Hierarchical Clustering for Property Graph Schema Discovery
Angela Bonifati, Stefania Dumbrava, Nicolas Mir

To cite this version:
Angela Bonifati, Stefania Dumbrava, Nicolas Mir. Hierarchical Clustering for Property Graph Schema Discovery. 25th International Conference on Extending Database Technology (EDBT), Mar 2022, Edinburgh, United Kingdom. hal-03692293

HAL Id: hal-03692293
https://hal.archives-ouvertes.fr/hal-03692293
Submitted on 9 Jun 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Hierarchical Clustering for Property Graph Schema Discovery

Angela Bonifati
Lyon 1 University & LIRIS CNRS
France
angela.bonifati@univ-lyon1.fr

Stefania Dumbrava
ENSIEE & Inst. Polytech. de Paris
France
stefania.dumbrava@ensiie.fr

Nicolas Mir
ENSIEE
France
nicolas.mir@ensiie.fr

ABSTRACT

The property graph model is becoming increasingly popular among users and is currently employed by several open-source and commercial graph database systems. Although property graphs are widely adopted, there is a lack of understanding of their underlying schema structure. In particular, the schema discovery problem consists of extracting the schema concepts from a property graph. A property graph schema helps provide a concise description of the data it represents, to make it more digestible for humans and interactive processes, as well as usable for query optimization purposes. In this paper, we address the property graph schema discovery problem and introduce the GMMSchema method based on hierarchical clustering using a Gaussian Mixture Model, which accounts for both label and property information on nodes. We experimentally analyze the accuracy and performance of GMMSchema, compared to those of its closest competitor, and showcase its superiority on several commonly used datasets, including real-world ones, such as the Covid19 knowledge graph, as well as the Fb25 and Mb6 NeuPrint graphs.

1 INTRODUCTION

Graphs are a natural abstraction for representing interconnected data. These have become increasingly pervasive in a wide variety of contexts, ranging from social networks to scientific repositories and the Semantic Web. To address the need to store, process, and analyze graph-shaped data, a novel type of NoSQL stores, namely graph databases, have been developed. Their most expressive underlying data model is the property graph one along with its variants [1, 3, 16], in which lists of properties are attached to both the nodes and the edges of a directed, labeled, multi-graph. Due to this rich formalism and to the fact that graph databases do not impose rigid schema constraints a priori, these can be readily employed to compactly capture complex graphs, integrating heterogeneous data sources. As graph datasets are large and rapidly evolving in practice, this flexibility is desirable for supporting scalable processing. However, the lack of a schema also has numerous drawbacks, as it makes data integration error-prone and hinders query optimization, meta-data management, and the extraction of type-based graph features needed in various machine learning applications.

To address this, we introduce a novel schema discovery method, leveraging a hierarchical clustering algorithm, based on Gaussian Mixture Models. To the best of our knowledge, ours is the first such approach to take into account labeling and property information simultaneously and, more generally, the first to perform schema discovery for property graphs using statistical methods. Schema inference for property graphs based on the MapReduce paradigm has been addressed in the past [14]. However, our method differs from previous work in that it is purely statistical and overcomes previous approaches in terms of both accuracy and performance.

The paper is organized as follows. Section 2 presents the related work in the area. Section 3 introduces preliminary concepts. Section 4 gives an overview of our method and illustrates the underlying GMMSchema algorithm. Section 5 evaluates the quality of the discovered schema and the overall performance of our approach, compared to those of its closest competitor.

2 RELATED WORK

Several works [10, 12] have investigated schema discovery for semi-structured data, to improve querying and data quality. In the context of RDF datasets, popular methods use clustering techniques to group similar entities into clusters corresponding to distinct types. As such, similarity measures are defined to capture the extent to which entities share the same properties. In [5], density-based clustering (DBSCAN) is applied to extracted structural patterns, instead of the original dataset, as in [13]. However, as DBSCAN is parametric, the method exhibits relatively large variance in accuracy. This is improved upon by the parameter-free StaTIX method in [15], based on the Louvain community-detection algorithm. While these approaches are hierarchical and soft (i.e., support overlapping types), similarly to GMMSchema, our approach is tailored to the more complex model of property graphs and ultimately produces a schema graph. In the context of JSON datasets, a MapReduce method is introduced by [2]. This combines individual type inference (Map operation) and type reduction (Reduce operation) to merge types based on equivalence relations. However, this approach is not directly applicable to property graphs, does not explicitly support overlapping types, and only accounts for limited type hierarchies.

