PG-HIVE: Hybrid Incremental Schema Discovery for Property Graphs

Sophia Sideri
sophisid@csd.uoc.gr
LIPADE, Université Paris Cité
Paris, France
CSD, University of Crete, Greece

Georgia Troullinou
georgia.troullinou@univ-grenoble-alpes.fr
CNRS, Univ. Grenoble Alpes
Grenoble, France

Elisjana Ymeralli
ymeralli@ics.forth.gr
ICS, FORTH
Heraklion, Greece

Vasilis Efthymiou
vefthym@hua.gr
Harokopio University of Athens
Athens, Greece

Dimitris Plexousakis
dp@ics.forth.gr
ICS, FORTH
Heraklion, Greece

Haridimos Kondylakis
kondylak@ics.forth.gr
ICS, FORTH
Heraklion, Greece

Abstract

Property graphs have rapidly become the de facto standard for representing and managing complex, interconnected data, powering applications across domains from knowledge graphs to social networks. Despite the advantages, their schema-free nature poses major challenges for integration, exploration, visualization, and efficient querying. To bridge this gap, we present PG-HIVE, a novel framework for automatic schema discovery in property graphs. PG-HIVE goes beyond existing approaches by uncovering latent node and edge types, inferring property datatypes, constraints, and cardinalities, and doing so even in the absence of explicit labeling information. Leveraging a unique combination of Locality-Sensitive Hashing with property- and label-based clustering, PG-HIVE identifies structural similarities at scale. Moreover, it introduces incremental schema discovery, eliminating costly recomputation as new data arrives. Through extensive experimentation, we demonstrate that PG-HIVE consistently outperforms state-of-the-art solutions, in both accuracy (by up to 65% for nodes and 40% for edges), and efficiency (up to 1.95x faster execution), unlocking the full potential of schema-aware property graph management.

Keywords

Schema Discovery, Property Graphs, Graph Databases, Clustering, Locality-Sensitive Hashing

1 Introduction

Schema discovery is the process of automatically inferring the underlying structure of data without relying on prior schema information [71, 80]. It has a significant role in modern data management, supporting integration of heterogeneous sources [59, 60, 62], query optimization [34, 74], exploration [68, 81], and data quality assurance [18]. While schema discovery has been widely studied for relational [2, 14, 57] and semi-structured data [31, 33, 72, 84, 102], property graphs (PGs) pose unique challenges.

Property Graphs are directed multigraphs, where both nodes and edges can have labels and properties (i.e., key-value pairs) [6]. Unlike relational databases with fixed schemas, graph databases, including property graphs, are more versatile, do not require an explicit schema [11, 15] to be defined, and can dynamically evolve as more data is ingested. Due to this flexibility, property graphs have been widely adopted in various domains, including social networks, healthcare, bioinformatics, and telecommunications [54, 85].

Although not requiring a fixed schema offers flexibility, it can also lead to a misconception about the structure of the graph database over time, potentially compromising data integrity and usability [67]. To alleviate this problem, schema discovery is crucial for understanding the structure and semantics of property graphs. In the context of a property graph, we define its schema as the collection of node and edge types, together with their associated properties, constraints, and cardinalities. A *type* refers to a category of nodes or edges that share a common structural information.

The problem. In property graphs, the absence of an explicit schema makes understanding the data's structure challenging. Nodes and edges can carry arbitrary or missing properties, lack consistent labels, and provide no guarantees of uniformity [58]. As graphs evolve, this ambiguity complicates integration, querying, and data quality assurance. Crucially, we cannot assume in advance what constitutes a type or how types should be distinguished, meaning that the data remains opaque until explicitly analyzed. The goal of schema discovery, therefore, is to automatically infer the types of nodes and edges, their properties, and the structural constraints that govern them—while ensuring accuracy, efficiency, and adaptability to dynamic graph scenarios.

Challenges. To address the problem of schema discovery, efficient processing methods are needed [80]. Although schema discovery has been studied for semi-structured RDF data [53, 56], PGs pose additional challenges [58]. PGs refer to a broad class of enriched graph structures allowing many technical interpretations in different software systems. These interpretations are often incompatible and based on different assumptions [82]. Unlike RDF [10, 61, 63], PGs lack a widely accepted schema language and allow nodes and edges to have arbitrary sets of properties, multiple labels, or none at all [86]. Hence, RDF schema discovery methods [20, 25, 27, 53, 55, 56] cannot be applied directly for inferring the schema of PGs [3]. Nevertheless, a small number of approaches for PG schema discovery have already emerged - GMMSchema [16] and its demo (DiscoPG [15]), as well as SchemI [64]. Although they produce relatively good results under ideal conditions, they employ methods that require *fully labeled data* and they are less effective in noisy or incomplete datasets (§ 5).

When data is incomplete, noisy, or integrated from heterogeneous sources, schema discovery becomes a rather difficult problem. Missing properties, inconsistent labels, and conflicting structural patterns obscure the identification of meaningful

EDBT '26, Tampere (Finland)

10.48786/edbt.2026.36

types. Additionally, we cannot assume that data is missing without domain knowledge, as it can vary in structure. In these cases, clustering plays a central role in grouping nodes and edges based on shared properties, labels, or semantics [16, 105]. Without clustering, exhaustive pairwise comparisons would be required, which is computationally infeasible for large graphs. The challenge, however, lies in choosing a clustering strategy that remains effective across diverse datasets. Because each graph exhibits different structural and semantic characteristics, no single method can generalize well. Consequently, identifying the appropriate number of clusters—and thus the correct number of types —remains an open problem, with current techniques offering only approximate solutions [100, 101].

The solution. In this paper, we present **PG-HIVE**, a new open-source approach for schema discovery in large property graph datasets. PG-HIVE is designed to infer the schema even in the absence of explicit labels. It uses both property and label information, when available, in an adaptive manner. To group structurally and semantically similar nodes and edges, clustering is performed using Locality-Sensitive Hashing (LSH) [32, 39, 92, 98]. The clustering parameters are dynamically adapted to the specific characteristics of each dataset. This forms a *hybrid schema discovery* approach, as it combines labels, properties, and graph structure to identify types, instead of relying only on the labels or the properties.

Unlike prior works, which focus only on the type extraction, PG-HIVE extracts a detailed schema by identifying (a) node types, (b) edge types, (c) property data types, (d) key constraints, and (e) cardinalities, thus capturing a richer schema structure. Moreover, the data can be processed incrementally, in batches, allowing the schema to be continuously updated as new data is added without requiring a full recomputation, making PG-HIVE efficient, modular, and suitable for real-world environments. We provide a thorough experimentation using eight datasets–four real and four synthetic–under varying noise levels, using two LSH-based clustering methods [65]. We explore different configurations, analyzing their impact on schema discovery compared to our adaptive approach. Then, we compare PG-HIVE against state-of-the-art methods (GMMSchema [16] and SchemI [64]), showing that PG-HIVE consistently has better accuracy in schema discovery, and it is particularly superior under noisy conditions and when label information is missing, where other approaches degrade. Across all datasets, PG-HIVE achieves up to 65% higher accuracy for nodes, 40% for edges, and 1.95x faster execution compared to existing methods, demonstrating robustness and efficiency.

To the best of our knowledge, PG-HIVE is the first system that supports an incremental and hybrid schema discovery for property graphs, adapting to the individual datasets' characteristics, and going beyond simple node type discovery.

Contributions. In summary, our contributions are the following:

- We introduce a hybrid schema discovery approach for property graphs that infers node and edge types by exploiting structural patterns and label information, remaining effective under noisy, incomplete, or unlabeled data.
- We introduce an adaptive LSH-based clustering strategy that adjusts its parameters based on dataset characteristics.
- We extend type discovery with schema constraints, including mandatory and optional properties, inferred property data types, and edge cardinalities, producing richer graph schemas.
- We support incremental schema discovery with monotonic guarantees, diminishing the need for full recomputation.

Table 1: Schema discovery approaches on property graphs.

	SchemI [64]	GMMSchema [16]	DiscoPG [15]	PG-HIVE (ours)
Label Independent	×	×	×	✓
Multilabeled Elements	×	✓	✓	✓
Schema Elements	Nodes & Edges	Nodes only	Nodes, queries associated Edges	Nodes, Edges & constraints
Constraints	×	×	×	✓
Incremental	×	×	✓	✓
Automation	✓	✓	✓	✓
Notes	Cannot handle missing labels	GMM, cannot handle missing labels	Demo of GMM-Schema	LSH and fine tuning

- We perform an extensive experimental evaluation on large real and synthetic datasets, demonstrating that PG-HIVE consistently outperforms state-of-the-art methods, particularly under high noise and limited label availability.

Outline. Section 2 discusses related work. Section 3 gives a conceptual description of the problem. Section 4 describes the methodology. Section 5 shows our experimental evaluation. Section 6 discusses conclusions and future work.

2 Related Work

Although schema discovery for big data has been extensively studied and explored, especially for RDF [20, 25, 27, 55, 56], tabular [2, 14, 57] and JSON [31, 33, 72, 84, 102] data, only a limited number of approaches [15, 16, 64] have tried to address the complexity and expressiveness of the property graph data model.

RDF approaches. RDF-based methods cannot be applied directly to PGs due to fundamental differences in the data model [3, 6]. RDF edges are labeled but have no properties, while PG edges may carry properties as key–value pairs. Additionally, PG nodes can have multiple or no labels, indicating their type, while in RDF, types are described by the relation `rdfs:Class`. Finally, RDF is fundamentally based on ontologies, whereas for PGs, a standard schema definition is not yet available.

PG schema definitions. Focusing more on the definition of schema for PGs, this has been defined differently across works, adding complexity to schema discovery. For example, approaches like [64] treat each distinct label (e.g., `Person`, `Student`) as a separate type, while several datasets [35, 45, 49, 94, 95] define types by sets of co-occurring labels, e.g., {`Student, Person`} versus {`Athlete, Person`}. In integration scenarios, different datasets may use distinct labels for the same conceptual entity (e.g., `Organization` and `Company`) or even employ labels in different languages. On the other hand, hierarchical datasets may flatten all nodes under a single generic label (e.g., `Thing` in CIDOC-CRM [28] datasets). These variations lead to ambiguities and inconsistencies, especially when labels are reused across domains (e.g., `Actor` as a person or as an organization) and complicate schema discovery.

PG schema discovery approaches. Table 1 summarizes the characteristics of the approaches for schema discovery for property graphs. One of the earliest works, SchemI [64], assumes that all nodes and edges are labeled, and groups similar node types based on shared labels. However, this method is limited to datasets with complete type label declarations, and it cannot infer schemas when labels and properties are missing or inconsistent. A more advanced approach, GMMSchema [16], introduces hierarchical clustering based on Gaussian Mixture Models (GMM) [13] to group nodes by analyzing labels and property distributions. However, it has several limitations: (i) it focuses only on node clustering and does not infer relationships between clusters (i.e.,

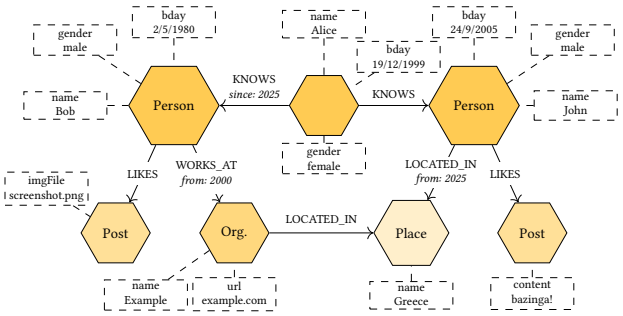

Figure 1: Example Property Graph.

edge types), (ii) it assumes fully labeled datasets, (iii) it is not designed to handle missing values or noisy properties, which are common in practice, and (iv) it applies sampling techniques to improve performance on large graphs, which impacts the completeness or precision of the inferred schema. Additionally, the demo of GMMSchema, DiscoPG [15], incorporates an incremental approach that uses memorization to avoid unnecessary search for types that have already been found. However, it remains fundamentally based on the GMMSchema and queries the associated edges of the discovered nodes, and might lose information of updated clusters.

In summary, most existing approaches either rely on fully annotated datasets, focus on type discovery without modeling constraints, or require manual input. They do not support incremental updates or adapt well to noisy and evolving data. **PG-HIVE** addresses these limitations by providing a hybrid incremental schema discovery framework that models more information about the schema, rather than only the types, such as constraints and cardinalities, even in the presence of incomplete or evolving data.

3 Preliminaries

A property graph consists of nodes, edges, and associated properties stored as key-value pairs. Nodes represent entities or objects (e.g., `people, places`), edges define relationships between nodes (e.g., `is_located_in, works_at`), and properties characterize both nodes and edges (e.g., `name, age, date`).

