Cardinality Estimation using Label Probability Propagation for Subgraph Matching in Property Graph Databases

Leonard Wörteler
University of Konstanz
Konstanz, Germany
leonard.woerteler@uni.kn

Moritz Renftle*
Karlsruhe Institute of Technology
Karlsruhe, Germany
moritz.renftle@kit.edu

Theodoros Chondrogiannis
University of Konstanz
Konstanz, Germany
theodoros.chondrogiannis@uni.kn

Michael Grossniklaus
University of Konstanz
Konstanz, Germany
michael.grossniklaus@uni.kn

ABSTRACT

Estimating query result cardinality is a central task of cost-based database query optimizers, enabling them to identify and avoid excessively large intermediate results. While cardinality estimation has been studied extensively in relational databases, research in the setting of graph databases has been more limited. In this paper, we address the problem of cardinality estimation for subgraph matching on property graph databases. Our novel cardinality estimation technique starts from a small amount of statistical information about node labels and relationship types, which is propagated along the graph query pattern in terms of label probabilities. Additionally, estimation quality can be improved by providing information about labels or properties to our technique, if available. In our experimental evaluation, we compare our approach to state-of-the-art cardinality estimation techniques for subgraph matching for property graph, RDF, and relational databases, and we demonstrate that our technique offers the best trade-off between accuracy and efficiency.

1 INTRODUCTION

Cardinality estimation is the most important factor in cost-based query optimization [33]. Without (reasonably) accurate cardinality estimates, a query optimizer is unable to compute the cost of a candidate execution plan in order to reliably predict its execution time [18]. Failing to do so can lead to the selection of a suboptimal or even catastrophic execution plan. At the same time, the calculation of cardinality estimates needs to be fast (ideally $\ll 1$ ms) as it will be applied to a large number of subplans during the enumeration of plan candidates. Finally, cardinality estimation also needs to be able to work from base statistics that can be managed efficiently. As a consequence, a well-balanced trade-off between accuracy and efficiency is a key requirement in the design of an effective cardinality estimation technique. While in the setting of relational database systems cardinality estimation is a well-studied problem [14, 15, 19, 23, 29, 37, 46], similar efforts in the setting of graph database systems are relatively recent and few [9, 24, 26, 27, 35].

The specific focus of this paper is cardinality estimation of subgraph matching queries in property graph databases. Subgraph matching is an important class of graph queries that has wide-ranging applications. For example, Sahu et al. [32] found that the five graph problems that are most relevant to both researchers and practitioners all involve subgraph matching. Additionally, subgraph matching is a central concept in graph query languages such as SPARQL [38] for RDF data and GQL [8], a new standard query language for property graph data, which will be based on openCypher [13], PGQL [1], and G-CORE [3].

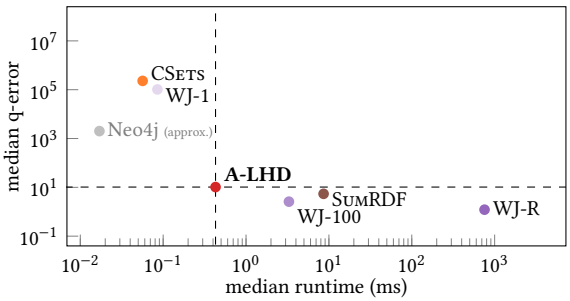


Figure 1: Comparison of the trade-off between accuracy and efficiency of our technique to the state of the art

In this paper, we present a novel technique to estimate the result cardinality of subgraph matching queries that is specifically designed for property graph databases. Our technique requires only a small amount of statistical information about node labels and relationship types. While this information is slightly more complex than the information used by Neo4j, our closest competitor, it can still be collected with very little overhead while yielding a substantial improvement in accuracy. In addition, our technique can also use optional information about label hierarchy and label disjointness to further improve estimates. At the core of our technique is a statistical model that defines how information is propagated and transformed along a pattern in order to estimate the result cardinality.

The main contribution of this paper is to demonstrate that our cardinality estimation technique achieves a more balanced trade-off between accuracy and efficiency than state-of-the-art techniques. Figure 1 plots median q-error (accuracy) vs. median runtime of the cardinality estimator (efficiency). The results clearly indicate that there is, to the best of our knowledge, no competitor that dominates our novel cardinality estimation technique, denoted as A-LHD, in terms of this trade-off. The specific technical contributions of this paper are as follows.

- We specify the statistical information required by our approach and identify additional information that can optionally be used to improve accuracy (Section 4).

*Author's work was conducted while at the University of Konstanz.

- We present a cardinality estimation model that defines for each operator of a simple algebra how it manipulates the statistical information (Section 5).
- We evaluate the accuracy and efficiency of our technique w.r.t. state-of-the-art approaches and study if additional statistical information improves estimations (Section 6).

We begin with a discussion on the state of the art in cardinality estimation techniques for subgraph matching in Section 2. We then formalize the property graph model and present our algebra that can represent subgraph matching queries in Section 3. Concluding remarks are given in Section 7.

2 RELATED WORK

While cardinality estimation has been studied extensively in the context of relational databases [14], the research on cardinality estimation on graph databases and, in particular on property graphs, has been much more limited [27]. Most existing methods focus on cardinality estimation for graph queries on RDF graphs [24, 35, 36].

