
- Density-based methods robust to noise and decent in run time but require additional processing
- squared space complexity is infeasible for larger data sets with many features, labels and properties
- but can be overcome using database-oriented implementations
- empty intersection in noisy domains
- Conceptual Clustering can be improved in quality [20], space and computational complexity

Feature Generation and Selection

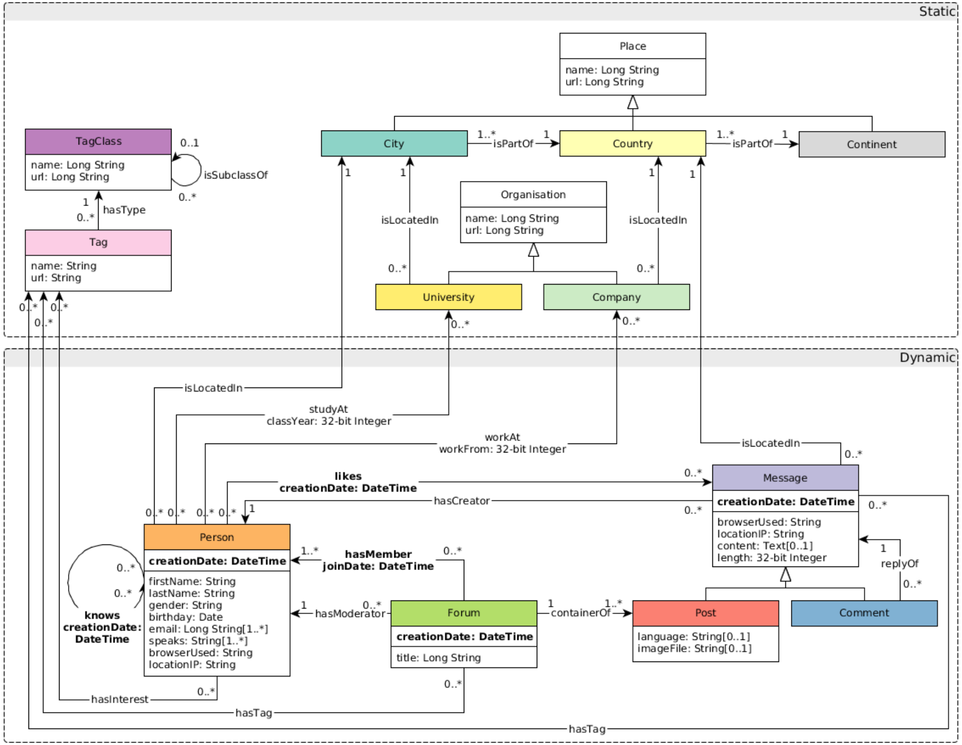
Given: Data set represented in property graph model $G = (V, E, \lambda, P, T, L, f_P, f_T, f_L)$
Wanted: Feature vector capturing graph-based information
Possible Features:

- labels V, L, f_l
- properties V, E, P, f_P,
- per node structural features: [21, 22]
  - node degree
  - average neighbour degree
  - number of edges incoming to the ego net
  - number of edges outgoing from the ego net
- characteristic set: all connected relationship types E, T, f_T per node [23]

Experimental Evaluation Setup

- COBWEB is used as algorithm to experiment with as it
  - is parameter-free.
  - requires linear space.
  - uses probabilistic concepts.
  - is hierarchical avoiding additional steps.
  - is able to deal with mixed data.
- The LDBC social network benchmark and New York road network data sets are used
- Neo4j was used as property graph database, Cobweb was implemented as front-end procedure

LDBC SNB Schema



Results I

- Feature Vector of the survey
- Hierarchy splits always nodes with the most frequently occurring label

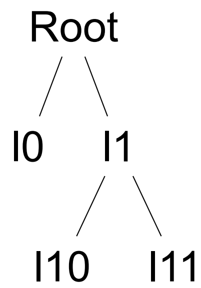


Figure *LDBC SNB: labels only*

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

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

Results II

- Adding properties, adds further refinements (e.g. splits on length for messages)
- Depending on the format and number of properties