Although the concept of PGs was first introduced by Rodrigues and Neubauer [83], several works have attempted to provide either informal (e.g., [1, 78, 89]) or formal (e.g., [4, 6–8, 17, 42, 43, 82, 88]) definitions of the property graph data model.

Theoretical concepts. In this work, we employ the formal PG-Schema model [8], which distinguishes between the data graph and its schema graph. PG-Schema builds upon the foundations of the property graph data model and PG-Keys [9] by introducing *keys* and their *constraints* [97]. Henceforth, we use the term *property keys* exclusively for attribute names (e.g., `gender`, `name`, `since`), and we refer to PG-Keys elements as *constraints*.[1] PG-Schema uses GQL's [41] predefined data types, but it can definitely be extended to support a wider range of data types like `STRING`, `BOOLEAN`, `INT`, `DOUBLE`, `TIMESTAMP`, or `DATE`, and can annotate the properties as `MANDATORY` or `OPTIONAL` to capture completeness.

In our context, let $\mathcal{L}$ be the universe of labels, $\mathcal{K}$ be the universe of property keys, and $\mathcal{D}$ be the universe of property values, and $\mathcal{DT}$ be a finite set of data types and Req = {MANDATORY, OPTIONAL}. We define a property graph, and an node/edge type as follows:

Definition 3.1 (Property Graph [8])**.** A property graph is a tuple $G = (V, E, \rho, \lambda, \pi)$, where

- V and E are disjoint finite sets of nodes and edges, respectively,
- $\rho : E \to (V \times V)$ is a total function mapping edges to ordered pairs of nodes,
- $\lambda : (V \cup E) \to 2^{\mathcal{L}}$ assigns to each node/edge a finite set of labels (possibly empty), and
- $\pi : (V \cup E) \times \mathcal{K} \to \mathcal{D}$ is a partial function returning the (single) value of key $k \in \mathcal{K}$ on element $x \in (V \cup E)$, if present.

Definition 3.2 (Node Type)**.** A node type is a tuple $V_s = (\lambda_n, \pi_n)$, where $\lambda_n \subseteq \mathcal{L}$ is a finite (possibly empty) set of labels associated with the node type, and $\pi_n \subseteq \mathcal{K} \times \mathcal{DT} \times \{\texttt{MANDATORY}, \texttt{OPTIONAL}\}$ is a finite set of property descriptions.

Definition 3.3 (Edge Type)**.** An edge type is a tuple $E_s = (\lambda_e, \pi_e, \rho_e, C_e)$, where:

- $\lambda_e \subseteq \mathcal{L}$ is a finite set of edge labels,
- $\pi_e \subseteq \mathcal{K} \times \mathcal{DT} \times \{\texttt{MANDATORY}, \texttt{OPTIONAL}\}$ is a finite set of edge property descriptions,
- $\rho_e = (V_{ss}, V_{st})$ is the ordered pair of source and target node types associated with the edge type,
- $C_e : E_s \to N_1 \times N_2$ assigns a cardinality constraint to the edge type, where $N_1, N_2 \in \mathbb{N}$ denote the minimum and maximum cardinality from source to target.

Henceforth, we refer to *types* as both nodes and edge types. Then, we assemble the aforementioned and construct the schema as:

Definition 3.4 (Schema Graph)**.** A schema graph is a tuple $S_G = (V_s, E_s)$, where V_s is a finite set of node types, and E_s is a finite set of edge types.

Example 1. Figure 1 shows a simple property graph with node types: `Person`, `Organization`, `Post`, `Place`, and edge types: `KNOWS`, `LIKES`, `WORKS_AT`, and `LOCATED_IN`. At the data level, nodes like *Alice*, *Bob*, and *John* instantiate these types, while the node *Alice* appears without a label, illustrating unlabeled instances. Edges such as `KNOWS`(*Alice*, *John*) or `WORKS_AT`(*Bob*, *Organization*) represent relationships, and attributes (e.g., `name`, `gender`, `bday`) appear as dashed literal boxes.

Different works interpret types in PGs differently. Some distinguish types explicitly by labels, the set of labels, or by their semantic meaning. In this paper, we adopt the PG-Schema [8] model and define the types formally as in Def. 3.2 and Def. 3.3, allowing different combinations of labels to correspond to different types. Therefore, we rely on patterns Def. 3.5 and Def. 3.6 to capture the variety of representations found in the datasets and support schema inference even in the absence of explicit or consistent label annotation. A type might be associated with multiple *patterns*, allowing us to capture and handle noisy or incomplete data.

Definition 3.5 (Node Pattern)**.** A node pattern is a tuple $T_{Np} = (L, K)$, where $L \subseteq \mathcal{L}$ is a set of labels and $K \subseteq \mathcal{K}$ is a set of property keys. Two node patterns $T_{Vp} = (L, K)$ and $T_{Vp'} = (L', K')$ are distinct, if $L \neq L'$ and $K \neq K'$.

Definition 3.6 (Edge Pattern)**.** An edge pattern is a tuple $T_{Ep} = (L, K, R)$, where $L \subseteq \mathcal{L}$ is a set of labels of an edge, $K \subseteq \mathcal{K}$ is a set of property keys of the edge, and $R = (L_s, L_t)$ where $L_s, L_t \subseteq \mathcal{L}$ specifies the source and target node label, respectively. Two edge patterns $T_{Ep} = (L, K, R)$ and $T_{Ep'} = (L', K', R')$ are distinct, if $L \neq L'$, $K \neq K'$ and $R \neq R'$.

[1] By contrast, PG-Keys [9] introduces *keys*, which refer to schema level identification and constraints that assign uniqueness, mandatory properties, edges, or paths.

Example 2. Based on Def. 3.5, 3.6, the patterns of the example in Figure 1 are the following:
Node patterns:
- T_{Vp1} = ({**Person**}, {name, gender, bday})
- T_{Vp2} = ({}, {name, gender, bday})
- T_{Vp3} = ({**Org.**}, {name, url})
- T_{Vp4} = ({**Post**}, {imgFile}),
- T_{Vp5} = ({**Post**}, {content})
- T_{Vp6} = ({**Place**}, {name})

Edge Patterns:
- T_{Ep1} = ({**KNOWS**}, {since}, ({**Person**}, {**Person**}))
- T_{Ep2} = ({**KNOWS**}, {}, ({**Person**}, {**Person**}))
- T_{Ep3} = ({**LIKES**}, {}, ({**Person**}, {**Post**}))
- T_{Ep4} = ({**WORKS_AT**}, {from}, ({**Person**}, {**Org.**}))
- T_{Ep5} = ({**LOCATED_IN**}, {}, ({**Org.**}, {**Place**}))
- T_{Ep6} = ({**LOCATED_IN**}, {from}, ({**Person**}, {**Place**}))

We use the term *patterns* to refer to both node and edge patterns. Multiple patterns can correspond to the same type, when their associated set of properties differ; in the case of edges, this may also occur when their source or target label sets differ. We refer to the set of all such patterns of a type $\tau \in V_s \cup E_s$ as *type patterns* of τ, formally defined as $P_v = \{(L, K) \mid L = \lambda(\tau)\}$ and $P_e = \{(L, K, R) \mid L = \lambda(\tau),\ R = (\lambda(s), \lambda(t))\}$. Thus, two instances with the same labels L but different property sets K; and, for edges, the same endpoints R, correspond to the same type but belong to different patterns.

The schema discovery problem in PGs. The problem of schema discovery in property graphs can be informally described as: *"Given a property graph G of arbitrary size and structure, with missing type information, heterogeneous properties, and frequent updates, infer the schema graph S_G efficiently and accurately."*
Several challenges arise in this context:

- **Label heterogeneity and ambiguity.** Labels are often used inconsistently [30, 52, 104]. Multiple labels may describe the same type (e.g., `Actor` and `Person`), or a single label may represent different entities (e.g., `Actor` as `Person` or `Organization`). Structural similarity alone cannot capture such semantic relations (e.g., `Intern` as a subtype of `Employee`).
- **Efficiency.** Naïve pairwise node comparisons are computationally prohibitive for large graphs.
- **Evolving datasets.** Since real-world graphs evolve, recomputing schemas from scratch is inefficient. Incremental updates are required to maintain performance.
- **Schema constraint level.** Schemas must balance strictness and flexibility [79]. Strict schemas enforce structure, while loose schemas allow variation but reduce precision. PG-Schema [8] captures this trade-off with `STRICT` and `LOOSE` modes.

4 The PG-HIVE Approach

To address the aforementioned limitations, we introduce PG-HIVE. Its pipeline in Figure 2 includes: (a) data loading, (b) preprocessing, (c) clustering to group similar nodes and edges, (d) extracting schema types, (e) inferring property constraints, (f) property data types, (g) cardinalities, and (h) serializing the schema in XSD [87] and a PG-Schema [8]. Optionally, this pipeline can be executed incrementally.

Algorithm 1 outlines the process, handling the input graph G in batches ($Gs_1, \ldots, Gs_n$). The main pipeline (Lines 3-6) is executed for each batch, with optional post-processing (Lines 7-9). The algorithm starts with loading the nodes and edges, and creating the representative vectors (Lines 3-4). Each node's vector consists of a fixed-dimensional Word2Vec embedding of its labels and a binary vector indicating the presence or absence of

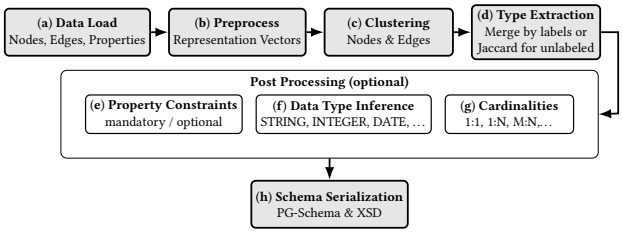

Figure 2: PG-HIVE process.

Algorithm 1: PG-HIVE Schema Discovery

Input: PG stream $G = \{Gs_1, Gs_2, \ldots, Gs_n\}$, Jaccard similarity threshold $\theta \in [0, 1]$, Boolean flag `runPostProcessing`
Output: Final schema graph S_G

```
1  S_G ← ∅
2  for each Gs_i ∈ G do
3      D ← loadNodesAndEdges(Gs_i)
4      X, b, T ← preprocess(D)
5      C ← LSHClustering(X, b, T)
6      S'_G ← extractTypes(C, S_G, θ = 0.9) ;    // Algorithm 2
7      if runPostProcessing or i = n then
8          inferPropertyConstraints(S'_G)
9          inferDataTypes(S'_G)
10         computeCardinalities(S'_G)
11     S_G ← updateSchema(S'_G)
12 return S_G
```

each property. Similarly, the edges have three fixed-dimensional Word2Vec embeddings (one for its label and two for the source and target labels), as well as the binary vector indicating its properties. The vectors are used then for the LSH clustering (Line 5). An important component is the type extraction step (Line 6), where the types inferred in the current batch are merged and integrated into the final schema S_G. If a new cluster has the same label as an existing type, it merges with the corresponding type. For unlabeled clusters, we compute the Jaccard similarity between their properties against those of existing types. The remaining unmerged clusters are appended to the new schema as new `ABSTRACT` [8] types. Optionally, post-processing steps can be applied after each iteration to infer constraints and cardinalities. In the sequel, we analyze in detail each step.

4.1 Data Loading & Representation

Firstly, we load the nodes and edges, plus their properties, from a PG storage system (e.g., Neo4j [1]) and transform them into a vector representation (e.g., DataFrame [12, 103]), using a single query to ensure a similar structure. Let G be our property graph, where $V = \{v_1, \ldots, v_n\}$ denotes the set of nodes and $E = \{e_1, \ldots, e_m\}$ the set of edges in the graph. Each node v_i has a set of properties $\mathcal{K}_{v_i} = \{k_1, \ldots, k_k\}$, and each edge e_j a set of properties $\mathcal{K}_{e_j} = \{k_1, \ldots, k_l\}$. Edges are also associated with a pair of nodes (V_s, V_t) (Def. 3.1).

Representation. We transform the data into a machine-readable format (e.g., one-hot encoding) that serves as input to the clustering process. Each node v_i is described by a vector $\mathbf{f}_{v_i} \in \mathbb{R}^{d+K}$, where d is the dimension of the Word2Vec [70] embedding, that will represent the label(s) of each node and K the total number of distinct properties of our dataset. The vector $\mathbf{f}_{v_i}$ is the concatenation of the embedding $\mathbf{w}_{v_i} \in \mathbb{R}^d$ and the binary vector

$\mathbf{b}_{v_i} \in \{0,1\}^K$ that indicates the presence or absence of each property in each instance. We train a Word2Vec model on the set of node and edge labels observed in the dataset to ensure consistent semantic embeddings across identical label sets. Similarly, each edge e_j is represented by a vector $\mathbf{f}e_j \in \mathbb{R}^{3d+Q}$, composed of three Word2Vec embeddings $\mathbf{w}_{e_j}, \mathbf{w}_{src_j}, \mathbf{w}_{tgt_j} \in \mathbb{R}^d$, for the edge type, the source node type, and the target node type, respectively, and a binary vector $\mathbf{b}_{e_j} \in \{0,1\}^Q$, for the edge properties, where Q is the total number of distinct edge properties of our dataset.