Characteristic Sets [24] is the cardinality estimation method used by the high-performance triplestore RDF-3X [25]. The characteristic set of a node is the set of types of its edges and functions as an identifier, i.e., nodes with the same set represent the same type of entity. By maintaining the counts of all unique sets, result cardinality is estimated as follows. First, the query pattern is decomposed into non-overlapping star subpatterns the cardinality of which is given by the precomputed counts. Then all remaining joins between edge endpoints are estimated using the independence assumption. While Characteristic Sets can be used for cardinality estimation on property graphs, it is not a good fit. In contrast to RDF graphs where all aspects of an entity have to be modeled using edges (triples), property graphs additionally offer properties and labels. The fact that property graph patterns do not often neatly decompose into stars, along with the high number of potentially correlated joins, lead to severe underestimation of result cardinality.

Another approach for cardinality estimation on RDF graphs is to compute estimations over graph summaries [7, 10, 21, 43]. SumRDF [35] is a recent work in this area that provides highly accurate estimations together with guaranteed error bounds. SumRDF first builds a graph summary by merging together similar nodes in the RDF graph and recording edge multiplicities. Naturally, the size of the summary, that is dictated by multiple parameters, strongly influences both the estimation results and the runtime. As Park et al. [27] discuss in their study, SumRDF produces fairly accurate estimations, but the graph summary can grow very large, which renders the approach slow.

With regard to cardinality estimation on property graphs, Gubichev [9] pioneered the technique that has been adopted by Neo4j. Gubichev's approach requires the maintenance of the following statistical information: a) for each label ℓ, the number of nodes with label ℓ, and b) for every label ℓ, relationship type t, and direction α, the count of all node-relationship pairs (n, r) such that n has label ℓ, r has type t and direction α, and n is an endpoint of r. The cardinality of very small patterns is estimated from these statistics directly. The cardinality of larger patterns is estimated by combining partial estimates and upper bounds on the cardinality of subpatterns. In contrast to our work, in order to combine partial estimates, Gubichev's approach always assumes that relationships are independent, which often leads to underestimation of the result cardinality. Yakovets et al. [4, 42, 44] have also proposed methods for cardinality estimation on property graphs. However, despite being lightweight and producing good estimates, these methods are limited to cardinality estimation for path patterns.

Another line of research aims at the adaptation of techniques from relational to graph databases [27, 39]. Park et al. [27] adapted Wander Join [19], a sampling-based approach for online aggregation, for cardinality estimation of subgraph matching queries. Wander Join performs a predefined number of random walks to gather samples and estimate the cardinality of the given pattern. This makes Wander Join quite flexible as the number of walks determines the trade-off between accuracy and runtime. However, as we show in our experimental evaluation, when configured to produce estimates as accurate as our approach, the runtime of Wander Join is very high, thus making the approach unsuitable for query optimization. Paradies et al. [26] employ two techniques that come from the relational model, i.e., the detection of soft functional dependencies and histograms that represent node degree distributions. This approach however considers neither node labels nor relationship types, and can only be used to estimate the cardinality of patterns of up to two nodes.

Finally, several cardinality estimation techniques have been proposed for XQuery processing in XML databases [20, 28, 40, 45]. As these techniques exploit properties specific to XML data (e.g., strictly tree-structured data), they are not directly applicable to property graph databases.

3 SUBGRAPH MATCHING

Our approach is designed for graph databases that store information according to the *Property Graph Model*. We adapt the definition of property graphs by Angles [2] to match the specifications of the data model that we use throughout this paper.

Definition 3.1 (Property Graph). A *property graph* is a directed labeled multigraph $G = (N, R, P, L, T, \rho, \pi, \lambda, \tau)$ where: N is a finite set of nodes; R is a finite set or relationships; P is a set of properties, i.e., key-value pairs; L is a set of node labels; T is a set of relationship types; $\rho : R \rightarrow N \times N$ is a function that assigns nodes to relationships; $\pi : N \cup R \rightarrow \mathcal{P}(P)$ is a function that assigns properties to nodes and relationships; $\lambda : N \rightarrow \mathcal{P}(L)$ is a function that assigns sets of labels to nodes; $\tau : R \rightarrow T$ is a function that assigns types to relationships.

Figure 2 shows a property graph representing a fictional network that we use as a running example.

3.1 Subgraph Matching Queries

A *subgraph matching query* q on a property graph G takes a *graph pattern* as input Π and returns a set of matching *property subgraphs*. In what follows, we formally define these concepts.

Definition 3.2 (Property Subgraph). A *property subgraph*, or simply subgraph, of a property graph $G = (N, R, P, L, T, \rho, \pi, \lambda, \tau)$ is a directed labeled multigraph $G' = (N', R', P', L', T', \rho, \pi, \lambda, \tau)$ with $N' \subseteq N$, $R' \subseteq R$, $P' \subseteq P$, $L' \subseteq L$, and $T' \subseteq T$, such that $\forall r \in R' : (\rho(r) = (n, n')) \Rightarrow n, n' \in N'$.

Definition 3.3 (Graph Pattern). A *graph pattern* over a property graph $G = (N, R, P, L, T, \rho, \pi, \lambda, \tau)$ is a directed labeled multigraph $\Pi = (N_\Pi, R_\Pi, P, L, T, \rho_\Pi, \pi_\Pi, \lambda_\Pi, \tau_\Pi)$, where ρ_Π, π_Π, and λ_Π are defined analogously to ρ, π, and λ, respectively, and $\tau_\Pi : R_\Pi \rightarrow \mathcal{P}(T)$ assigns a *set* of possible types to each relationship in the pattern.