This work has been re-purposed for property graphs in [14]. This performs schema inference by considering either a label-based or a property-based heuristic for merging nodes. The former might lead to losing property co-occurrence information and, thus, to missing node types. The latter might result in inferring spurious types. To palliate the relative drawbacks of each approach, GMMSchema combines both label and property information into base types with respect to which data is clustered.

3 PRELIMINARIES

Assume that \( \mathcal{L}, \mathcal{K}, \mathcal{D}, T \) are pairwise disjoint sets of labels, property names (keys), values, and data types. For a set \( X \), let \( \mathcal{P}(X) \) be the set of all its finite subsets.

Definition 3.1 (Property Graph). A property graph \( G \) is a tuple \((V, E, \rho, \lambda, \sigma)\), where \( V \) and \( E \) are disjoint finite sets of vertices and edges, the incidence function \( \rho : E \rightarrow (V \times V) \) is a total function associating each edge with a pair of nodes, the labeling function \( \lambda : (V \cup E) \rightarrow \mathcal{P}(\mathcal{L}) \) is a partial function\(^1\) associating a vertex/edge with a set of labels, and \( \sigma : (V \cup E) \times \mathcal{K} \rightarrow \mathcal{P}(\mathcal{N}) \) is a partial function associating vertex/edges with properties and, for each property, assigning a set of values from \( \mathcal{D} \).

\(^1\)We denote its extension to element sets as \( \lambda : (\mathcal{P}(V) \cup \mathcal{P}(E)) \rightarrow \mathcal{P}(\mathcal{L}) \).
Let us henceforth fix a property graph $G$. For lack of a standard, we base our definition of a schema for $G$ on [4].

**Definition 3.2 (Base Types).** The set of base types $\mathcal{B}T$ consists of element types. Each element type $b$ is a 4-tuple $(L, K, O, E_b)$, where $L \subseteq L$ is a set of labels, $K \subseteq K$ is a set of property names, $O \subseteq L$ is a subset of optional property names, and $E_b \subseteq \mathcal{B}T$ is the set of element types that $b$ extends.

**Definition 3.3 (Schema Graph).** The schema graph of $G$ is the tuple $(\mathcal{V}T \cup \mathcal{E}T, \rho, \lambda, \sigma)$, where $\mathcal{V}T$ and $\mathcal{E}T$ are disjoint finite sets of vertex and edge types, $\sigma: \mathcal{E}T \to (\mathcal{V}T \times \mathcal{V}T)$ is a total function associating each edge with a pair of nodes, $\lambda: (\mathcal{V}T \cup \mathcal{E}T) \to \mathcal{B}T$ is a total function associating a vertex/edge with a base type, and $\sigma: (\mathcal{V}T \cup \mathcal{E}T) \times \mathcal{K} \to \mathcal{T}$ is function associating vertex/edges with properties and, for each property, assigning its corresponding data type.

**Example 3.4 (Social Network Forums).** Consider a social network graph, similar to those generated by the LDBC Social Network Benchmark [9]. This models a property graph instance, in which persons know each other and create post in various forums. Let us take the below node instance representing a post.

```plaintext
{'Post': { 'content': 'Lorem ipsum, dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.'}}
```

**Listing 1: Dictionary storing a node instance.**

Its base type would be ($\{Post\}, K, \{language, content\}, \emptyset$), where $K = \{creationDate, locationIP, browser, length\}$ contains mandatory properties, $\{language, content\}$ - its optional ones, and the type has no parent, as in Figure 2.

**Gaussian Mixture Models.** Our method uses the Gaussian Mixture Model (GMM) [8], implementing an Expectation Maximization (EM) algorithm to fit a mixture of multi-variate Gaussian distributions. Given a set of node types $x = \{x_1, \ldots, x_n\}$, we fit $x$ as a GMM parametrized by $\theta = \{\theta_1, \ldots, \theta_k\}$, where $\theta_i = (\mu_i, \Sigma_i)$, where $i$ is the $i$-th mixture component. This represents a Gaussian $N(\mu_i, \Sigma_i)$, with prior probability $w_i$, such that $\sum_{i=1}^k w_i = 1$. For every label $L$, we consider the set $C$ of L-labeled nodes and run the GMM algorithm to fit a mixture of two normal distributions. We use the resulting model to split $C$-nodes into two clusters, depending on their similarity to the reference base type. The procedure is then re-iterated to discover further sub-clusters. At the end, we will have computed the set $\mathcal{H}$ of all discovered types and also constructed a dictionary $C_H$, associating every discovered type to the clusters representing its sub-types. This can be directly used to build the schema graph, as will be illustrated.