In the case where the label is absent, we use a zero vector of the same size as the Word2Vec vector (Example 3). On the other hand, if we have multiple labels on an instance, we sort them alphabetically for uniformity and then concatenate them as one and finally transform them as the rest. These hybrid vectors combine both the semantic information of the labels and the structural representation for nodes and edges [13]. This representation prevents semantically different nodes, or edges, from being merged due to their same structure. On the other hand, when an instance has multiple labels, we assume the sorted concatenation of them as a unique label. This helps identify types (nodes or edges) with multiple labels, by having the same Word2Vec if they have multiple same labels or if some of the labels are different.

Example 3. Let's consider the global property keys set $\mathcal{K}_n$ = {`imgFile`, `content`, `name`, `url`, `bday`, `gender`}. The node "Bob" of type **Person** with properties {`name`, `gender`, `bday`}, in a Word2Vec size of 5 is represented as the concatenation of:

$$\underbrace{[0.12, 0.85, -0.33, 0.47, 0.19]}_{\text{Word2Vec("Person")}} \parallel \underbrace{[0, 0, 1, 0, 1, 1]}_{\text{binary props}}$$

For an unlabeled node like "Alice", the representation would be like:

$$\underbrace{[0, 0, 0, 0, 0]}_{\text{Word2Vec (unlabeled)}} \parallel \underbrace{[1, 0, 1, 0, 1, 1]}_{\text{binary props}}$$

Edges have the set of properties $\mathcal{K}_e$ = {`since`, `from`}. We represent "WORKS_AT" from **Person** to **Org.** with property {`from`} as:

$$\underbrace{[0.44,\ -0.11,\ 0.93,\ 0.05,\ -0.27]}_{\text{Word2Vec("WORKS_AT")}} \parallel \underbrace{[0.12,\ 0.85,\ -0.33,\ 0.47,\ 0.18]}_{\text{Word2Vec("Person")}}$$

$$\underbrace{[0.66,\ 0.10,\ -0.21,\ 0.73,\ -0.08]}_{\text{Word2Vec("Org.")}} \parallel \underbrace{[0, 1]}_{\text{binary props}}$$

4.2 Clustering

To identify the node and edge types, we employ Locality-Sensitive Hashing (LSH) [40, 48, 65] for clustering their vector representation. LSH has been utilized for several applications such as nearest neighbour search [36, 46, 48, 96], document clustering [21, 22], and RDF data management [5, 53]. In our context, we use LSH to cluster nodes and edges with highly similar vectors, indicating that they belong to the same type. LSH is particularly well-suited for this task as it approximates similarity across large datasets without requiring expensive pairwise comparisons, thus ensuring efficiency.

We examine two LSH approaches: Euclidean LSH (ELSH) [32, 92] also known as p-stable or bucketed random projections, which is tailored to ℓ_2 distance, and MinHash LSH [26, 65, 92], used for Jaccard similarity over sets. Although they can be applied better to different cases [99], in our heterogeneous context, they are both efficient (§ 5) [50].

ELSH. ELSH [65] is particularly suitable for our feature vectors, which combine label embeddings with binary property indicators. These vectors appear in a high-dimensional numeric space, where Euclidean distance works well. ELSH has two parameters [73, 92]: a) the *bucket length* $b > 0$, which controls the width of the hash buckets and thus the granularity of similarity (larger bucket ⇒ more collisions, higher recall but lower precision), and b) the *number of hash tables* $T \in \mathbb{N}$, which affects recall (higher T ⇒ more chances to collide) but increases runtime. ELSH is effective for real-valued embeddings, but requires parameter tuning to balance precision and recall [50, 99].

MinHash. MinHash LSH [65], approximates the Jaccard similarity between sets. It creates signatures for each set so the probability of two sets to collide in a hash function is equal to their Jaccard similarity. In our context, when nodes and edges are modeled simply as sets of properties, MinHash provides a simple similarity approach. As a parameter, MinHash only requires the number of hash tables T [73, 92], which affects the trade-offs between recall and efficiency. MinHash performs well for sparse, set-like data, but it is less suitable when vectors include continuous components such as embeddings [50, 99].

Collision probabilities and parameter effects. Each dataset requires different parameters to achieve the best possible clustering [77], as it depends on the structural and statistical characteristics of the graph. The labels and the sparsity of properties affect the optimal bucket length and number of hash tables. For instance, denser graphs with many overlapping properties benefit from finer bucket lengths to better separate similar types, while sparser graphs require longer buckets to avoid mixing types. Similarly, the number of hash tables balances recall with computational efficiency.

Let r_{in} be the distance between items of the same type and $r_{\text{out}} > r_{\text{in}}$ the distance between different types. For **ELSH**, the collision probability of two points with distance d in one table is $p_b(d)$, a decreasing function of d determined by the bucket length b [32]. With T independent hash tables combined under the OR rule, the probability that two vectors collide in at least one table is $P_{b,T}(d) = 1 - \left(1 - p_b(d)\right)^T$ [65], which decreases as d grows and increases with T. Hence, decreasing b (narrower buckets) or increasing T increases selectivity and so improves the separation between $P_{b,T}(r_{\text{in}})$ and $P_{b,T}(r_{\text{out}})$, leading to clusters with fewer mixed types (i.e., fewer false negatives) [40, 47].

For **MinHash**, the collision probability for two sets A, B, is the probability that their signatures agree in one hash function equals their Jaccard similarity, $\Pr[h(A) = h(B)] = J(A,B) = \frac{|A \cap B|}{|A \cup B|}$ [65]. With T hash functions, the collision rate remains $J(A,B)$, and this estimation becomes more reliable as T increases. In practice, similar sets collide often, and dissimilar ones rarely, making MinHash a simple similarity approach.

Adaptive parameterization. To take into account the heterogeneity of property graphs and avoid manual tuning of LSH, we introduce an adaptive strategy for selecting the key parameters b and T, leveraging the characteristics of the dataset.

Before clustering, we examine a small portion of the graph to infer key characteristics, like how sparse the dataset is or how many labels the dataset has. This helps us guide the parameter selection for clustering. More specifically, we randomly sample 1% of the graph, or at least 10k nodes (whichever is larger), and compute the Euclidean distances between the sampled elements and take their average as the distance scale μ. This ensures that the bucket length is adapted to the actual distance of the data. Then, we set the bucket width proportional to μ, as: $b_{\text{base}} = 1.2 \cdot \mu$. The factor 1.2 avoids overfragmentation when the sample distance is small.

To also consider the number of labels, we refine b using the number of distinct labels L observed in the dataset: $b = b_{\text{base}} \cdot \alpha$, where $\alpha = 0.8$ for $L \leq 3$, $\alpha = 1.0$ for $4 \leq L \leq 10$, and $\alpha = 1.5$ for $L > 10$. This heuristic reflects that graphs with few labels require tighter buckets to keep types distinct, while graphs with many labels benefit from wider buckets to avoid overfragmentation. At this point, we prefer more separate types, as we are going to perform a merging step afterwards (§ 4.3).

The number of hash functions is scaled according to the dataset size N and then adjusted according to the number of labels L. Smaller graphs use fewer hash functions, while larger graphs demand more to distinguish the different types. Graphs with few labels need more hash functions to prevent mixed elements from being grouped, while graphs with many labels can separate from the label information and therefore need fewer functions. The value of T is heuristically determined by $T = b_{\text{base}} \cdot \max(5, \alpha \cdot \min(25, \log_{10} N))$, for nodes and $T = b_{\text{base}} \cdot \max(3, \alpha \cdot \min(20, \log_{10} E))$ for edges.

These heuristics ensure a balance between precision and recall across graphs of varying size and label diversity, while avoiding both over-fragmentation and excessive merging. Regardless of the adaptive approach, users can always provide their own LSH parameters, that suit the respective dataset.

Practical ranges. Empirically for our experiments, $\alpha \in [0.5, 2]$ and $T \in [15, 35]$ work well across datasets; edges benefit from slightly smaller $\alpha \in [0.5, 1.5]$, due to smaller vector representations, however, we explore the space of alternatives in (§ 5).

Clustering result. A cluster is a group of nodes or edges that have similar structural characteristics, such as properties and labels. From these structural similarities (i.e., patterns Def. 3.5, 3.6), we define a type pattern from the cluster representative. Additionally, we maintain the associated instances of each cluster that represent this type. This way, we specify a type pattern and collect the set of instances assigned to it.

Cluster representative. Each cluster C (of nodes or edges) is summarized by a *representative pattern* $rep(C) = (L, K, R)$, where:

- L is the set of all labels that appear in at least one instance of C,
- $R = (L_s, L_t)$ for edges, is the set of all labels that appear in at least one instance on the source and target label sets, respectively, and
- P is the set of properties observed across the instances of C.

We refer to the type patterns emerging from the clustering process, each representing a potential schema element, as *candidate types*. We consider our candidate types (node or edges) T as the cluster C representative itself. These clusters form the basis for the type inference, as they capture the structure of the data, and they are used in the next steps.

Example 4. Nodes "Bob" and "John" (Figure 1) both have the label `Person` and have the same structure (`name`, `gender`, `bday`), so they are assigned together. The unlabeled node "Alice" has the same property vector, but it is not placed in the same cluster. The two `Post` nodes are initially assigned in different clusters due to structural differences—one includes an `imgFile` property while the other a `content`. `LIKES` edges are grouped together, as they consistently link a `Person` node to a `Post`.

4.3 Extracting Types

After clustering, we refine the candidate node and edge types by merging clusters that correspond to the same schema type. We distinguish two cases. If a cluster has at least one label, we treat it as *labeled*, otherwise as *unlabeled*, so we can handle them differently and avoid mixing different types. An overview of the heuristics used is described in Algorithm 2.

Merging labeled clusters. Node and edge clusters that have the same label(s) are merged directly, as they represent the same type, and preserve their information (Lemma 1 and Lemma 2).

Merging unlabeled clusters. For unlabeled clusters, we compare them against labeled ones using the Jaccard similarity of their property sets. Given two clusters C_1, C_2 with type patterns $T_1 = (\mathcal{L}_1, \mathcal{K}_1)$ and $T_2 = (\mathcal{L}_2, \mathcal{K}_2)$, the similarity is $J(C_1, C_2) = \frac{|\mathcal{K}_1 \cap \mathcal{K}_2|}{|\mathcal{K}_1 \cup \mathcal{K}_2|}$. If $J(C_1, C_2) \geq \theta$ (we set $\theta = 0.9$), they are merged into $T_M = (\mathcal{L}_1 \cup \mathcal{L}_2,\ \mathcal{K}_1 \cup \mathcal{K}_2)$. We use a high similarity threshold to avoid over-merging; lowering θ would increase recall but mix types and will decrease precision.

Lemma 1 (Monotonicity of merging node types)**.** Let $T_{N1} = (\mathcal{L}_1, \mathcal{K}_1)$, $T_{N2} = (\mathcal{L}_2, \mathcal{K}_2)$, and $T_{NM} = (\mathcal{L}_1 \cup \mathcal{L}_2,\ \mathcal{K}_1 \cup \mathcal{K}_2)$ be the merge of T_{N1} and T_{N2}. Then, $\mathcal{K}_i \subseteq \mathcal{K}_M$ and $\mathcal{L}_i \subseteq \mathcal{L}_M$ for $i \in \{1, 2\}$. Thus, no node property or node label is lost.

Proof. By construction: $\mathcal{L}_M = \mathcal{L}_1 \cup \mathcal{L}_2$, $\mathcal{K}_{e,M} = \mathcal{K}_{e,1} \cup \mathcal{K}_{e,2}$. □

Lemma 2 (Monotonicity of merging edge types)**.** Let $T_{E1} = (\mathcal{L}_1, \mathcal{K}_{e,1}, \mathcal{R}_1)$, $T_{E2} = (\mathcal{L}_2, \mathcal{K}_{e,2}, \mathcal{R}_2)$ with $\mathcal{R}_1 = (\mathcal{L}_{s,1}, \mathcal{L}_{t,1})$, and $\mathcal{R}_2 = (\mathcal{L}_{s,2}, \mathcal{L}_{t,2})$ and $T_{EM} = \big(\mathcal{L}_1 \cup \mathcal{L}_2,\ \mathcal{K}_{e,1} \cup \mathcal{K}_{e,2},\ (\mathcal{L}_{s,1} \cup \mathcal{L}_{s,2},\ \mathcal{L}_{t,1} \cup \mathcal{L}_{t,2})\big)$ be the merge of T_{E1} and T_{E2}. Then, $\mathcal{L}_i \subseteq \mathcal{L}_M$, $\mathcal{K}_{e,i} \subseteq \mathcal{K}_{e,M}$, and $\mathcal{R}_1, \mathcal{R}_2 \subseteq \mathcal{R}_M$. Thus, no label, property, or endpoint is lost.