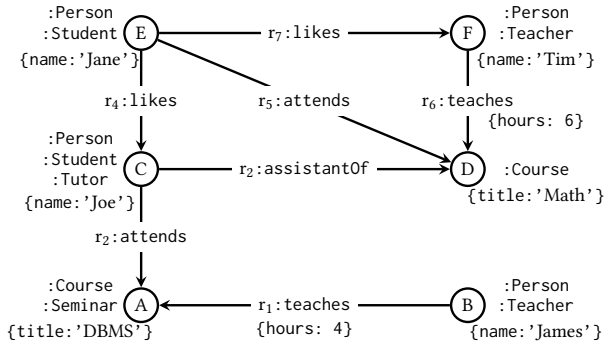


Figure 2: Example property graph

Definition 3.4 (Subgraph Matching Query). Given a graph pattern $\Pi = (N_\Pi, R_\Pi, P, L, T, \rho_\Pi, \pi_\Pi, \lambda_\Pi, \tau_\Pi)$, a property graph $G = (N, R, P, L, T, \rho, \pi, \lambda, \tau)$, and a function $\delta : R_\Pi \to 0|1$ that assigns to each relationship its *directedness*, i.e., whether its direction must be preserved when matching the pattern against a graph G, a *subgraph matching query*, or simply graph query, $q(G, \Pi, \delta)$ returns the set M of all *mappings* (functions) $\mu : (N_\Pi \cup R_\Pi) \to (N \cup R)$ s.t.

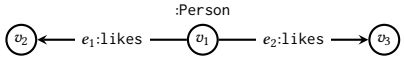
- $\left(\bigcup_{v \in N_\Pi} \mu(v), \bigcup_{e \in R_\Pi} \mu(e), P, L, T, \rho, \pi, \lambda, \tau\right)$ is a subgraph of G that is homomorphic to Π subject to δ,
- $\forall e, e' \in R_\Pi : e \neq e' \Rightarrow \mu(e) \neq \mu(e')$ ("edge isomorphism"),
- $\forall v \in N_\Pi : \pi(\mu(v)) \supseteq \pi_\Pi(v) \wedge \lambda(\mu(v)) \supseteq \lambda_\Pi(v)$,
- $\forall e \in R_\Pi : \pi(\mu(e)) \supseteq \pi_\Pi(e) \wedge \tau(\mu(e)) \in \tau_\Pi(e)$.

The size of the result set of a subgraph matching query q over a property graph G is called the *result cardinality*.

To define the results of a graph pattern match we use the semantics of Neo4j (sometimes called "Cyphermorphism"), which uses homomorphism semantics for matching nodes and isomorphism semantics for matching relationships. This means that two distinct nodes in the pattern may be mapped to the same node in the subgraph, but no two distinct relationships in the pattern may be mapped to the same relationship in the subgraph.

3.2 Property Graph Algebra

The method presented in this paper estimates the cardinality of the whole graph pattern incrementally. In each step, an additional component of the pattern (e.g., existence of a relationship or a node label) is considered. In order to derive efficiently computable estimation formulas for the steps in this process we map them to a set of *operators* from a sample property graph algebra that is based on the work of Hölsch and Grossniklaus [11]. Our algebra has the sole purpose of defining the result of a subgraph matching query $q(G, \Pi, \delta)$ the cardinality of which we want to estimate, rather than specifying the evaluation of a general graph query. The operators of our algebra use node and relationship variables to track node and relationship matches respectively. Operators can introduce additional variables and reference previously introduced ones by name. Variables can map to multiple nodes and relationships and, thus, one expression can match multiple subgraphs. For instance, consider the graph pattern shown below that is matched against the property graph from our running example in Figure 2.



An expression corresponding to this graph pattern can be represented by the following operator sequence.

$$\bigcirc_{v_1}^{G} \longrightarrow \sigma_{v_1:\texttt{Person}} \longrightarrow \overset{\varepsilon}{v_1} \xrightarrow{e_1:\texttt{likes}} v_2 \longrightarrow \overset{\varepsilon}{v_1} \xrightarrow{e_2:\texttt{likes}} v_3$$

To estimate the cardinality of a result set we start from base statistics which are updated iteratively as operators from the expression representing the graph pattern are processed.

In what follows, we define how our algebraic operators manipulate the result, while we specify in Section 5 how they affect result cardinality and label probability. The result of an expression X is denoted by $\text{res}(X)$.

GetNodes operator The GetNodes operator produces the set of mappings corresponding to all single node subgraphs of the property graph G:

$$\text{res}(\bigcirc_v^G) := \{\{v \mapsto n\} \mid n \in N\}.$$

Node variable v can be matched to any node of G. For instance, in the property graph G of our running example in Figure 2, the result of $\text{res}(\bigcirc_v^G)$ is the set that contains six subgraphs, each one containing a distinct node of G, and v refers to any of these nodes.

LabelSelection operator The LabelSelection operator $\sigma_{v:\ell}$ takes an expression $\mathcal{R}$ as input and returns only those of its results that correspond to subgraphs for which the node matched by node variable v has label ℓ.

$$\text{res}(\sigma_{v:\ell}(\mathcal{R})) := \{\mu \mid \mu \in \text{res}(\mathcal{R}) \wedge \ell \in \lambda(\mu(v))\}.$$

For instance, consider graph G in our running example in Figure 2 and assume that the input to the LabelSelection operator are all the single node subgraphs of G. For the label 'Person', the result of $\sigma_{v:\texttt{Person}}(\bigcirc_v^G)$ contains the single node subgraphs associated with B, C and E, i.e., the nodes that have the label 'Person'. Node variable v refers to any node in {B, C, E}.