**4 HIERARCHICAL CLUSTERING**

In this section, we present the GMMSchema algorithm for property graph schema discovery. It partitions the dataset, using unsupervised clustering, such that each computed cluster corresponds to a node type, determined by an unique combination of labels and properties. The method is outlined in Figure 1.

**Algorithm 1 Schema Graph Computation**

```
1: input : $G$ - property graph, $n$ - threshold
2: output : $\mathcal{H}$ - node types clusters
3: function gmm_schema($G, n$)
4: $\mathcal{L}_G \leftarrow \bigcup_{v \in \mathcal{V}} \lambda(v)$ → Set of node labels
5: $\mathcal{H} \leftarrow \emptyset$ → Initial set of all discovered clusters
6: $C_H \leftarrow \emptyset$ → Initial (base type, sub-clusters) dictionary
7: for $L \in \mathcal{L}_G$ do
8:   $C \leftarrow \{v \in \mathcal{V} | L \in \lambda(v)\}$
9:   $\mathcal{H}, C_H \leftarrow$ hierarchical_clustering($C, \mathcal{H}, C_H, n$)
10: return $\mathcal{H}, C_H$
```

First, we pre-process the graph instance $G$ and collect statistics regarding property key frequency, per node label. As described in Algorithm 1, for each label $L$ in the set of all node labels $\mathcal{L}_G$, we compute its corresponding node types and sub-types. Hence, we call the hierarchical inference algorithm on the set $C$ of all nodes whose set of labels contain $L$. For each such node set $C$, the Hierarchical_Clustering algorithm proceeds iteratively, by first constructing a reference base type $b_{ref}$. This intuitively represents the most general type that all nodes in $C$ extend, i.e., the parent cluster for all sub-clusters in $C$. To build it, we consider the set of all cluster labels $\mathcal{L}_C$, as well as the $n$ most frequent properties in $C$. Next, we separate the nodes in $C$ into two clusters, based on how similar their base type is to $b_{ref}$. To this end, for each node, we use the Dice metric to compare the set of its labels and properties to that corresponding to all labels and property keys in $\mathcal{L}_C$ and $K_C$, respectively. These similarities are then regrouped into a feature vector $d$. By fitting a GMM model using $d$ and estimating its parameters with the EM algorithm, we obtain the mixture components $b_1$ and $b_2$. The prediction step allows us to classify the considered data samples and determine whether they fit $b_{ref}$. This amounts to 2-clustering and splitting $C$ into $C_1$ and $C_2$. If the property keys of $C_1$ and $C_2$ overlap, we assign the intersection set to $b_{ref}$. We update the hierarchy dictionary $C_H$ with the information that the base types of $C_1$ and $C_2$ extend $b_{ref}$. For each sub-cluster, we re-iterate the procedure, recompute a reference node, and use it to discover new sub-types.