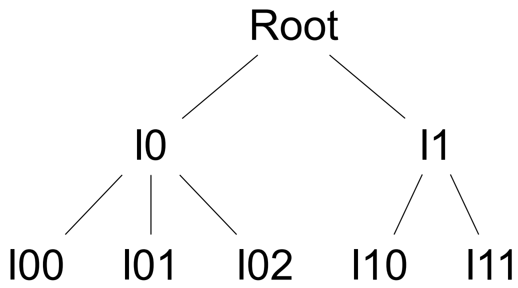


Figure *LDBC SNB: labels and properties*

Results III

- Using labels, structural features and the characteristic set supports road net-like data sets
- Especially useful if labels, relationship types and properties are uniform, non-existing or noise
- Hierarchy induces distinct connectivity profiles within neighbourhood for road net

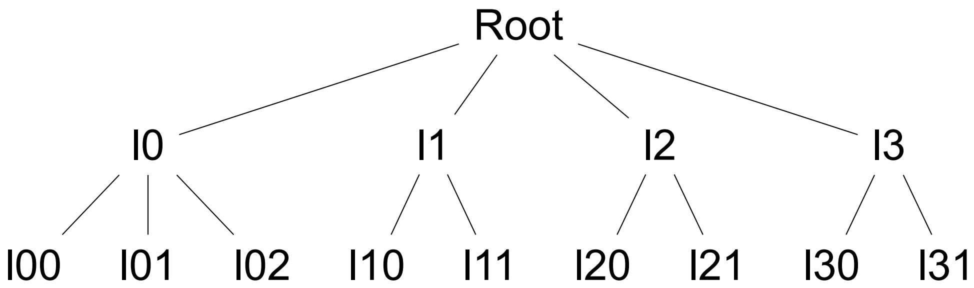


Figure *New York road net: labels, structural features and characteristic set*

Results IV

- Separates by label message at first split
- On the second, divides messages into those with low degree and high degree with characteristic sets being IS_COMMENT_OF, HAS_TAG on the high degree side with
- on the other branch of the tree posts are being split from the rest of the nodes

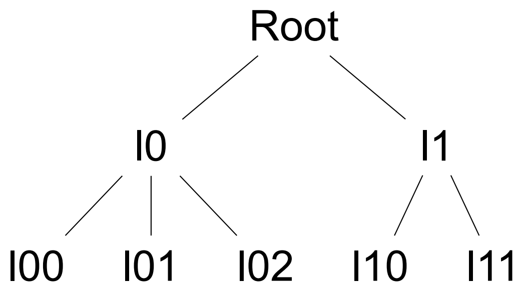


Figure *LDBC SNB: labels, structural features and characteristic set*

Results V

- Using all available features introduces noise, depending on the amount of overall information in the features
- here the second level split now clusters messages that were created using the same browser

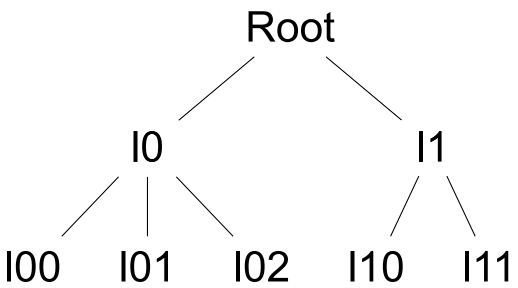


Figure *LDBC SNB: labels, properties, structural features and characteristic set*

Discussion: Feature Vector

Property graph model allows many ways of representing data:

- *OO-like* Many node labels, many node properties, no relationships at all
  $\Rightarrow$ Labels and properties
- *RDF-like* Many labels, many relationship types, no properties
  $\Rightarrow$ Structural features, labels and characteristic set
- *More complex* Many labels, many rel. types, many properties for both nodes and relationships, varying structural features
  $\Rightarrow$ All features carry information, which are the predictive ones?

Too much information introduces noise, not enough information yields trivial hierarchies

$\Rightarrow$ Use adaptive feature vector, depending on the information available in the database profile

Summary

- Classical hierarchical agglomerative clustering does not scale well enough to derive meaningful hierarchies directly
- two-phase approaches improve the performance, but require parameters to be optimized
- Conceptual clustering provides parameter-free hierarchical clustering, possibly online and with potential optimizations to scale
- Feature Vectors need to be constructed adaptively when possible
- too large ones slow computation down and introduce noise
- too small ones do not provide enough information for all data representations in the property graph model

Cardinality Estimation I

- many databases use independence, and uniformity assumption, database profile, histograms, sampling techniques
- Example Neo4j:
  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.
  5. One additional index at a time.
- Taxonomies capture differences in value distributions dependent on the description of the concept

Cardinality Estimation II

- Storing taxonomies explicitly as index in the database profile provides more information on the selectivity and conditional covariances between attributes
- instead of the whole taxonomy, labels can be used along with the concept description
- Query optimizer needs to be adapted to query the index appropriately
- Further refinements to COBWEB may help to speed up
- Algorithm is incremental in nature (update-able)

References I

Simon, H. A. *The sciences of the artificial.* (1968).