PropertySelection operator The PropertySelection operator $\sigma'_{v\{\phi\}}$ takes an expression $\mathcal{R}$ as input and returns only those of its results that correspond to subgraphs for which the node matched by node variable v has all properties in the set $\theta \subseteq P$. Hence, the result of PropertySelection is

$$\text{res}(\sigma'_{v\{\theta\}}(\mathcal{R})) := \{\mu \mid \mu \in \text{res}(\mathcal{R}) \wedge \theta \subseteq \pi(\mu(v))\}.$$

Consider graph G in Figure 2. The result for $\sigma'_{v\{\texttt{name: 'Tim'}\}}(\bigcirc_v^G)$ contains one mapping in which v is mapped to node F.

Expand operator The Expand operator takes an expression $\mathcal{R}$ as input and expands those mappings in $\text{res}(\mathcal{R})$ that have relationships r of types $T' \subseteq T$ and direction $\alpha \in \{\rightarrow, *, \leftarrow\}$. The result of the Expand contains one mapping per input mapping and qualifying relationship.

$$\text{res}(\varepsilon_{v\ \alpha\ v'}^{e:T'}(\mathcal{R})) := \{\mu \cup \{e \mapsto r, v' \mapsto n\} \mid \mu \in \text{res}(\mathcal{R}) \wedge r \in R \wedge (\mu(v), n) \in \rho_\alpha(r) \wedge \tau(r) \in T'\},$$

where we set $\rho_\rightarrow(r) := \{(n, n')\}$, $\rho_\leftarrow(r) := \{(n', n)\}$, $\rho_*(r) := \{(n, n'), (n', n)\}$ for $\rho(r) = (n, n')$.
For instance, consider again the graph G of our running example in Figure 2. If the input $\text{res}(\mathcal{R})$ of the Expand operator only contains the mapping $\{v \mapsto \text{E}\}$ corresponding to the single-node subgraph of E, then the result of $\varepsilon_{v \xrightarrow{r:\texttt{likes}} v'}(\mathcal{R})$ will be $\{\{v \mapsto \text{E}, r \mapsto \text{r}_4, v' \mapsto \text{C}\}, \{v \mapsto \text{E}, r \mapsto \text{r}_7, v' \mapsto \text{F}\}\}$. Now, v refers to the starting point of each newly added relationship, and v' refers to the end point of the same relationship.

MergeOn operator The MergeOn operator $\triangledown_{v=v'}$ takes an expression $\mathcal{R}$ and unifies two node variables from its result. In other words, for two given node variables v, v' MergeOn chooses only those mappings from $\text{res}(\mathcal{R})$ where variables v and v' refer to exactly the same node, and omits v'. Hence, the result of MergeOn is

$$\text{res}(\triangledown_{v=v'}(\mathcal{R})) := \{\, \mu \backslash \{v' \mapsto \mu(v')\} \mid \mu \in \text{res}(\mathcal{R}) \wedge \mu(v) = \mu(v') \,\}.$$

For instance, consider the evaluation of the following expression on graph G of our running example in Figure 2.

$$\varepsilon_{v_1} \xrightarrow{e_3:\texttt{attends}} v_4 \left(\varepsilon_{v_2} \xrightarrow{e_2:\texttt{assistantOf}} v_3 \left(\varepsilon_{v_1} \xrightarrow{e_1:\texttt{likes}} v_2 \left(\bigcirc_{v_1}^{G} \right) \right) \right)$$

After evaluating this expression, $\text{res}(\mathcal{R})$ will contain a mapping μ={$v_1 \mapsto$ E, $e_1 \mapsto$ r$_4$, $v_2 \mapsto$ C, $e_2 \mapsto$ r$_5$, $v_3 \mapsto$ D, $e_3 \mapsto$ r$_5$, $v_4 \mapsto$ D}. Since node D is reachable via two different paths from node E, two node variables map to D. This means that μ represents a cyclic subgraph. To ensure that all results have this property, the MergeOn operator $\triangledown_{v_3=v_4}(\mathcal{R})$ is applied. In practice, MergeOn is the operator that allows us to express cyclic graph patterns.

4 STATISTICAL INFORMATION

In this section, we describe the types of statistical information used in our cardinality estimation technique and give an overview of how the proposed algorithm propagates this information along a graph pattern to estimated result cardinality.

4.1 Required Statistical Information

Our cardinality estimation technique requires the following statistical information about nodes and relationships of a property graph $G = (N, R, P, L, T, \rho, \pi, \lambda, \tau)$:

(1) for each label $\ell \in L$, the number of nodes having ℓ, i.e.,

$$NC(\ell) = |\sigma_{v:\ell}(\bigcirc_v^G)|$$

(2) the number of relationships with direction $\alpha \in \{\rightarrow, *, \leftarrow\}$ of type $t \in T$ from nodes with label $\ell_i \in L$ to nodes with label $\ell_j \in L$, i.e.,

$$RC_\alpha(\ell_i, t, \ell_j) = |\sigma_{v':\ell_j}(\varepsilon_{v\,\alpha\,v'}^{r:t}(\sigma_{v:\ell_i}(\bigcirc_v^G)))|$$
$$RC_\alpha(\ell_i, t, *) = |\varepsilon_{v\,\alpha\,v'}^{r:t}(\sigma_{v:\ell_i}(\bigcirc_v^G))|$$
$$RC_\alpha(*, t, \ell_j) = |\sigma_{v':\ell_j}(\varepsilon_{v\,\alpha\,v'}^{r:t}(\bigcirc_v^G))|$$
$$RC_\alpha(*, t, *) = |\varepsilon_{v\,\alpha\,v'}^{r:t}(\bigcirc_v^G)|$$