**Algorithm 2 Cluster Hierarchy Computation**

```
1: input : $C$ - cluster, $\mathcal{H}$ - set of all discovered clusters, $\mathcal{H}_H$ - (base type, sub-clusters) dictionary, $n$ - threshold
2: output : $\mathcal{H}$ - updated set of discovered clusters
3: function hierarchical_clustering($C, \mathcal{H}, n$)
4: $\mathcal{L}_C \leftarrow \lambda(C)$ → Current cluster node labels
5: $K_C \leftarrow$ set of the $n$ most frequent node properties in $C$
6: $b_{ref} \leftarrow (\mathcal{L}_C, K_C, \emptyset, \emptyset)$ → Reference base type for $C$'s nodes
7: for $v \in C$ do
8:   $b \leftarrow \lambda_T(v)$ → Store element type of $v$
9:   $d(v) \leftarrow$ Dice($b, b_{ref}$) → Store similarity metric
10: $\bar{d} \leftarrow$ fit($d$, where $\bar{d} = [\theta_1, \theta_2]$)
11: $C_1, C_2 \leftarrow$ predict($\bar{d}$)
12: if $\bar{d}(C_1) \cap \bar{d}(C_2) \neq \emptyset$ then
13:   $b_{ref} \leftarrow (\mathcal{L}_C, \bar{d}(C_1) \cup \bar{d}(C_2), \emptyset, \emptyset)$
14: if $j \in \{1, 2\}$ and $C_j \neq \emptyset$ then
15:   $C_H[b_{ref}] \leftarrow C_H[b_{ref}] \cup C_j$
16: $\mathcal{H} \leftarrow \mathcal{H} \cup \{C_j, C_j\}$
17: $\mathcal{H}, C_H \leftarrow$ hierarchical_clustering($C_j, \mathcal{H}, C_H, n$)
18: return $\mathcal{H}, C_H$
```
Nodes: 7
Unlabeled: 1
10
3
8
14
59
Labels: 36
Edges: 10
802
0
474
0
1
Node: 3

Their property key intersection gives us the base type of the Post memory, and a 1.5 TB hard drive.

tual Machine with 12 2GHz 64-bits Intel Xeon CPUs, 62 GB of online
3
compatible with Neo4j 4.3.4. The code base is readily available
Our schema inference method is implemented in Python 3 and is
5 EXPERIMENTAL EVALUATION

labels, thus supporting the full property graph model.

Example 4.1 (Social Network - Schema Graph). We focus on

the discovery of the sub-types corresponding to nodes labeled Post. Hence, we run the GMM_Schema algorithm over the graph instance, taking the value \( n = 2 \), for example\(^2\). The corresponding reference node is \( b_{ref} = (\{Post\}, \{creationDate, locationIP\}, 0, 0) \), as creationDate and locationIP are common to all cluster nodes. Using the GMM model, we classify Post nodes into 2-clusters. Their property key intersection gives us the base type of the parent node \( b = (\{Post\}, K, 0, 0) \), where

\[
K = \{creationDate, locationIP, browser, length\}.
\]

Repeating the procedure in each sub-cluster does not infer new types, as all nodes in each share the same properties. Their reference nodes are: \( b_1 = (\{Post\}, K, \{language, content\}, \{b\}) \) and \( b_2 = (\{Post\}, K, \{imageFile\}, \{b\}) \), each representing a new subtype, i.e., Post1 and Post2 in Figure 5. Each of these newly discovered node types will then have the same connectivity characteristics, i.e., attached in/out-going edge types, as their parent node, Post. The discovered schema graph will additionally include two edge subtype edges from Post1 and, respectively, Post2 to Post.

Note that, for ease of exposition, nodes in the running example have single labels only. Our approach, however, covers multiple labels, thus supporting the full property graph model.

5 EXPERIMENTAL EVALUATION

Our schema inference method is implemented in Python 3 and is compatible with Neo4j 4.3.4. The code base is readily available online\(^3\). All experiments were performed on an Openstack Virtual Machine with 12 2GHz 64-bits Intel Xeon CPUs, 62 GB of memory, and a 1.5 TB hard drive.

\(^2\)Note that the value of \( n \) does not impact the overall precision, but only the convergence rate of the method.

\(^3\)https://github.com/naussicaa/pg-schemainference

Figure 2: Ground-Truth Schema for LDBC

The runtime efficiency of our algorithm could be improved by fitting multiple Gaussians during one iteration, instead of just performing 2-clustering. Optimization issues regarding this hyperparameter are left for future work.

Figure 3: GMM_Schema for LDBC

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Node Labels</th>
<th>Edge Labels</th>
<th>Unlabeled Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDBC</td>
<td>1,577,397</td>
<td>8,179,418</td>
<td>7</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>Mb6</td>
<td>486,275</td>
<td>961,271</td>
<td>10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Fib25</td>
<td>982,473</td>
<td>1,625,428</td>
<td>10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Covid19</td>
<td>50,927,757</td>
<td>253,658,374</td>
<td>121</td>
<td>156</td>
<td>474</td>
</tr>
</tbody>
</table>

Figure 4: Dataset characteristics prior to schema discovery.