Proof. By construction: $\mathcal{L}_M = \mathcal{L}_1 \cup \mathcal{L}_2$, $\mathcal{K}_{e,M} = \mathcal{K}_{e,1} \cup \mathcal{K}_{e,2}$, and $\mathcal{R}_M = \mathcal{R}_1 \cup \mathcal{R}_2 = (\mathcal{L}_{s,1} \cup \mathcal{L}_{s,2},\ \mathcal{L}_{t,1} \cup \mathcal{L}_{t,2})$. □

This ensures that in the merging step properties and labels are not eliminated, only added.

Unmatched unlabeled clusters. If an unlabeled cluster is not merged with labeled ones, we then merge it with other unlabeled clusters using the same Jaccard approach. Finally, if it remains unlabeled, we keep it as an `ABSTRACT` type (as in the PG-Schema [8]).

Edges. We merge edges only by label and get the set of source and target node types to define the connectivity $\rho_s(e_t) = (t_{\text{src}}, t_{\text{tgt}})$.

Example 5. The nodes "Bob", "John", and "Alice" (Figure 1) all have identical property sets (`name`, `gender`, `bday`). The cluster containing "Alice" lacks a type label, but it has high Jaccard similarity with the cluster labeled as `Person`, which includes "Bob" and "John". Based on this similarity, we merge Alice's cluster with the `Person` cluster. In another case, the two `Post` nodes have different structures. However, since they both have the same label, "`Post`", we merge them.

4.4 Post Processing

As mentioned before, we compute, optionally, more characteristics of the schema, such as whether a property is optional or mandatory, their data types and edge cardinalities. This helps in providing a precise schema, which supports validation processes, and can be described through the PG-Schema [8] and PG-Schema-PC [97] grammar.

Property constraints. Next, we identify mandatory and optional properties for each type from the merging step. A property is characterized as mandatory for a given type T if it appears in every instance of that type, otherwise, it is considered optional. Let $I_T = \{i_1, i_2, \ldots, i_n\}$ be the instances of type T (either nodes or edges), and let p be a property observed in at least one instance. Let $f_T(p) = \frac{|\{i \in I_T | p \in \mathcal{P}_i\}|}{|I_T|}$ denote the frequency of property p in type T. A property p is considered *mandatory* for a type T, if

Algorithm 2: Extracting and Merging Types

Input: Clusters $C = \{C_1, C_2, \ldots, C_m\}$ with labels $\lambda(C)$ and properties $\mathcal{P}_C$, Jaccard similarity threshold $\theta \in [0, 1]$
Output: Refined set of types T

$T \leftarrow \emptyset$;
for each $C_i \in C$ **do**
  if $\lambda(C_i) \neq \emptyset$ **then**
    if $\exists\, T_j \in T : \lambda(T_j) = \lambda(C_i)$ **then**
      mergeClusters(C_i, T_j) ;
    else
      add C_i to T;
for each $C_u \in C$ *with* $\lambda(C_u) = \emptyset$ **do**
  $Cands \leftarrow \{C_l \in T \mid \lambda(C_l) \neq \emptyset \wedge \text{Jaccard}(\mathcal{K}_{C_u}, \mathcal{K}_{C_l}) \geq \theta\}$;
  if $Cands \neq \emptyset$ **then**
    mergeClusters$(C_u, Cands)$;
for each *pair* (C_u, C_v) *of unlabeled clusters* **do**
  if Jaccard$(\mathcal{K}_{C_u}, \mathcal{K}_{C_v}) \geq \theta$ **then**
    mergeClusters(C_u, C_v) ;
return T

$f_T(p) = 1$, else, p is *optional*, assigning the property constraint to $\pi_s(t, p)$.

Example 6. Properties `name`, `gender`, and `bday` are mandatory for the type `Person` (Figure 1), since all `Person` instances have them. In contrast, some `Post` instances have the `imgFile` property, while others don't, so we consider it as an optional property for the type `Post`.

Property data types. To enrich the schema with property types, we analyze the data of the property p with values $v_1, v_2, \ldots, v_k$ from each type T. A priority-based inference is applied by checking the data type of each value v. First, for numeric types, if $v \in \mathbb{Z}$, then we consider p to be an integer, if $v \in \mathbb{R} \setminus \mathbb{Z}$, then p is a float, if $v \in \{\texttt{true}, \texttt{false}\}$ then p is Boolean, and finally, using regex for date/time ISO formats, defaulting to a string, assigning the property constraint to $\pi_s(t, p)$. As these heuristics are expensive performance-wise, optionally we add a flag to infer this information, by sampling a small amount of data (e.g., 10% of the properties, and at least 1000), which notably has minor differences § 5, as we fallback to a string default. We leave for future work the identification of more detailed datatypes, such as enumerated types or bounded ranges.

Example 7. For the `Person` type in Figure 1, the properties `name` and `gender` are assigned as strings, and `bday` is inferred as a date based on its format (e.g., `19/12/1999`).

Cardinalities. To find the cardinalities C of edge types, we query how many distinct targets each source node connects to, for each edge type, and vice versa. Specifically, let $\mathcal{E}$ be the set of edges in the graph, and let $\text{src}(e)$ and $\text{tgt}(e)$ in V denote the source and target node of an edge $e \in \mathcal{E}$, respectively. For each edge type ρ, we compute the maximum out-degree $\max_{\text{out}}(\rho) = \max_s |\{t \mid (s \to t) \in \mathcal{E},\ \text{type}(s \to t) = \rho\}|$ and the maximum in-degree $\max_{\text{in}}(\rho) = \max_t |\{s \mid (s \to t) \in \mathcal{E},\ \text{type}(s \to t) = \rho\}|$. The pair $(\max_{\text{out}}(\rho), \max_{\text{in}}(\rho))$ is then interpreted as follows: $(1, 1)$ implies a 0:1 relationship, $(>1, 1)$ a N:1, $(1, >1)$ a 0:N, and $(>1, >1)$ an M:N cardinality. We cannot determine whether the source's lower bound is exactly 0 or 1, as we query only the edges. This requires examining if all nodes are connected to the respective edge, which increases computational time; we leave this as future work. This information shows the constraints in the graph, which can be leveraged in tasks such as data validation, consistency enforcement, or query optimization.

Example 8. The `WORKS_AT` edge (Figure 1) connects the `Person` nodes to exactly one `Org.`, while an organization may have multiple employees. Thus, the inferred cardinality is N:1. Similarly, the `KNOWS` relationship connects people to other people, so the inferred cardinality is M:N.

4.5 Schema Serialization

From the previous steps, we assemble a property graph schema $S_G = (V_s, E_s, \rho_s)$ following Def. 3.4. This schema expresses the complete structure of the data. For interoperability, we export the schema in XSD [87] and PG-Schema [8], enabling easy integration into external tools [90]. Even though PG-schema has not yet been expressed in a standard structured language [3, 24], we generate both `LOOSE` and `STRICT` graph schema declarations for demonstration[2]. The `LOOSE` schema type can be used for flexible data insertions, allowing nodes and edges to deviate, and the `STRICT`, which also describes data types and schema constraints. `STRICT` schema demands a rigorous structure, which can be overwhelming for real datasets, which include noisy, incomplete, and inconsistent data.

4.6 Incremental Step

To make it possible to process large datasets on machines with limited memory, we introduce an *incremental* approach. Instead of recomputing the schema from scratch, PG-HIVE can process new data insertions in small batches. Every new batch stream Gs of nodes and edges is first transformed into vectors and clustered, just like in the initial pipeline. The resulting clusters are then merged with the existing schema (Algorithm 2), ensuring that similar types are combined while avoiding redundant definitions. This merging step extends the schema with new patterns when necessary, while maintaining consistency with previously discovered types.

Schema merging. Let $S_1 = (V_{s1}, E_{s1}, \rho_{s1})$ and $S_2 = (V_{s2}, E_{s2}, \rho_{s2})$ be two property graph schemas. The schema merging produces a new schema $S_{\text{merged}} = (V_m, E_m, \rho_m)$, such that: $\forall G_1 \in G(S_1) : G_1 \in G(S_{\text{merged}})$, $\forall G_2 \in G(S_2) : G_2 \in G(S_{\text{merged}})$, where $G(S)$ denotes the set of graphs conforming to schema S. S_{merged} is the least general schema that can be valid, so it maintains meaningful type definitions and structural coherence, while general enough to accommodate both S_1 and S_2.

Merge rules. Similarly with the Algorithm 2 and § 4.3, we describe the merge rules as follows:

- **Node types.** V_m is constructed by unifying V_{s1} and V_{s2}. Labeled types with the same label(s) are merged. Unlabeled types are merged firstly with labeled, then with unlabeled types if they have similar structure, otherwise, they are included as new `ABSTRACT` types.
- **Edge types.** E_m is created by merging edge types with the same labels and updating their connectivity function $\rho_m(e)$.
- **Properties.** π_m is defined as the union of properties from π_{s1}, π_{s2}.

Thus, S_{merged} generalizes both S_1 and S_2, avoids excessive generalization, and reflects the least general schema that ensures compatibility.

Monotonicity. The correctness of the merging step in the incremental approach is based on Lemma 1 and Lemma 2 which

[2] LOOSE, STRICT SCHEMA

guarantees that when two types are merged, their labels, properties and relations are preserved due to their union. Therefore, no information is ever discarded on the schema, while every label and property present in S_1 or S_2 is also present in S_{merged}.

Formally, let S_i be the schema after processing batch i, and S_{i+1} the schema after batch i+1. Then $S_i \sqsubseteq S_{i+1}$, meaning that S_{i+1} is a generalization of S_i, extending it with new labels and properties, but without removing existing ones. Thus, the sequence $(S_1, S_2, \ldots, S_n)$ forms a monotone chain of schemas, where each preserves and extends the previous one.

By updating the schema incrementally, PG-HIVE adapts as the graph grows. Newly observed structures are immediately incorporated and constraints can be refined on demand. This avoids expensive full recomputation while keeping the schema accurate and responsive to evolving data. In practice, this makes the method suitable for dynamic environments where updates are frequent. Handling updates and deletions is left for future work, but the current incremental strategy already reduces computational cost significantly while ensuring that the schema remains up to date.

4.7 Theoretical Guarantees

Following, we discuss the guarantees of the inferred schema.

Efficiency. PG-HIVE remains scalable as it avoids pairwise comparisons by using the LSH. The comparisons would need a computational time of $O(N^2D)$. LSH reduces this to $O(NTD)$ (ELSH) or $O(NT)$ (MinHash), where N is the number of elements, $T \ll N$ is the number of tables, and D the dimension of the embedding.

Type completeness. Let $G = (V, E)$ be the input property graph and $S_G = (V_s, E_s)$ the schema inferred by PG-HIVE. For every node $v \in V$ with label ℓ and set of properties $\mathcal{P}_v$, PG-HIVE guarantees that there exists a type $t \in V_s$ such that $\ell \in \lambda_s(t)$ and $\mathcal{P}_v \subseteq \pi_s(t)$. This means that no label or property of the graph is lost. This is ensured since we take the union of each cluster, rather than the intersection, so information is preserved.

Property constraints. PG-HIVE distinguishes mandatory and optional properties in a sound way. For a type T with instances I_T, a property p is mandatory only if it appears in *all* instances of T (§ 4.4), otherwise, it is optional. Hence, every property marked as mandatory is indeed present in every instance of the type.

Data type inference. For each property, we examine its observed values and assign the most specific compatible type following a simple hierarchy: `integer`, `float`, `date/time` and `string`. As a result, all values of a property are consistent with the inferred type, even though the type may be a generalization as string.

Cardinalities. For every edge type, PG-HIVE computes the maximum in- and out-degree observed in the data ($\max_{\text{out}}(\rho)$, $\max_{\text{in}}(\rho)$). These values serve as upper bounds to the respected cardinalities. For example, if a relationship is inferred as $(0, N)$, then, for each source node attached to this edge has more than one target.

Schema merging. When merging two schemas S_1 and S_2, the resulting schema S_{merged} is guaranteed to be general enough to cover both schemas (§ 4.6). Labeled clusters with the same label are merged, unlabeled clusters are merged only when structurally similar, and otherwise kept as abstract types. This ensures that the merged schema is a general schema covering both inputs.