In comparison to Gubichev's work [9] that has been adopted by Neo4j, our technique requires slightly more information. Instead of counting (ℓ, t)-pairs, our approach maintains (ℓ_i, t, ℓ_j)-triple counts. Instead of linearly, the required space is worst-case quadratic in the number of distinct node labels. However, the worst case only occurs when a graph contains relationships with very diverse sets of labels attached to the source and target nodes. In graphs that (mostly) adhere to some schema, one would not expect to find all label combinations for the two nodes in a relationship. As such, the overall space required is still expected to be small as data sets with more than a hundred distinct node labels are rare.

4.2 Optional Statistical Information

Our technique can use additional statistical information w.r.t the labels and the properties of a graph to improve estimations.

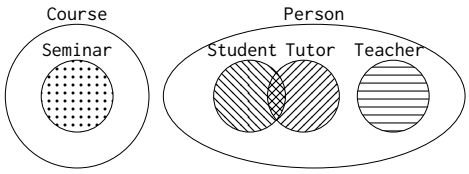


Figure 3: Distribution of node labels

4.2.1 Additional Label Information. Most existing graph database systems including Neo4j assume statistical independence of node labels, which is not a realistic assumption in practice. To explore this further, consider a property graph $G = (N, R, P, L, T, \rho, \pi, \lambda, \tau)$, and labels $\ell_i, \ell_j \in L$. Also, let $\text{P}(v{:}\ell)$ be the probability that any randomly chosen node which variable v maps to has label ℓ. For the relationship between two labels $\ell_i, \ell_j \in L$, we distinguish the following cases:

(1) ℓ_i is a *sublabel* of ℓ_j, i.e., every node that has label ℓ_i also has label ℓ_j but not necessarily vice-versa. Hence we have

$$\text{P}(v{:}\ell_i \cap v{:}\ell_j) = \text{P}(v{:}\ell_i).$$

If ℓ_i is a *sublabel* of ℓ_j we write $\ell_i \preccurlyeq \ell_j$.

(2) ℓ_i and ℓ_j are *disjoint*, i.e., no node has both labels ℓ_i and ℓ_j. Hence we have

$$\text{P}(v{:}l_i \cap v{:}\ell_j) = 0.$$

(3) ℓ_i and ℓ_j are *overlapping*, i.e., nodes that have label ℓ_i may or may not have label ℓ_j and vice-versa. In this case we assume statistical independence, thus we have

$$\text{P}(v{:}\ell_i \cap v{:}\ell_j) \approx \text{P}(v{:}\ell_i) \cdot \text{P}(v{:}\ell_j)$$

Figure 3 shows the distribution of the node labels in our running example. We observe that `Seminar` is a sublabel of `Course`, `Student`, `Tutor` and `Teacher` are sublabels of `Person`, while labels `Student` and `Tutor` are overlapping.

Information about sublabels is stored using the *label hierarchy* data structure H_L. For ease of reading, given a label ℓ we denote by $HN(\ell)$ the node of H_L that ℓ is associated with. Given a property graph $G = (N, R, P, L, T, \rho, \pi, \lambda, \tau)$, H_L of G stores all labels in L in a hierarchical fashion such that:

- the root node of H_L is associated with $*$, i.e., a virtual label that acts as superlabel to all labels in L,
- if for two labels ℓ_i and ℓ_j we have that ℓ_i is a sublabel of ℓ_j then HN_i is a descendant of HN_j.

H_L is utilized for cardinality estimation as follows. Given a set of labels $L_v \subseteq L$ and a node variable v, let $\text{P}(\bigcap_{\ell \in L_v} v{:}\ell)$, i.e., the probability that any node which v maps to is associated with all labels in L_v, be the probability we wish to estimate. By accessing H_L, we determine all pairs of labels $\ell_i, \ell_j \in L_v$ that are in a sublabel relationship. For every ℓ_j that is a sublabel of ℓ_i we have

$$\text{P}(v{:}\ell_i \cap v{:}\ell_j) = \text{P}(v{:}\ell_j).$$

Hence, to compute $\text{P}(\bigcap_{\ell \in L_v} v{:}\ell)$ we can disregard all labels $\ell_i \in L_v$ such that $\exists \ell_j \in L_v : \ell_j \preccurlyeq \ell_i$, as the probability that a node has label ℓ_i is already covered by its sublabels. Figure 4 illustrates the label hierarchy of our running example.

Information about disjoint labels is stored using the *label partition* data structure. The label partition of G is a set of pairwise disjoint sets of labels $D_L = \{D_1, ..., D_m\}$ such that for every pair ℓ_i, ℓ_j where $\ell_i \in D_i$, $\ell_j \in D_j$ and $i \neq j$, ℓ_i and ℓ_j are disjoint. For instance, the label partition of our running example is:

{{`Seminar`, `Course`}, {`Student`, `Tutor`, `Teacher`, `Person`}}.