5.1 Datasets and Metrics

Datasets. We experimentally analyze our approach on several datasets (see Figure 4), described as follows. The LDBC Social Network Benchmark (LDBC) \(^9\) is a synthetic graph, corresponding to the topology of a realistic social network and comes together with a ground-truth schema, capturing single-labeled nodes and type hierarchies. The Mb6 \(^19\) and Fib25 \(^18\) datasets model NeuPrint connectome data and correspond to the mushroom body and, respectively, medulla neural networks in the fruit fly brain. Both these datasets also contain ground-truth schema with diverse properties, as well as multi-labeled nodes. Finally, Covid19 corresponds to the Covid-19 Knowledge Graph \(^7\), which integrates data from the Genotype Tissue Expression (GTEx) the Covid-19 Disease Map (Cord-19), and the bibliographical ArXiv, BioRxiv, MedRxiv, PubMed, and PubMed Central repositories. The graph has no ground-truth schema and is continuously evolving. Hence, the reported results consider the August 2021 version.

Metrics. To assess the quality of the discovered schema, we employ several metrics, as follows. The Adjusted Random Index (Rand Index) measures the similarity between two clusterings, considering all sample pairs and counting those assigned in the same/different clusters in the predicted and true clusterings. The Adjusted Mutual Index (AMI) also measures the similarity between two clusterings, but accounts for potentially unbalanced clusters in the ground truth clustering. Precision captures the proportion of identified types part of the ground truth, while recall measures the performance of binary classification. The F1-score provides an average of precision and sensitivity.

5.2 Evaluation and Discussion

We now present and discuss the results of evaluating GMM-Schema on the LDBC, Mb6, Fib25, and Covid19 datasets.

Schema Quality. As a first indicator of the quality of our schema discovery method, we first analyze the changes in dataset characteristics it generates. Looking at Figure 4 and 5, we notice that,
on average, 2-3 types are discovered per node label, while in the case of edge labels, GMMSchema leads to up to 3 orders of magnitude more edge types (for the fib25 dataset). This is due to the high hierarchy depth of the dataset, which leads to the creation of \( \approx 20 \) new subtype edges. Comparing these numbers to those corresponding to the property-oriented schema inference method in [14] (Figure 6), we observe that, the baseline can infer up to 3 times more node types, up to 3 orders of magnitude more edge types, up to 7 orders of magnitude more subtype edges (for the mb6 dataset), and leads to a hierarchies that can be double the depth of those discovered by GMMSchema. However, looking at the perfect accuracy of our schema discovery method, as given by the precision, recall, and F1-score in Figure 8, we can conclude that many of the inferred nodes are actually redundant and that GMMSchema helps address this by also leveraging the information in node labels. Also, comparing with the numbers reported by the label-oriented heuristic in [14] (Figure 7), we can observe GMMSchema also addresses the missing types due to only considering node label information, by also taking into account their corresponding properties. Hence, our schema discovery method achieves a good balance between these two approaches.

In order to assess the quality of our underlying hierarchical clustering method based on Gaussian Mixture models, we compute its corresponding Rand Index and Adjusted Mutual Information (AMI) scores. To this end, we consider the HDBSCAN [6] clustering algorithm as our baseline, since it is the closest to our approach. As shown in Figure 8, we notice that the two methods agree the most on the largest datasets (Covid19 and LDBC) under both metrics. Also, note that the AMI score is better suited for hard clustering (i.e., assigning a given node to a single cluster), whereas both GMMSchema and HDBSCAN perform soft clustering. Moreover, in the LDBC and Covid19 datasets, it is easier to determine the specific cluster corresponding to a node, as the hierarchy is very shallow and there is little overlap in the labels/properties representing type. Conversely, for Mb6 and Fib25, as there are many such overlaps and as the datasets are smaller, being forced to choose a single "right" cluster for a node is harder and more penalizing.

**Scalability.** Figure 9a captures the overall runtime performance of our GMMSchema method, compared to that of the baseline in [14]. When benchmarking its performance for graphs of up to 2M nodes and edges, GMMSchema exhibits better scalability. For Mb6, GMMSchema is up to 5 times more efficient, while for LDBC and Fib25, it is up to 8 times faster. One explanation for this is that, unlike in our approach, the pre-processing stage of the baseline method requires writing all nodes and edges in a JSON file. Given that Covid19 is rapidly evolving and that simultaneously takes into account both the label and property information on nodes. This addresses the limitations with respect to incomplete/spurious node inference of its closest competitor, whose method is based on MapReduce inference. Preliminary experiments show the superiority of our approach, in terms of accuracy and performance, and highlight the promise of employing statistical methods for schema discovery.