Incrementality. When data arrives in batches, PG-HIVE extends the schema without invalidating what has already been discovered. Formally, if S_i is the schema after batch i, and S_{i+1} after the next batch, then $S_i \sqsubseteq S_{i+1}$ § 4.6, meaning that S_{i+1} generalizes S_i. So, no previously valid instance is excluded, and the schema evolves monotonically as the graph grows.

In short, PG-HIVE guarantees that (i) all labels and properties observed in the data are preserved, (ii) mandatory and optional properties are classified correctly, (iii) inferred datatypes are always compatible with observed values; if fully computed, (iv) edge cardinalities reflect sound upper bounds, and (v) schemas evolve monotonically when updated incrementally. Together, these guarantees ensure that PG-HIVE produces consistent and faithful schemas without losing information from the underlying graph.

Time complexity. *Static module.* We examine the three main components: preprocessing, clustering and type extraction. Preprocessing transforms each element into a vector representation using binary property indicators and fixed-dimensional Word2Vec embeddings with the time complexity $O(N \cdot (P + D))$, where N is the number of data elements (nodes or edges), P the number of properties and D the embedding size. In LSH, each vector is projected into T_n number hash tables, and computing each projection requires a small cost of $O(D)$. Thus the clustering requires $O(N \cdot T_n \cdot D)$ time. Finally, the worst-case scenario of the merging step where the LSH clusters lack any labels, requires computing the Jaccard similarity between all pairs of clusters, has a time complexity of $O(C_n^2)$, where C_n is the number of clusters produced. Combining these steps the total complexity of PG-HIVE is: $O(N \cdot (P + T_n \cdot D)) + O(C_n^2)$. In practice, P, T_n and D are constants, and typically much smaller than N, so the complexity is dominated by $O(N)$ and the cost of the merging step $O(C_n^2)$, resulting in a time complexity of $O(N + C_n^2)$.

Incremental module. In the incremental approach, each new batch of B data elements (nodes or edges) passes through the same three steps: preprocessing, clustering, and merging. Similarly, the preprocessing requires $O(B \cdot (P + D))$, and the clustering $O(B \cdot T_n \cdot D)$. In the merging step, let C_b denote the number of clusters in the current batch. These clusters are compared against the existing C_n clusters of the schema, which results in a cost of $O(C_b \cdot C_n)$. So the total complexity per batch is $O(B \cdot (P + T_n \cdot D)) + O(C_b \cdot C_n)$. Again, P, T_n and D are constants, and much smaller than N, so the final complexity is $O(B + (C_b \cdot C_n))$.

Compared to the static case, the incremental design reduces the workload in smaller updates. Since $B \ll N$ and $C_b \ll C_n$, the incremental approach scales due to the batch size, reducing the overhead while ensuring that the schema remains up to date.

5 Evaluation

This section presents our evaluation methodology and results.

Setup. All experiments were conducted on a 4-node Spark Standalone cluster (Ubuntu 20.04.2 LTS, 38 cores/machine, SSDs, Gigabit Ethernet), using Spark 3.4.1, Scala 2.12.10, and Neo4j 4.4.0. All configurations and code are publicly available for reproducibility (§ 6).

Datasets. Dataset characteristics are summarized in Table 2. We used four synthetic and four real datasets for the evaluation (indicated as S and R, respectively). We also include the individual structural patterns (Def. 3.5, 3.6) of the various datasets, to highlight the variations in the structure of instances. All datasets are available in the public resource [91].

POLE [76] is a small benchmark with crime investigation data. HET.IO [45] integrates biomedical entities, such as genes, diseases, and drugs. FIB25 [94] and MB6 [95] model connectome data and correspond to the mushroom body and, respectively,

Table 2: Dataset statistics.

Dataset	Nodes	Edges	Node Types	Edge Types	Node Labels	Edge Labels	Node Pat.	Edge Pat.	Real Synth.
POLE	61,521	105,840	11	16	11	16	17	16	S
MB6	486,267	961,571	4	5	10	3	52	4	S
HET.IO	47,031	2,250,197	11	24	12	24	14	35	R
FIB25	802,473	1,625,428	4	5	10	3	31	4	S
ICIJ	2,016,523	3,339,267	5	14	6	14	208	42	R
CORD19	5,485,296	5,720,776	16	16	16	16	89	16	R
LDBC	3,181,724	12,505,476	7	17	8	15	9	15	S
IYP	44,539,999	251,432,812	86	25	33	25	1210	790	R

medulla neural networks in the fruit fly brain. ICIJ [49] contains information on offshore entities, linked to major leaks such as the Panama Papers. The LDBC [35, 93] simulates a large-scale social network. CORD19 [29] integrates genotype, disease, and bibliographical data. Finally, IYP [37] describes networking and internet measurements, from integrated datasets.

For all datasets, we already have the schema ground truth (node and edge types), yet none of them include property data types and constraints. Some of these datasets are already used for evaluating PG schema discovery [16, 64], or PG-related tasks [19, 23, 38, 66]. However, to the best of our knowledge, this is the first time that all of them are used together in a single benchmark, significantly increasing the total number of datasets compared to previous works.

Noise injection. To evaluate the resilience of the various approaches in harder cases, we introduce noise to the datasets. We injected noise by randomly removing 0%–40% of node/edge properties, testing three label availability scenarios: 100% (all labels retained), 50% (half retained), and 0% (no labels). This approach enables us to examine how different pattern inconsistencies and poor label annotation affect the quality of schema discovery.

Baselines. We compare our approach using ELSH and MinHash [65], against GMMSchema [16] (GMM), and SchemI [64], the only competitors available in the domain.

Evaluation metrics. We evaluate the **quality** of the resulting schema and the **efficiency** of the corresponding algorithms. In all approaches, a discovered type is represented as a cluster accompanied by a set of properties. Each discovered cluster may contain data of multiple types, and the label(s) associated with a cluster are not known beforehand — they are inferred from the composition of the cluster itself after the clustering process. For evaluation purposes, we assign the label(s) to each cluster, namely the most frequent (majority) actual label(s) among the nodes or edges it contains (e.g., if most elements have label `Person` or `{Person & Student}`). To evaluate the quality of the generated clusters, we use the **majority-based F1-Score (F1*-Score)**, where the correctness of a node/edge placement is determined based on whether its actual type matches the majority label(s) of its cluster. This approach has been adopted in previous works to assess clustering performance [69]. This reflects how well the discovered clusters assign the types to the same groups, as well as how we represent the schema types, especially in noisy and semi-labeled datasets.

Beyond clustering quality, we also focus on evaluating the correctness of inferred data type attributes. We compare our sample-based inference against a full scan of the dataset. Formally, for each property p, let D_p denote the set of all values of p (by scanning the full dataset), and $S_p \subseteq D_p$ the sampled values. Let $f(p)$ be the datatype inference function. We define **the sampling error** for p as, $\text{error}(p) = \frac{1}{|S_p|} \sum_{v \in S_p} \mathbf{1}\big(f(v) \neq f(D_p)\big)$, where $f(D_p)$ is the inferred datatype when considering all values of p, and $\mathbf{1}(\cdot)$ equals 1 if the sample-based inference disagrees with the full-scan inference, 0 otherwise.

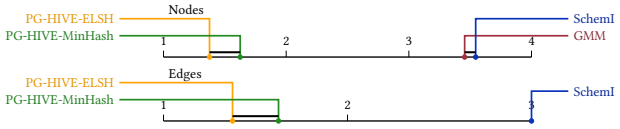

Figure 3: Statistical significance analysis of F1-scores across datasets for nodes (top) and edges (bottom) –GMM does not produce edge types.

5.1 Results

Statistical significance. First, we focus on the effectiveness and present the F1*-scores across all 40 test cases (8 datasets x 5 noise levels) under 100% label availability. *Note that GMM and SchemI are able to work only under fully labeled datasets (100% label availability).* Using the Nemenyi test [44, 75] we show in Figure 3 which pairs of algorithms differ significantly. Each point of Figure 3 indicates the average rank of the method over all cases of noise, with lower ranks indicating better average performance. As shown, PG-HIVE-ELSH and PG-HIVE-MinHash form a group with no major difference between them, while both significantly outperform GMM and SchemI for nodes, and SchemI for edges –GMM cannot discover edge types. This confirms that PG-HIVE is statistically better across all test cases.

Quality of the discovered schema as noise increases. Then we focus in detail on the F1*-scores across different levels of noise. Figure 4 shows the F1*-scores as noise increases (0-40%), under three label availability scenarios (100%, 50%, 0%).

Overall, across all datasets and test cases, PG-HIVE maintains high accuracy with scores above 0.9 even in high levels of noise and no label availability. This happens due to PG-HIVE's hybrid approach, leveraging LSH to approximate structural similarities effectively, which remains robust as noise misleads the clustering. Additionally, the merging step further refines clusters, resulting in accurate results. In the corner case where no noise is ingested (100% label availability and 0% noise), PG-HIVE and GMMSchema both accurately discover the correct number of node types (F1*-score approx. 1.0) across datasets, whereas they are both superior to SchemI. However, as noise increases, PG-HIVE's approximation and adaptive merging retain accurate inference and correctly group the corresponding elements by focusing on structural patterns.

In contrast, GMMSchema and SchemI are sensitive to noise and missing labeling information. For the node type inference, in 100% label availability, GMMSchema starts with a high F1*-score (1.0), as the Gaussian Mixture Model clustering performs well on clean data, but as noise exceeds 20%, the variety in property distributions causes misclustering, dropping the F1*-score below 0.6. SchemI consistently has a lower F1*-score (0.6-0.8) than PG-HIVE. For the edge types, only SchemI is able to detect them with a significantly inferior performance than PG-HIVE.

With incomplete labeling information, only PG-HIVE is able to produce results, and as such, we don't see the SchemI and GMMSchema lines for 50% and 0% labeling information. Although type inference becomes straightforward when label availability is 100%, with 50% label availability, slight misgroupings may occur due to similar structures in different types. In the 0% label availability case, inference becomes more challenging, as types with identical structures are merged based on structural similarity

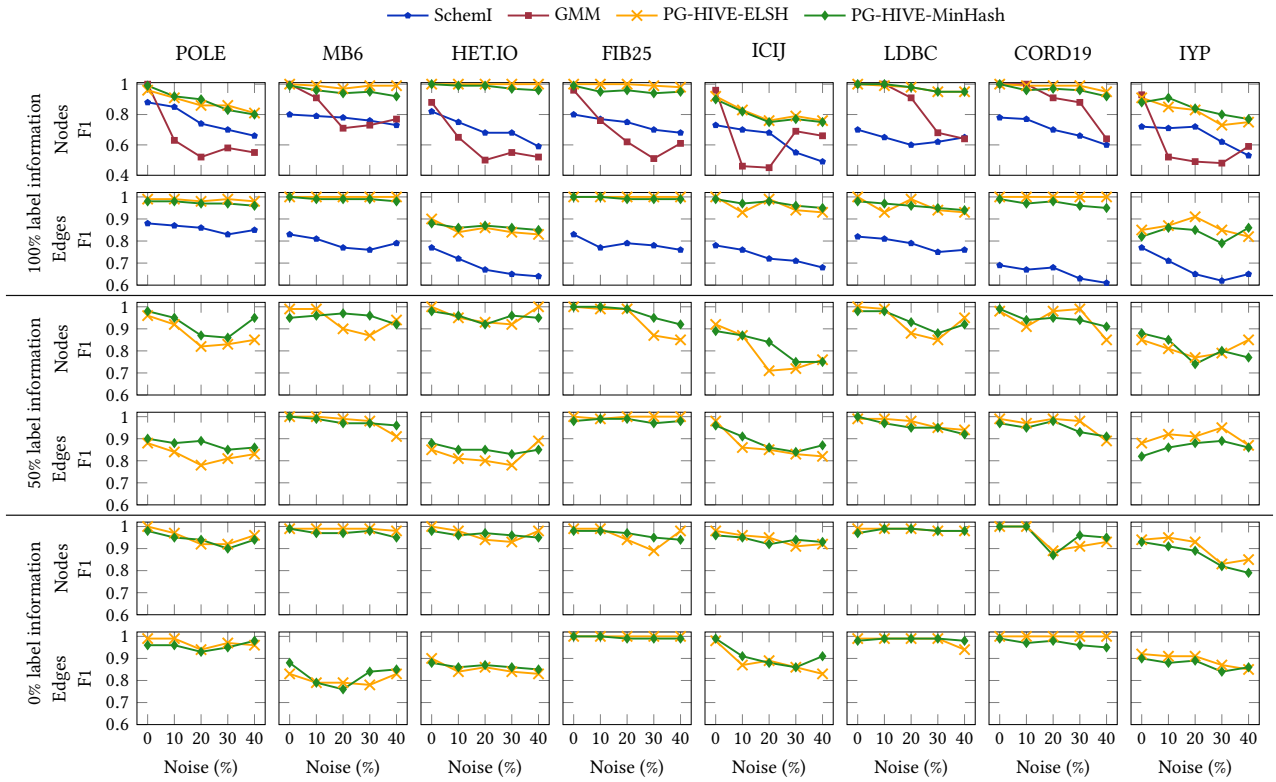

Figure 4: F1 scores across all noise levels (0-40%) and label availability (0-50-100%).

using both LSH and the second merging step, potentially reducing precision but still enabling robust discovery with PG-HIVE.