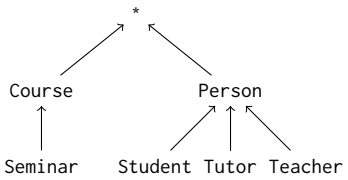


Figure 4: Label hierarchy of our running example

Given again a set of labels $L_v \subseteq L$, a node variable v, and $\mathrm{P}(v{:}L_v)$ the probability we wish to estimate, we utilize the label partition D_L to determine if there exists a pair of labels $\ell_i, \ell_j \in L_v$ such that ℓ_i and ℓ_j are disjoint. If so, we immediately determine that

$$\mathrm{P}(\textstyle\bigcap_{\ell \in L_v} v{:}\ell) = \mathrm{P}(v{:}\ell_i \cap v{:}\ell_i) = 0.$$

Thus, a single disjoint pair of labels in L_v is sufficient to determine that no node mapping to v can have all labels in L_v.

Consider again our running example (Figure 2). Assume we want to estimate the probability $\mathrm{P}(\bigcap_{\ell \in L_v} v{:}\ell)$ of any node which variable v maps to having all labels L_v = {`Person`, `Student`, `Teacher`}. First, we disregard label `Person` from L_v as both `Student` and `Teacher` are its sublabels. Then, we observe that labels `Student` and `Teacher` are disjoint, hence

$$\mathrm{P}(\textstyle\bigcap_{\ell \in L_v} v{:}\ell) = \mathrm{P}(v{:}\texttt{Student} \cap v{:}\texttt{Teacher}) = 0.$$

Similarly, for L'_v = {`Person`, `Student`, `Tutor`} we have

$$\mathrm{P}(\textstyle\bigcap_{\ell \in L'_v} v{:}\ell) \approx P(v{:}\texttt{Student}) \cdot \mathrm{P}(v{:}\texttt{Tutor})$$

as `Student` and `Tutor` are overlapping sublabels of `Person`.

As the focus of this paper is on cardinality estimation, we do not present any new techniques to obtain and maintain this optional statistical information. However, since both the Label Hierarchy and the Label Partition assume the existence of a schema, existing techniques for schema inference [17] and dependency discovery [16] can be used to gather such information. Furthermore, in most cases, schema evolution can be expected to be very infrequent compared to updates in the data. In scenarios without an (explicit or implicit) schema or a schema that changes frequently, this optional information is not expected to be useful and can be omitted.

4.2.2 Additional Property Information. Two aspects of properties are of interest for the cardinality estimation process, namely (1) the existence of the property key and (2) the correct value. To take into account the existence of property keys, we maintain statistics about pairs of property key/type for relationships and property key/label for nodes (and additionally wildcards $*$ instead of label/type). Since estimating cardinality based on properties is essentially the same problem as estimating the selectivity of a selection in relational systems, we follow the approach of the open-source RDBMS PostgreSQL [30], i.e., we store raw counts, numbers of distinct values, and the ten most frequent values together with their respective frequencies. This could potentially lead to huge amounts of statistics being kept. However, labels/types and property keys attached to nodes and relationships tend to be highly correlated in practical applications, so only small subsets of the full cross products have to be stored. Based on these statistics we can estimate the selectivity $sel : (L \cup T \cup \{*\}) \times P \to [0, 1]$, namely the probability $sel(lt, p)$ that a node or relationship has a certain property p given that it has the specified label or relationship type lt.

4.3 Propagation of Statistical Information

Recall that our estimation technique represents the graph pattern Π of a subgraph matching query $q(G, \Pi, \delta)$ as an operator sequence that evaluates to the result of q. To estimate the result cardinality, our technique starts from the initial operator, i.e., a GetNodes operator, and iteratively considers each subsequent operator. Every prefix of the operator sequence corresponds to a subpattern π_i. During cardinality estimation, for each such subpattern π_i our approach maintains

- the **estimated result cardinality** C of π_i and
- a mapping M that assigns to every pair (v, ℓ) the **label probability** $\mathrm{P}[\pi_i](v{:}\ell)$ that a randomly chosen node bound to node variable v has label $\ell \in L$.

The number of label probabilities $|M|$ maintained by our technique for each subsequence is equal to the current number of node variables multiplied by the number of labels in L.

To compute the result cardinality and the mapping for a subpattern π_i, we consider the result cardinality and the mapping of the previous subsequence π_{i-1}, the available statistical information, and the type of the subsequent operator. Note that each additional operator potentially affects the probabilities of all labels on all node variables. However, updating the mappings for all variables would require a quadratic number of estimations. Hence, since we aim for a trade-off between efficiency and accuracy, we update the mappings only of variables directly mentioned by the operator.

Algorithm 1 gives the pseudocode of our cardinality estimation technique. The algorithm takes an operator sequence Op as input and computes the cardinality C of the result evaluated by Op using the statistics we described above. First, the estimated cardinality C and label probability mapping M are initialized in Line 1. Then, between Lines 2 and 17 the operators of the input sequence are examined one by one. The previous operator's estimated cardinality C_{prev} and the mappings M_{prev} are stored in Line 3, and M and C are then updated between Lines 4 and 16 depending on the current operator Op_i. We elaborate on how the estimated result cardinality C and mapping M is computed for each operator in Section 5. Finally, the estimated result cardinality C is returned in Line 17.

Since our technique estimates result cardinality in a greedy fashion, the order in which operators are processed can influence the quality of the resulting estimations. While determining the best ordering is out of the scope of our work, we employ heuristics to avoid propagating information along long chains of relationships. We start by choosing the node variable with the highest overall degree as the starting point. Then we expand the pattern in breadth-first order. Label and property selections are introduced as early as possible. Relationships that would close cycles are deferred until the end, at which point their respective Expand operators as well as the required MergeOn operators are introduced. In a preliminary experiment, we compared the order created by our heuristic method against 100 randomly generated orders per query. The results showed that our order ranks on average in the top-30% among the randomly generated ones in terms of accuracy.