Ward, J. J. H. *Hierarchical Grouping to Optimize an Objective Function.* in (1963).

McInnes, L., Healy, J. & Astels, S. hdbscan: Hierarchical density based clustering. *J. Open Source Software* **2,** 205 (2017).

Michalski, R. Knowledge acquisition through conceptual clustering: A theoretical framework and algorithm for partitioning data into conjunctive concepts. *International Journal of Policy Analysis and Information Systems* **4,** 219–243 (1980).

Chaudhuri, K. & Dasgupta, S. *Rates of convergence for the cluster tree.* in *NIPS* (2010).

MacQueen, J. *et al. Some methods for classification and analysis of multivariate observations.* in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* **1** (1967), 281–297.

Theodoridis, S. & Koutroumbas, K. in *Pattern Recognition (Fourth Edition)* (eds Theodoridis, S. & Koutroumbas, K.) Fourth Edition, 627–652 (Academic Press, Boston, 2009). ISBN: 978-1-59749-272-0. `http://www.sciencedirect.com/science/article/pii/B9781597492720500141`.

Ester, M., Kriegel, H.-P., Sander, J. & Xu, X. *A density-based algorithm for discovering clusters in large spatial databases with noise.* in (AAAI Press, 1996), 226–231.

Ankerst, M., Breunig, M. M., Kriegel, H.-p. & Sander, J. *OPTICS: Ordering Points To Identify the Clustering Structure.* in (ACM Press, 1999), 49–60.

Gennari, J. H., Langley, P. & Fisher, D. Models of incremental concept formation. *Artificial intelligence* **40,** 11–61 (1989).

McKusick, K. & Thompson, K. Cobweb/3: A portable implementation. (1990).

Corter, J. E. & Gluck, M. A. Explaining basic categories: Feature predictability and information. *Psychological Bulletin* **111,** 291 (1992).

Gluck, M. *Information, uncertainty and the utility of categories.* in *Proc. of the Seventh Annual Conf. on Cognitive Science Society, 1985* (1985).

References II

Pawlik, M. & Augsten, N. RTED: a robust algorithm for the tree edit distance. *Proceedings of the VLDB Endowment* **5,** 334–345 (2011).

Pawlik, M. & Augsten, N. Tree edit distance: Robust and memory-efficient. *Information Systems* **56,** 157–173 (2016).

Jones, E., Oliphant, T., Peterson, P., *et al.* *SciPy: Open source scientific tools for Python.* [Online; accessed <today>]. 2001. `http://www.scipy.org/`.

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

Novikov, A. PyClustering: Data Mining Library. *Journal of Open Source Software* **4,** 1230. `https://doi.org/10.21105/joss.01230` (2019).

MacLellan, C., Harpstead, E., Aleven, V. & Koedinger, K. TRESTLE: A Model of Concept Formation in Structured Domains. *Advances in Cognitive Systems* **4** (2016).

Fisher, D. Iterative optimization and simplification of hierarchical clusterings. *Journal of artificial intelligence research* **4,** 147–178 (1996).

Henderson, K. *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* (2011), 663–671.

Henderson, K. *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* (2012), 1231–1239.