Comparing the two PG-HIVE variations, there is no significant difference (see Figure 3), in terms of F1*-scores. They both maintain high accuracy (above 0.9) under different noise levels and label availability, as their LSH-based clustering and second-step merging are efficient enough to capture structural similarities. The slight differences, if any, occur due to the adaptive parameterization § 4.2, which might overlook some outliers; however this is tackled with our adaptive parameterization.

Comparing node and edge types, inference of node types can be more challenging. Nodes may appear unlabeled in the datasets, along with different structural variations. While edges usually have fewer structural variations, extracting their types relies on their labeling information and the concatenation of the different (if such) source and target labels. Additionally, the corresponding internal LSH representation and the merging step give an advantage to edge discovery. Nodes are represented as a Word2Vec along with the one-hot encoded binary vector, while edges have 3 Word2Vec and a binary vector. This increased expressivity of the Word2Vec allows a better separation in edge clustering, when labels are not identical, resulting in higher F1* scores (above 0.9) even under noise. And finally, the second merging step groups together edges with the same labels, if separated before due to structural differences.

PG-HIVE across datasets. Each dataset has diverse properties, as well as multi-labeled nodes, which increase the perplexity of type inference. Simpler or homogeneous datasets, such as POLE, MB6, FIB25, and HET.IO have a flat structure and are easier to infer the schema, even in the absence of label annotation. In contrast, ICIJ and IYP, which are integrated datasets and structurally heterogeneous, pose additional challenges due to structural variability. In these cases, PG-HIVE slightly declines as noise grows, while baselines degrade greatly. LDBC and CORD19, while larger in size, have fewer structural differences, and so remain simple to discover their types. Additionally, MB6 and FIB25 obtain nodes annotated with multiple labels, complicating type inference due to overlapping semantic roles, which can lead to misgroupings if we rely only on the structural similarity, under high levels of noise. Similarly, LDBC, HET.IO, ICIJ and IYP include one extra label that may add semantic information (e.g., HET.IO has assigned to all its nodes an extra `HetionetNode` label, for integration scenarios). These multi-labeling scenarios require careful consideration of label and structural information in PG-HIVE's clustering, particularly in heterogeneous datasets where baseline methods struggle to adapt.

Efficiency. Figure 5 shows execution times across all datasets and noise levels. The execution time includes the preprocessing, clustering, and type extraction. We can observe that PG-HIVE (both ELSH and MinHash variants) has better execution times, by up to 1.95x on average than SchemI in all cases. PG-HIVE still maintains a comparable efficiency to the GMMSchema method, which only retrieves node types. PG-HIVE is able to process small and less complicated datasets (POLE, MB6, HET.IO, FIB25) in less than half a minute, bigger and more versatile datasets (LDBC, ICIJ, CORD19), in 3-7 minutes, while more complicated datasets (IYP) need 15 minutes.

We can notice that, in our approach, noise does not affect the computational time, also justified by § 4.7. However, this is not the case for GMMSchema. As the noise increases, the number

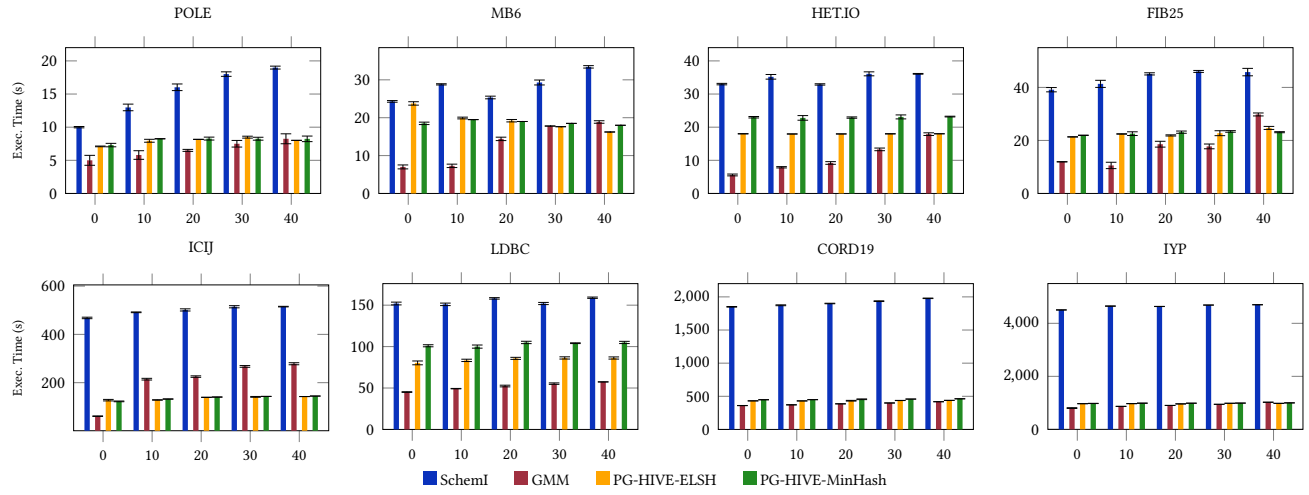

Figure 5: Execution time until type discovery on each dataset across different noise percentages (0% - 40%).

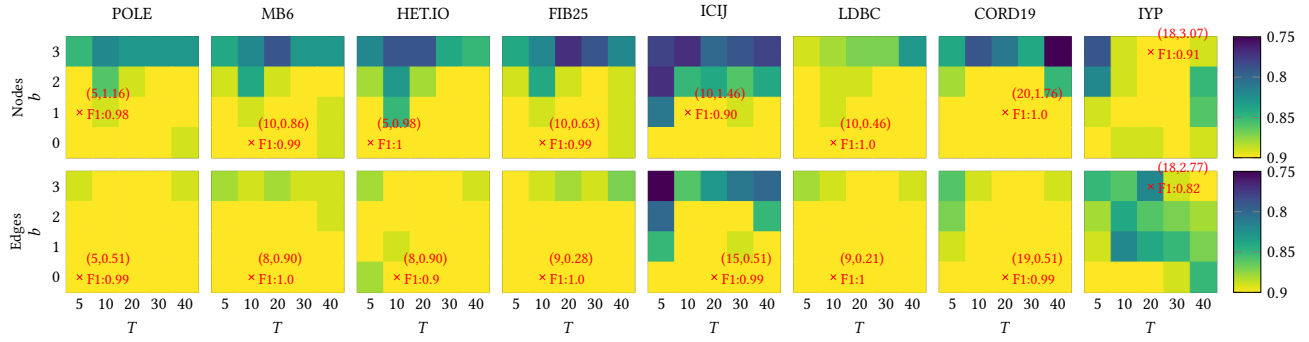

Figure 6: Heatmaps of F1-scores across datasets (100% label availability & 0% noise) for nodes (top) and edges (bottom) with varying T and a; red × (T, a) denotes the adaptive choice for ELSH.

of clusters grows, resulting in higher computational cost, and in most cases, at 40% noise, PG-HIVE is more efficient, despite the fact that GMMSchema only discovers node types. PG-HIVE's efficiency can, in addition, be theoretically justified by LSH's approximate hashing, which scales linearly with data size (O(N)), unlike GMM's quadratic complexity in noisy scenarios, making PG-HIVE suitable for large and unstructured graphs.

Adaptive parameters. In Figure 6, we ran experiments in the ELSH approach with different parameters (T, a), comparing against our adaptive approach (marked in red). The results of the F1*-score indicate that in most datasets, the adaptive approach is very close to the best-performing setting (yellow indicates higher F1*, thus blue lower). This happens due to the upper bound in the heuristic-based estimation, which narrows down the search space. When performance drops, it is due to the second merging step (§ 4.3), not parameter miscalculation. However, as this relies on approximate heuristics, there might be cases where we cannot find the optimal parameter, but still we can identify one with high accuracy. IYP is such a case; however, although we could not identify the optimal parameters, we still retained high accuracy.

In general, lowering b values over-separates patterns; thus, we have a higher F1*-score, while increasing b and T values, merges distinct patterns, lowering F1*-score. This is also in line with our theoretical expectations (§ 4.2).

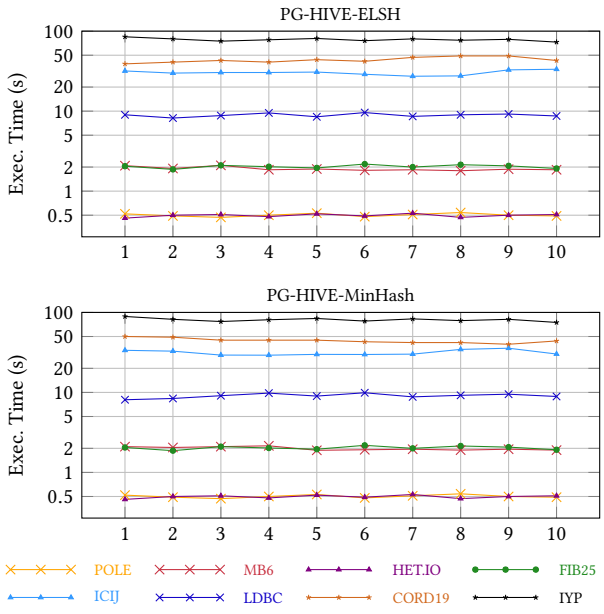

Figure 7: Incremental execution time per iteration.

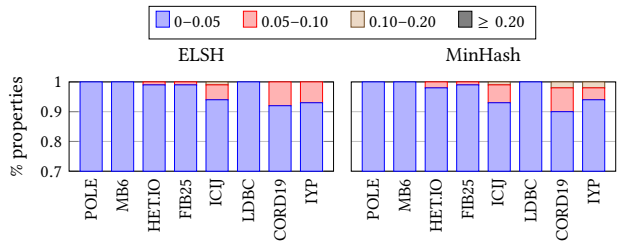

Figure 8: Distribution of data type inference errors using sampling, across datasets for ELSH (left) and MinHash (right).

Incremental module. To evaluate the incremental approach, we randomly separate the graph into 10 batches and run both of our approaches. In Figure 7, we show the running time in seconds that each approach needs to process each batch of size B. The incremental approach suits to streaming data, by avoiding full recomputation. The consistent running times justify the incremental design's efficiency, as it processes only new data and merges with the existing schema ($O(B + C_b * C_n)$), with $C_b, C_n \ll B$.

Schema constraints. The inference of optional/mandatory attributes and cardinalities has previously been explained in § 4.7. In this direction, we focus next on evaluating the quality of the identified datatypes. In our approach, we infer the datatypes based on sampling (§ 4.4). As such, we compare our approach with the dominant types determined using a full scan, in terms of sampling error. The results are shown in Figure 8, grouped into bins (0–0.05, 0.05–0.10, 0.10–0.20, and ≥0.20), and normalized by the number of properties per dataset.

As shown, most properties fall into the lowest error range, demonstrating consistent datatype inference across datasets for both clustering approaches. The few outliers occur on bigger datasets with heterogeneous properties like ICIJ, CORD19, and IYP, where the small sample does not reflect all the properties available in the data. Misinterpretations of data are usually of the following types: assigning `DOUBLE` rather than `INTEGER`, or assigning `DATE`, but the full scan might observe outliers and assign the property as `STRING`. Nevertheless, this problem can be easily mitigated by increasing the sampling percentage and is dataset dependent. We leave this exploration for future work.

Summary. Our extensive evaluation on both real and synthetic datasets demonstrates that PG-HIVE consistently outperforms existing schema discovery approaches, even under high levels of noise and in the absence of label information. In terms of quality, PG-HIVE maintains high accuracy, even under challenging scenarios with 0% label information, where GMMSchema and SchemI do not work at all, and 40% noise, where GMMSchema and SchemI underperform. Specifically, PG-HIVE achieves up to 65% higher F1*-score for node types and up to 40% for edge type discovery compared to existing approaches. The statistical significance analysis (Figure 3) confirms that both PG-HIVE approaches (ELSH and MinHash) are superior to prior methods across all datasets.

In terms of efficiency, PG-HIVE has slightly slower execution times than GMMSchema, which can be justified as GMMSchema only infers node types, while PG-HIVE discovers both node and edge types, as well as constraings. Nevertheless, PG-HIVE remains efficient, achieving on average 1.95x faster execution compared to SchemI. Moreover, our incremental approach demonstrates an efficient running time on each dataset. Our adaptive parameter selection strategy leverages heuristics (§ 4.2), which limit the exploration space while achieving near-optimal parameter selection. Finally, datatype inference shows a small sampling error, demonstrating that the sampling-based method can be both reliable and efficient even for large datasets.