5 CARDINALITY ESTIMATION TECHNIQUE

For each of the operators introduced in Section 3.2, we define how its result cardinality is estimated in relation to its input and how label probabilities change as a result of the operator.

Algorithm 1: Cardinality Estimation Algorithm

Input: Operator sequence $Op = \langle Op_1, \ldots, Op_m \rangle$
Output: Estimated cardinality C

$C \leftarrow 0, M \leftarrow \emptyset$
for $i \in \{1, \ldots, m\}$ **do**
  $C_{\text{prev}} \leftarrow C, M_{\text{prev}} \leftarrow M$
  switch Op_i **do**
    case $\bigcirc_n^G$ **do** // Section 5.1
      $C \leftarrow NC(*)$
      $M \leftarrow \left\{ (n, \ell) \mapsto \frac{NC(\ell)}{NC(*)} \;\middle|\; \ell \in L \right\}$
    case $\sigma_{n:\ell}$ **do** // Section 5.2
      $C \leftarrow C_{\text{prev}} \cdot M_{\text{prev}}[n{:}\ell]$
      update M using M_{prev}
    case $\sigma'_{n\{\phi\}}$ **do** // Section 5.3
      $(C, M) \leftarrow$ PROPSELCARD$(\sigma'_{n\{\phi\}}, C_{\text{prev}}, M_{\text{prev}})$
    case $\varepsilon_{n \xrightarrow{L} n'}$ **do** // Section 5.4
      $(C, M) \leftarrow$ EXPANDCARD$(\varepsilon_{n \xrightarrow{L} n'}, C_{\text{prev}}, M_{\text{prev}})$
    case $\nabla_{n_i = n_j}$ **do** // Section 5.5
      $(C, M) \leftarrow$ MERGECARD$(\nabla_{n_i = n_j}, C_{\text{prev}}, M_{\text{prev}})$
return C

Note that to determine the result cardinality and the label probabilities for each operator, we assume the availability of the optional statistical information described in Section 4.2. If either or both of H_L and D_L are not available, they can be substituted as $H_L = \{\ell \to * \mid \ell \in L\}$ and $D_L = \{L\}$, respectively. If no property statistics are available, all property selectivities $sel(\cdot, \cdot)$ are estimated as 10%.

5.1 Estimation for GetNodes

Result cardinality. The result cardinality of the GetNodes operator is given by the number of nodes in the database, i.e.,

$$\left|\bigcirc_v^G\right| = NC(*).$$

New label probabilities. The probability of drawing a node matched by v having a label ℓ from the result of GetNodes equals the fraction of nodes in the database having label ℓ, i.e.,

$$\mathrm{P}[\bigcirc_v^G](v{:}\ell) = \frac{NC(\ell)}{NC(*)}.$$

This applies for all labels $\ell \in L$.

5.2 Estimation for LabelSelection

Result cardinality. The result cardinality of the LabelSelection operator is given by the probability of a node matched by v in the input $\mathcal{R}$ having the label ℓ multiplied by the size of the input, i.e.,

$$|\sigma_{v:\ell}(\mathcal{R})| = \mathrm{P}[\mathcal{R}](v{:}\ell) \cdot |\mathcal{R}|$$

New label probabilities. We derive the new label probabilities for a label ℓ' after the LabelSelection operator of a label ℓ.

- **Case 1:** $\ell = \ell'$

  In the result of the LabelSelection all nodes matched by n have the label ℓ. Hence, if $\ell = \ell'$ then

$$\mathrm{P}[\sigma_{v:\ell}(\mathcal{R})](v{:}\ell') = 1.$$

- **Case 2:** $\ell \neq \ell'$ *and* $\ell \preccurlyeq \ell'$

  Similar to Case 1, in the result of LabelSelection all nodes matched by n have the label ℓ. Since ℓ is a sublabel of ℓ', then, by definition, all nodes matched by v also have label ℓ', i.e.,

$$\mathrm{P}[\sigma_{v:\ell}(\mathcal{R})](v{:}\ell') = 1 \quad \text{iff} \quad \ell \preccurlyeq \ell'.$$

- **Case 3:** $\ell \neq \ell'$ *and* $\ell' \preccurlyeq \ell$

  If ℓ' is a sublabel of ℓ, all the nodes in the result of LabelSelection that have label ℓ' have ℓ as well. Hence, we have

$$\mathrm{P}[\sigma_{v:\ell}(\mathcal{R})](v{:}\ell') = \frac{\mathrm{P}[\mathcal{R}](v{:}\ell' \cap v{:}\ell)}{\mathrm{P}[\mathcal{R}](v{:}\ell)} = \frac{\mathrm{P}[\mathcal{R}](v{:}\ell')}{\mathrm{P}[\mathcal{R}](v{:}\ell)}$$

- **Case 4:** ℓ, ℓ' are overlapping

  If labels ℓ and ℓ' are overlapping, we have

$$\begin{aligned}\mathrm{P}[\sigma_{v:\ell}(\mathcal{R})](v{:}\ell') &= \frac{\mathrm{P}[\mathcal{R}](v{:}\ell \cap v{:}\ell')}{\mathrm{P}[\mathcal{R}](v{:}\ell)} \\ &\approx \frac{\mathrm{P}[\mathcal{R}](v{:}\ell) \cdot \mathrm{P}[\mathcal{R}](v{:}\ell')}{\mathrm{P}[\mathcal{R}](v{:}\ell)} = \mathrm{P}[\mathcal{R}](v{:}\ell')\end{aligned}$$