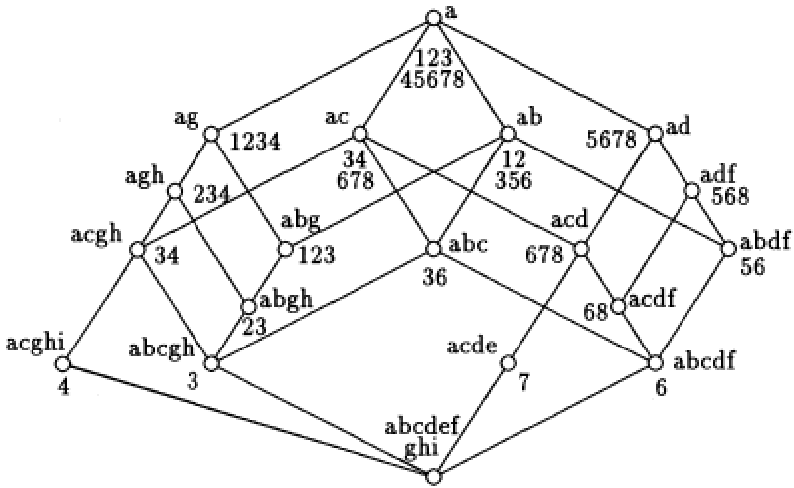
Neumann, T. & Moerkotte, G. *Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins.* in *2011 IEEE 27th International Conference on Data Engineering* (2011), 984–994.

Appendix


IF YOU DON'T
DO ANYTHING
YOU'LL NEVER
MAKE MISTAKES
memegenerator.es

Concept Lattices I

		a	b	c	d	e	f	g	h	i
1	Fischegel	×	×					×		
2	Brasse	×	×					×	×	
3	Frosch	×	×	×				×	×	
4	Hund	×		×				×	×	×
5	Wasserpest	×	×		×		×			
6	Schilf	×	×	×	×		×			
7	Bohne	×		×	×	×				
8	Mais	×		×	×		×			



Concept Lattices II

- Mathematical framework, developed in the 80s
- Able to deal with a wide range of data types (numeric, nominal, ordinal, different scales, . . .)
- Contains reductions of arbitrary (eventually uncountably infinite) complexity to binary
- Has been extended to Fuzzy Concept Lattices for fuzzy sets
- Problem: Construction of lattice may end in construction of the power set for independent uniform attribute distributions

Probabilistic Concepts in a Concept Lattice

Given: Attribute A and B are present

Desired: What is the joint distribution for all other non-queried attributes?

How does this help us?

1. When using all data
   - Complete analytic characterization pre-computed
   - One index for all data
   - maximal linear lookup times for all aggregation queries ($\log_b(2^n)$ assuming $b \geq 2$)
   - Index-backed property look-up (unclustered)
   - Exact cardinalities catching all conditional co-variances and conditional selectivities
2. When sampling from data
   - Cardinality estimates incorporating all conditional co-variances
   - Probably approximately correct learned analytic characterization

Neo4j: Physical Storage I

- Uses offsets as ids
- Linked list of fixed size records
- Strings and arrays stored separately and dynamically
- Nodes and relationships each reference their first property
- Nodes reference first relationship of their relationship chain
- Relationships reference start and end node, previous and next element in the relationship chain for both start and end node
- Indexes consume approximately 1/3 of the average property value size

Neo4j: Physical Storage II

There are three types of files:

1. nodestore.db (node related data; 15B/entry)
2. relationshipstore.db (relationship related data, 34B/Entry)
3. propertystore.db (property related data, varies)

Neo4j: Query Processing I

Steps taken when a query is processed by Neo4j

1. Parse into AST
2. Optimize and normalize AST into typed AST:
   For example move all from MATCH to WHERE
3. Generate query graph from AST
   Also: Retrieve statistics from profile, i.e. label selectivity, index
4. Create logical plan from query graph
5. Generate physical/execution plan

Neo4j: Query Processing II

The statistical information that Neo4j keeps is:

- The number of nodes having a certain label.
- The number of relationships by type.
- The number of relationships by type, ending or starting from a node with a specific label.
- Selectivity per index. (By sampling 5% after update)

Example I

Example Cypher Query

```
MATCH (p:Person)-[:LIKES]-(t:Technology)
WHERE p.profession='CS Student'
RETURN p
```

Transformed

```
MATCH (p)-[r]-(t)
WHERE p.label='Person', r.type='LIKES', t.label='Technology', p.profession='CS Student'
RETURN p
```

Example II

Now query type hierarchy for node with label Person having relationship of type LIKES and property named profession taking the value ’CS Student’. This can be done in $\log_b$(sample size).

- Should yield a lower cardinality than simply cardinality of label Persons and number relationships by type with start from specific label
- Also better than index over Person.profession as relationship type is also taken into account