Overall, PG-HIVE provides a balance between accuracy and efficiency, outperforming the current state-of-the-art methods. This comprehensive evaluation justifies PG-HIVE's advancements, as it addresses real-world cases like multi-labeled and heterogeneous datasets, confirming its importance for property graph management.

6 Conclusion and Future Work

In this paper, we present PG-HIVE, a hybrid incremental schema discovery approach for property graphs. Our approach discovers node types, edge types, and infers schema characteristics, without any prior schema information and with high levels of noise. In addition, it can process large datasets in smaller batches incrementally. Unlike previous methods, it does not rely only on predefined labels; instead, it groups nodes and edges based on both structure and semantics. Our experiments show that it outperforms baseline approaches and infers the schema successfully even with semi-annotated data.

As future work, we intend to investigate more challenging scenarios that further complicate schema discovery. Specifically, we aim to: a) handle cases where no label information is available and data is extremely sparse, b) detect types that share identical type patterns but lack distinguishing labels, and c) support integration scenarios when label semantics are not consistent (e.g., labels in different languages). To address these challenges, we plan to enhance PG-HIVE to handle the label variations by integrating large language models (LLMs), to semantically align labels across datasets, without relying on exact string matches. Additionally, for enumerations and value semantics, we should leverage the property values and their semantic interpretation, along with additional schema constraints.

Acknowledgments

The work reported in this paper was partially supported by the TALE H.F.R.I. Project (Project Number: 16819), by DataGEMS (Grant Agreement 101188416) and MULTIPULM (Grant Agreement No 101226783) EU projects funded by the European Union's Horizon Europe Research and Innovation programme, and by DEDALUS project (TA 5180519) funded by SUB1.1 Clusters of Research Excellence (CREs), Greece.

Artifacts

All configurations and code are publicly available for reproducibility here: https://github.com/sophisid/PG-HIVE. The datasets used and their sources are listed as: POLE [76] Github repo, MB6 [51, 95] CSV Dataset, HET.IO [45] Github repo, FIB25 [51, 94] CSV Dataset, ICIJ [49] Github repo, LDBC [35, 93] CSV Dataset, CORD-19 [29] Github repo, IYP [37] Github repo. We have gathered all datasets in a unified public resource [91].

References

[1] [n. d.]. The Neo4j Graph Platform. https://neo4j.com/ Accessed: 2025-04-17.

[2] Marco D Adelfio and Hanan Samet. 2013. Schema extraction for tabular data on the web. *Proceedings of the VLDB Endowment* 6, 6 (2013), 421–432.

[3] Shqiponja Ahmetaj, Iovka Boneva, Jan Hidders, Katja Hose, Maxime Jakubowski, Jose Emilio Labra Gayo, Wim Martens, Fabio Mogavero, Filip Murlak, Cem Okulmus, et al. 2025. Common foundations for SHACL, ShEx, and PG-Schema. In *Proceedings of the ACM on Web Conference 2025*. 8–21.

[4] Rana Alotaibi, Chuan Lei, Abdul Quamar, Vasilis Efthymiou, and Fatma Özcan. 2021. Property Graph Schema Optimization for Domain-Specific Knowledge Graphs. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 924–935. doi:10.1109/ICDE51399.2021.00085

[5] Güneş Aluç, M Tamer Özsu, and Khuzaima Daudjee. 2019. Building self-clustering RDF databases using Tunable-LSH. *The VLDB Journal* 28, 2 (2019), 173–195.

[6] Renzo Angles. 2018. The Property Graph Database Model.. In *AMW*.

[7] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5 (2017), 68:1–68:40. doi:10.1145/3104031

[8] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. 2023. PG-Schema: Schemas for Property Graphs. *Proc. ACM Manag. Data* 1, 2 (2023), 198:1–198:25. doi:10.1145/3589778

[9] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savkovic, Michael Schmidt, Juan F. Sequeda, Slawek Staworko, and Dominik Tomaszuk. 2021. PG-Keys: Keys for Property Graphs. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2423–2436. doi:10.1145/3448016.3457561

[10] Grigoris Antoniou and Frank Van Harmelen. 2004. *A semantic web primer*. MIT press.

[11] Sydney Anuyah, Victor Bolade, and Oluwatosin Agbaakin. 2024. Understanding Graph Databases: A Comprehensive Tutorial and Survey. *CoRR* abs/2411.09999 (2024). doi:10.48550/ARXIV.2411.09999 arXiv:2411.09999

[12] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomasz Kaftan, Michael J Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1383–1394. doi:10.1145/2723372.2742797

[13] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[14] Alex Bogatu, Alvaro AA Fernandes, Norman W Paton, and Nikolaos Konstantinou. 2020. Dataset discovery in data lakes. In *2020 ieee 36th international conference on data engineering (icde)*. IEEE, 709–720.

[15] Angela Bonifati, Stefania-Gabriela Dumbrava, Emile Martinez, Fatemeh Ghasemi, Malo Jaffré, Pacome Luton, and Thomas Pickles. 2022. DiscoPG: Property Graph Schema Discovery and Exploration. *Proc. VLDB Endow.* 15, 12 (2022), 3654–3657. doi:10.14778/3554821.3554867

[16] Angela Bonifati, Stefania Dumbrava, and Nicolas Mir. 2022. Hierarchical Clustering for Property Graph Schema Discovery. In *EDBT*. 2:449–2:453. doi:10.48786/EDBT.2022.39

[17] Angela Bonifati, George HL Fletcher, Hannes Voigt, Nikolay Yakovets, and HV Jagadish. 2018. Querying Graphs. Morgan & Claypool Publishers. *Synthesis Lectures on Data Management* (2018).

[18] Angela Bonifati, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko, and Hannes Voigt. 2019. Schema validation and evolution for graph databases. In *International Conference on Conceptual Modeling*. Springer, 448–456.

[19] Angela Bonifati, Yann Ramusat, Filip Murlak, Amela Fejza, and Rachid Echahed. 2024. DTGraph: declarative transformations of property graphs. In *VLDB 2024-50th International Conference on Very Large Databases*.

[20] Redouane Bouhamoum, Zoubida Kedad, and Stéphane Lopes. 2018. Schema Discovery in Large Web Data Sources. In *Proceedings of the 1st International Conference on Big Data and Cyber-Security Intelligence, BDCSIntell 2018, Hadath, Lebanon, December 13-15, 2018 (CEUR Workshop Proceedings, Vol. 2343)*, Marie-Rita Hojeij, Béatrice Finance, Yehia Taher, Karine Zeitouni, Rafiqul Haque, and Mohamed Dbouk (Eds.). CEUR-WS.org, 67–74. https://ceur-ws.org/Vol-2343/paper14.pdf

[21] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, 21–29.

[22] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 1998. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 327–336.

[23] Francesco Cazzaro, Justin Kleindienst, Sofia Marquez Gomez, and Ariadna Quattoni. 2025. ZOGRASCOPE: A New Benchmark for Semantic Parsing over Property Graphs. *arXiv preprint arXiv:2503.05268* (2025).

[24] Kasidis Chantharojwong, Sourav S Bhowmick, and Byron Choi. 2025. LICS: Towards Theory-Informed Effective Visual Abstraction of Property Graph Schemas. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–26.

[25] Zoé Chevallier, Zoubida Kedad, Béatrice Finance, and Frédéric Chaillan. 2024. Data Search and Discovery in RDF Sources. In *The Semantic Web: ESWC 2024 Satellite Events - Hersonissos, Crete, Greece, May 26-30, 2024, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 15344)*, Albert Meroño-Peñuela, Óscar Corcho, Paul Groth, Elena Simperl, Valentina Tamma, Andrea Giovanni Nuzzolese, María Poveda-Villalón, Marta Sabou, Valentina Presutti, Irene Celino, Artem Revenko, Joe Raad, Bruno Sartini, and Pasquale Lisena (Eds.). Springer, 188–192. doi:10.1007/978-3-031-78952-6_24

[26] Peter Christen, Thilina Ranbaduge, and Rainer Schnell. 2020. Building Blocks for Linking Sensitive Data. In *Linking Sensitive Data: Methods and Techniques for Practical Privacy-Preserving Information Sharing*. Springer, 123–167.

[27] Klitos Christodoulou, Norman W. Paton, and Alvaro A. A. Fernandes. 2013. Structure inference for linked data sources using clustering. In *Joint 2013 EDBT/ICDT Conferences, EDBT/ICDT '13, Genoa, Italy, March 22, 2013, Workshop Proceedings*, Giovanna Guerrini (Ed.). ACM, 60–67. doi:10.1145/2457317.2457328

[28] CIDOC CRM Special Interest Group. [n. d.]. CIDOC Conceptual Reference Model (CIDOC CRM). http://www.cidoc-crm.org/. Accessed: 2025-09-30.

[29] CovidGraph. 2021. COVID-19 Knowledge Graph. https://covidgraph.org/.

[30] Michele Dallachiesa, Charu C Aggarwal, and Themis Palpanas. 2019. Improving Classification Quality in Uncertain Graphs. *Journal of Data and Information Quality (JDIQ)* 11, 1 (2019), 1–20.

[31] Calvin Dani, Shiva Jahangiri, and Thomas Hütter. 2024. Introducing Schema Inference as a Scalable SQL Function [Extended Version]. *arXiv preprint arXiv:2411.13278* (2024).

[32] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 253–262.

[33] Michael DiScala and Daniel J Abadi. 2016. Automatic generation of normalized relational schemas from nested key-value data. In *Proceedings of the 2016 International Conference on Management of Data*. 295–310.

[34] Stefania Dumbrava, Angela Bonifati, Amaia Nazabal Ruiz Diaz, and Romain Vuillemot. 2019. Approximate querying on property graphs. In *International Conference on Scalable Uncertainty Management*. Springer, 250–265.

[35] Orri Erling, Alex Averbuch, Josep Lluís Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat-Pérez, Minh-Duc Pham, and Peter A. Boncz. 2015. The LDBC Social Network Benchmark: Interactive Workload. In *SIGMOD*. 619–630. doi:10.1145/2723372.2742786

[36] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2001. Approximate nearest neighbor searching in multimedia databases. In *Proceedings 17th International Conference on Data Engineering*. IEEE, 503–511.

[37] Romain Fontugne, Malte Tashiro, Raffaele Sommese, Mattijs Jonker, Zachary S Bischof, and Emile Aben. 2024. The wisdom of the measurement crowd: Building the internet yellow pages a knowledge graph for the internet. In *Proceedings of the 2024 ACM on Internet Measurement Conference*. 183–198.

[38] Dimitrios Giakatos, Malte Tashiro, and Romain Fontugne. 2025. Pythia: Facilitating Access to Internet Data Using LLMs and IYP. In *2025 IEEE 50th Conference on Local Computer Networks (LCN)*. IEEE, 1–7.

[39] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie (Eds.). Morgan Kaufmann, 518–529. http://www.vldb.org/conf/1999/P49.pdf

[40] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[41] GraphQL Foundation. 2021. GraphQL Specification. https://spec.graphql.org/October2021/. Accessed: 2025-09-25.

[42] Olaf Hartig. 2014. Reconciliation of RDF* and Property Graphs. *CoRR* abs/1409.3288 (2014).

[43] Olaf Hartig and Jan Hidders. 2019. Defining Schemas for Property Graphs by using the GraphQL Schema Definition Language. In *GRADES/NDA*. 6:1–6:11. doi:10.1145/3327964.3328495

[44] Steffen Herbold. 2020. Autorank: A python package for automated ranking of classifiers. *Journal of Open Source Software* 5, 48 (2020), 2173.

[45] Daniel S. Himmelstein, Antoine Lizee, Connor Hessler, Lars Brueggeman, Sabrina L. Chen, Dexter Hadley, Ari Green, Pouya Khankhanian, Sergio E. Baranzini, Matteo Pellegrini, and et al. 2017. Systematic integration of biomedical knowledge prioritizes drugs for repurposing. *eLife* 6 (2017), e26726. doi:10.7554/eLife.26726

[46] Michael E Houle and Jun Sakuma. 2005. Fast approximate similarity search in extremely high-dimensional data sets. In *21st International Conference on Data Engineering (ICDE'05)*. IEEE, 619–630.

[47] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

[48] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[49] International Consortium of Investigative Journalists. 2016. ICIJ Offshore Leaks Database (including Bahamas Leaks). https://offshoreleaks.icij.org.

[50] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. 2021. A survey on locality sensitive hashing algorithms and their applications. *arXiv preprint arXiv:2102.08942* (2021).