Future extensions consists of further optimizing the approach, inferring cardinality information, and supporting incremental maintenance to preserve schema validity under updates. Also, integrating topological information through graph embeddings could benefit scalability. Finally, investigating how data analysis methods, such as formal concept analysis [17], or recent grammatical inference methods [11] can be applied to property graphs and comparing them to our work opens interesting perspectives.

### Figures

**Figure 5:** Dataset characteristics with GMMSchema discovery.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Node Types</th>
<th>Edge Types</th>
<th>Subtype Edges</th>
<th>Hierarchy Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDBC</td>
<td>17</td>
<td>36</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>Mb6</td>
<td>19</td>
<td>27</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>Fib25</td>
<td>28</td>
<td>106</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6:** Dataset characteristics (property-based inference).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Node Types</th>
<th>Edge Types</th>
<th>Subtype Edges</th>
<th>Hierarchy Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDBC</td>
<td>17</td>
<td>72</td>
<td>51</td>
<td>5</td>
</tr>
<tr>
<td>Mb6</td>
<td>68</td>
<td>795</td>
<td>786</td>
<td>9</td>
</tr>
<tr>
<td>Fib25</td>
<td>47</td>
<td>427</td>
<td>418</td>
<td>8</td>
</tr>
</tbody>
</table>

**Figure 7:** Dataset characteristics (label-based inference).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Rand Index</th>
<th>AMI</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDBC</td>
<td>0.96</td>
<td>0.91</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Mb6</td>
<td>0.79</td>
<td>0.49</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Fib25</td>
<td>0.15</td>
<td>0.41</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Covid19</td>
<td>0.94</td>
<td>0.91</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

**Figure 8:** GMMSchema clustering quality estimates.

(a) LDBC, Fib25, Mb6  
(b) Covid19

**Figure 9:** GMMSchema vs. Baseline total avg. runtimes.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Pre-processing</th>
<th>Sampling</th>
<th>Clustering</th>
<th>Total</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDBC</td>
<td>22.15</td>
<td>3.27</td>
<td>22.02</td>
<td>47.44</td>
<td>830</td>
</tr>
<tr>
<td>Mb6</td>
<td>1.05</td>
<td>0.55</td>
<td>2.95</td>
<td>4.53</td>
<td>48</td>
</tr>
<tr>
<td>Fib25</td>
<td>5.33</td>
<td>1.83</td>
<td>5.83</td>
<td>12.19</td>
<td>76</td>
</tr>
<tr>
<td>Covid19</td>
<td>44.12</td>
<td>1.92</td>
<td>46.04</td>
<td>500.12</td>
<td>266</td>
</tr>
</tbody>
</table>

**Figure 10:** Component-wise and total runtimes (s) for GMMSchema and the baseline on LDBC, Mb6, Fib25, Covid19.

we notice that the most expensive step is generally the clustering one. This is to be expected, as the core hierarchical_clustering function can call itself recursively. Each such time, a Gaussian Mixture Model has to be retrained, in order to split the considered dataset portion into two clusters, by deciding, for each node, whether its base type matches that of the reference base type.

### 6 Conclusion and Future Work

This paper introduces the novel GMMSchema algorithm for schema discovery in the property graph setting. To the best of our knowledge, this is the first approach that targets property graphs and that simultaneously takes into account both the label and property information on nodes. This addresses the limitations with respect to incomplete/spurious node inference of its closest competitor, whose method is based on MapReduce inference. Preliminary experiments show the superiority of our approach, in terms of accuracy and performance, and highlight the promise of employing statistical methods for schema discovery.

Future extensions consist of further optimizing the approach, inferring cardinality information, and supporting incremental maintenance to preserve schema validity under updates. Also, integrating topological information through graph embeddings could benefit scalability. Finally, investigating how data analysis methods, such as formal concept analysis [17], or recent grammatical inference methods [11] can be applied to property graphs and comparing them to our work opens interesting perspectives.
REFERENCES