[51] Janelia Research [n. d.]. NeuPrint Manual. https://www.janelia.org/sites/default/files/Project%20Teams/Fly%20EM/3.1%20191002_neuPrintManual-draft.pdf An introduction to the NeuPrint data model and Cypher Query Language.

[52] Ming Ji, Jiawei Han, and Marina Danilevsky. 2011. Ranking-based classification of heterogeneous information networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1298–1306.

[53] Nikolaos Kardoulakis, Kenza Kellou-Menouer, Georgia Troullinou, Zoubida Kedad, Dimitris Plexousakis, and Haridimos Kondylakis. 2021. Hint: Hybrid and incremental type discovery for large RDF data sources. In *Proceedings of the 33rd International Conference on Scientific and Statistical Database Management*. 97–108.

[54] Avita Katal, Mohammad Wazid, and Rayan H Goudar. 2013. Big data: issues, challenges, tools and good practices. In *2013 Sixth international conference on contemporary computing (IC3)*. Ieee, 404–409.

[55] Kenza Kellou-Menouer and Zoubida Kedad. 2015. Schema Discovery in RDF Data Sources. In *Conceptual Modeling - 34th International Conference, ER 2015, Stockholm, Sweden, October 19-22, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9381)*, Paul Johannesson, Mong-Li Lee, Stephen W. Liddle, Andreas L. Opdahl, and Oscar Pastor López (Eds.). Springer, 481–495. doi:10.1007/978-3-319-25264-3_36

[56] Kellou-Menouer Kenza, Kardoulakis Nikolaos, Troullinou Georgia, Kedad Zoubida, Plexousakis Dimitris, and Kondylakis Haridimos. 2022. A survey on semantic schema discovery. *The VLDB Journal* 31, 4 (2022), 675–710. doi:10.1007/S00778-021-00717-X

[57] Aamod Khatiwada, Harsha Kokel, Ibrahim Abdelaziz, Subhajit Chaudhury, Julian Dolby, Oktie Hassanzadeh, Zhenhan Huang, Tejaswini Pedapati, Horst Samulowitz, and Kavitha Srinivas. 2025. Tabsketchfm: Sketch-based tabular representation learning for data discovery over data lakes. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 1523–1536.

[58] Haridimos Kondylakis, Stefania Dumbrava, Matteo Lissandrini, Nikolay Yakovets, Angela Bonifati, Vasilis Efthymiou, George Fletcher, Dimitris Plexousakis, Riccardo Tommasini, Georgia Troullinou, et al. 2025. Property Graph Standards: State of the Art & Open Challenges. *Proceedings of the VLDB Endowment* 18, 12 (2025), 5477–5481.

[59] Haridimos Kondylakis, Vasilis Efthymiou, Georgia Troullinou, Elisjana Ymeralli, and Dimitris Plexousakis. 2024. Property Graphs at Scale: A Roadmap and Vision for the Future (Short Paper). In *CAiSE Workshops*, Vol. 521. 180–185.

[60] Haridimos Kondylakis, Giorgos Flouris, and Dimitris Plexousakis. 2009. Ontology and schema evolution in data integration: review and assessment. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 932–947.

[61] Haridimos Kondylakis and Dimitris Plexousakis. 2011. Exelixis: evolving ontology-based data integration system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis (Eds.). ACM, 1283–1286.

[62] Haridimos Kondylakis and Dimitris Plexousakis. 2012. Ontology evolution: assisting query migration. In *International Conference on Conceptual Modeling*. Springer, 331–344.

[63] Haridimos Kondylakis and Dimitris Plexousakis. 2012. Ontology Evolution: Assisting Query Migration. In *Conceptual Modeling - 31st International Conference ER 2012, Florence, Italy, October 15-18, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7532)*, Paolo Atzeni, David W. Cheung, and Sudha Ram (Eds.). Springer, 331–344.

[64] Hanâ Lbath, Angela Bonifati, and Russ Harmer. 2021. Schema Inference for Property Graphs. In *EDBT*. 499–504. doi:10.5441/002/EDBT.2021.58

[65] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. 2020. *Mining of Massive Datasets* (3rd ed.). Cambridge University Press. http://www.mmds.org/ Chapter 3: Finding Similar Items.

[66] Ke Liang, Lingyuan Meng, Meng Liu, Yue Liu, Wenxuan Tu, Siwei Wang, Sihang Zhou, Xinwang Liu, Fuchun Sun, and Kunlun He. 2024. A Survey of Knowledge Graph Reasoning on Graph Types: Static, Dynamic, and Multimodal. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 12 (2024), 9456–9478. doi:10.1109/TPAMI.2024.3417451

[67] Matteo Lissandrini, Davide Mottin, Katja Hose, and Torben Bach Pedersen. 2022. Knowledge graph exploration systems: Are we lost?. In *Annual Conference on Innovative Data Systems Research*.

[68] Matteo Lissandrini, Davide Mottin, Yannis Velegrakis, and Themis Palpanas. 2018. X2Q: your personal example-based graph explorer. *Proceedings of the VLDB Endowment* 11, 12 (2018), 2026–2029.

[69] Hira Mahmood, Tahir Mehmood, and Laila A Al-Essa. 2023. Optimizing clustering algorithms for anti-microbial evaluation data: A majority score-based evaluation of K-means, Gaussian mixture model, and multivariate T-distribution mixtures. *IEEE Access* 11 (2023), 79793–79800.

[70] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781* (2013).

[71] Renée J Miller and Periklis Andritsos. 2003. On Schema Discovery. *IEEE Data Engineering Bulletin* 26, 3 (2003), 39–44.

[72] Michael J Mior. 2023. JSONoid: Monoid-based Enrichment for Configurable and Scalable Data-Driven Schema Discovery. *arXiv preprint arXiv:2307.03113* (2023).

[73] Apache Spark MLlib. 2024. Apache Spark MLlib: LSH Implementation. https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/ml/feature/LSH.scala. Accessed: 2024-11-14.

[74] Thomas Mulder, George Fletcher, and Nikolay Yakovets. 2025. Optimizing navigational graph queries. *The VLDB Journal* 34, 2 (2025), 16.

[75] Peter Bjorn Nemenyi. 1963. *Distribution-free multiple comparisons*. Princeton University.

[76] Neo4j. 2025. Crime Investigation (POLE: Person–Object–Location–Event) dataset. https://github.com/neo4j-graph-examples/pole. Example dataset from the Neo4j Graph Examples repository.

[77] Uchechukwu Fortune Njoku, Alberto Abelló Gamazo, Besim Bilalli, and Gianluca Bontempi. 2024. Finding relevant information in big datasets with ML. In *Proceedings 27th International Conference on Extending Database Technology (EDBT 2024): Paestum, Italy, March 25-March 28*. OpenProceedings, 846–849.

[78] Oracle. 2025. Property Graph Overview. https://docs.oracle.com/en/database/oracle/property-graph/. Accessed: 2025-04-03.

[79] Ayush Pandey, Stefania Dumbrava, Marc Shapiro, Carla Ferreira, Mário Pereira, and Nuno Preguiça. 2025. Towards Local-First Distributed Property Graphs. In *Proceedings of the 12th Workshop on Principles and Practice of Consistency for Distributed Data*. 22–29.

[80] Norman W Paton and Zhenyu Wu. 2024. Dataset Discovery and Exploration: State-of-the-art, Challenges and Opportunities.. In *EDBT*. 854–857.

[81] Axel Polleres, Romana Pernisch, Angela Bonifati, Daniele Dell'Aglio, Daniil Dobriy, Stefania Dumbrava, Lorena Etcheverry, Nicolas Ferranti, Katja Hose, Ernesto Jiménez-Ruiz, et al. 2023. How does knowledge evolve in open knowledge graphs? *Transactions on Graph Data and Knowledge* 1, 1 (2023), 11–1.

[82] Angles Renzo, Arenas Marcelo, Barceló Pablo, Hogan Aidan, Reutter Juan, and Vrgoč Domagoj. 2017. Foundations of modern query languages for graph databases. *ACM Computing Surveys (CSUR)* 50, 5 (2017), 1–40.

[83] Marko A. Rodriguez and Peter Neubauer. 2010. Constructions from Dots and Lines. *CoRR* abs/1006.2361 (2010).

[84] Sameer Sadruddin, Jennifer D'Souza, Eleni Poupaki, Alex Watkins, Hamed Babaei Giglou, Anisa Rula, Bora Karasulu, Sören Auer, Adrie Mackus, and Erwin Kessels. 2025. LLMs4SchemaDiscovery: A Human-in-the-Loop Workflow for Scientific Schema Mining with Large Language Models. *arXiv preprint arXiv:2504.00752* (2025).

[85] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. 2020. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *The VLDB journal* 29 (2020), 595–618.

[86] Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid Aref, Marcelo Arenas, Maciej Besta, Peter A Boncz, et al. 2021. The future is big graphs: a community view on graph processing systems. *Commun. ACM* 64, 9 (2021), 62–71.

[87] Anirban Sarkar. 2012. Conceptual Level Design of Semi-structured Database System: Graph-semantic Based Approach. *CoRR* abs/1202.4532 (2012). arXiv:1202.4532 http://arxiv.org/abs/1202.4532

[88] Chandan Sharma and Roopak Sinha. 2022. FLASc: a formal algebra for labeled property graph schema. *Autom. Softw. Eng.* 29, 1 (2022), 37. doi:10.1007/S10515-022-00336-Y

[89] Joshua Shinavier and Ryan Wisnesky. 2019. Algebraic Property Graphs. *CoRR* abs/1909.04881 (2019). arXiv:1909.04881 http://arxiv.org/abs/1909.04881

[90] Sophia Sideri, Vasilis Efthymiou, Dimitris Plexousakis, and Haridimos Kondylakis. 2024. PG2RDF: Schema-Guided Transformation of Property Graphs to RDF. In *CAiSE Workshops*, Vol. 521. 180–185.

[91] Sophia Sideri, Georgia Troullinou, Elisjana Ymeralli, Vasilis Efthymiou, Dimitris Plexousakis, and Haridimos Kondylakis. 2025. *PG-Schema Bench: A Benchmark for Schema Discovery in Property Graphs*. doi:10.5281/zenodo.17801336

[92] Apache Spark. 2025. MLlib - Locality Sensitive Hashing (LSH). https://spark.apache.org/docs/latest/ml-features.html#locality-sensitive-hashing-lsh. Accessed: 2025-04-28.

[93] Gábor Szárnyas, Jack Waudby, Benjamin A. Steer, Dávid Szakállas, Altan Birler, Mingxi Wu, Yuchen Zhang, and Peter A. Boncz. 2022. The LDBC Social Network Benchmark: Business Intelligence Workload. *Proc. VLDB Endow.* 16, 4 (2022), 877–890. doi:10.14778/3574245.3574270

[94] Shin-ya Takemura and et al. 2015. Synaptic circuits and their variations within different columns in the visual system of Drosophila. *Proceedings of the National Academy of Sciences* 112 (2015), 13711–13716.

[95] Shin-ya Takemura and et al. 2017. A Connectome of a Learning and Memory Center in the Adult Drosophila Brain. *eLife* 6 (2017).

[96] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2010. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Transactions on Database Systems (TODS)* 35, 3 (2010), 1–46.

[97] Dominik Tomaszuk and José Emilio Labra Gayo. 2025. On Property Constraints in PG-Schema. In *Proceedings of the 13th Knowledge Capture Conference 2025*. 69–73.

[98] Hongya Wang, Jiao Cao, LihChyun Shu, and Davood Rafiei. 2013. Locality sensitive hashing revisited: filling the gap between theory and algorithm

analysis. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi (Eds.). ACM, 1969–1978. doi:10.1145/2505515.2505765

[99] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2017. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2017), 769–790.

[100] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Graph-and tree-based indexes for high-dimensional vector similarity search: analyses, comparisons, and future directions. IEEE Data Eng. Bull. 46 (3), 3–21 (2023).

[101] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. 2024. Steiner-hardness: A query hardness measure for graph-based ann indexes. *Proceedings of the VLDB Endowment* 17, 13 (2024), 4668–4682.

[102] Joohyung Yun, Byungchul Tak, and Wook-Shin Han. 2024. ReCG: Bottom-Up JSON Schema Discovery Using a Repetitive Cluster-and-Generalize Framework. *Proceedings of the VLDB Endowment* 17, 11 (2024), 3538–3550.

[103] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. USENIX Association, 10–10.

[104] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2005. Learning from labeled and unlabeled data on a directed graph. In *Proceedings of the 22nd international conference on Machine learning*. 1036–1043.

[105] Amal Zouaq and Felix Martel. 2020. What is the schema of your knowledge graph? leveraging knowledge graph embeddings and clustering for expressive taxonomy learning. In *Proceedings of the international workshop on semantic big data*. 1–6.