- **Case 5:** ℓ, ℓ' are disjoint
  Finally, if labels ℓ and ℓ' are disjoint, we have

$$\mathrm{P}[\sigma_{v:\ell}(\mathcal{R})](v{:}\ell') = \frac{\mathrm{P}[\mathcal{R}](v{:}\ell \cap v{:}\ell')}{\mathrm{P}[\mathcal{R}](v{:}\ell)} = 0$$

5.3 Estimation for PropertySelection

Given the expression $\sigma'_{v\{\theta\}}(\mathcal{R})$, we first determine if v refers to a node or a relationship variable. In order to be able to treat both cases alike, we gather all labels or types that *can* be attached to nodes or relationships matched by the variable, i.e.,

$$L' = \begin{cases} \{\ell \mid \ell \in L \wedge \mathrm{P}[\mathcal{R}](v{:}\ell) > 0\} & \text{if } v \text{ is a node variable} \\ \tau_\Pi(v) & \text{otherwise} \end{cases}$$

Result cardinality. We calculate the result cardinality as the cardinality of $\mathcal{R}$ multiplied by the *selectivity* of the property predicate θ, which we estimate as the average over selectivities estimated for each of the possible labels or types. To counteract catastrophic underestimation, we assume that multiple predicates on the same node are correlated and choose the selectivity of the most selective one among them.

$$\begin{aligned}\left|\sigma'_{v\{\theta\}}(\mathcal{R})\right| &= |\mathcal{R}| \cdot \mathrm{P}[\mathcal{R}](P' \subseteq \pi(\mu(v))) \\ &\approx |\mathcal{R}| \cdot \min_{p \in \theta} \begin{cases} \mathrm{avg}_{lt \in L'} \, sel(lt, p) & \text{if } L' \neq \emptyset \\ sel(*, p) & \text{otherwise} \end{cases}\end{aligned}$$

New label probabilities. If v is a relationship variable, we do not change the label probabilities for any nodes. If v is a node variable instead, we set the probabilities of labels for which a selectivity of 0% is estimated to zero and scale the others' probabilities by the inverse of the overall selectivity.

$$\mathrm{P}[\sigma'_{v\{\theta\}}(\mathcal{R})](v{:}\ell) \approx \begin{cases} \mathrm{P}[\mathcal{R}](v{:}\ell) & \text{if } v \text{ is a rel. variable} \\ 0 & \text{if } \min_{p \in \theta} sel(\ell, p) = 0 \\ \mathrm{P}[\mathcal{R}](v{:}\ell) \frac{|\mathcal{R}|}{\left|\sigma'_{v\{\theta\}}(\mathcal{R})\right|} & \text{otherwise} \end{cases}$$

In practice we limit the probabilities to the range $[0, 1]$.

5.4 Estimation for Expand

To estimate the cardinality of the Expand operator, we first make the assumption that similar nodes, e.g., nodes with the same label, have similar degree. More specifically, by using the required statistics described in Section 4.1, the average degree of a node $n \in N$ with label $\ell \in L$ by considering only relationships of direction $\alpha \in \{\rightarrow, *, \leftarrow\}$ and relationship types $T' \subseteq T$, where the adjacent node has label ℓ', is:

$$\overline{\deg}_{\ell\ {}^{T'}_{\alpha}\ \ell'} = \frac{\sum_{t \in T'} RC_\alpha(\ell, t, \ell')}{NC(\ell)}$$

while for nodes that do not have any label, we assume that their node degree equals the average degree of all nodes of the property graph G. Consequently, for any input expression $\mathcal{R}$ we have

$$\left|\sigma_{v':\ell'}\left(\varepsilon_{v\ {}^{r:T'}_{\alpha}\ v'}(\sigma_{v:\ell}(\mathcal{R}))\right)\right| \approx |\sigma_{v:\ell}(\mathcal{R})| \cdot \overline{\deg}_{\ell\ {}^{T'}_{\alpha}\ \ell'} \tag{1}$$

while for nodes not having any label, we have

$$\left|\sigma_{v':\ell'}\left(\varepsilon_{v\ {}^{r:T'}_{\alpha}\ v'}(\mathcal{R} \setminus \bigcup_{\ell' \in L} \sigma_{v:\ell'}(\mathcal{R}))\right)\right| \approx \left|\mathcal{R} \setminus \bigcup_{\ell \in L} \sigma_{v:\ell}(\mathcal{R})\right| \cdot \overline{\deg}_{*\ {}^{T'}_{\alpha}\ \ell'}. \tag{2}$$

Result cardinality. To estimate the result cardinality of the Expand operator, we first consider a label partition D_L as described in Section 4.2, and we sort the labels ℓ in each cluster $D_i \in D_L$ in descending order of the product $\mathrm{P}[\mathcal{R}](v{:}\ell) \cdot \frac{1}{NC(\ell)}$. This way labels that cover most of the nodes matched by v in $\mathcal{R}$ and whose number of nodes in the database is closest to $|\mathcal{R}|$ have higher order. If the product is the same for two labels, the label ℓ with higher $\mathrm{P}[R](v{:}\ell)$ comes first. In what follows, as the ordering is essential for the estimation, we treat clusters $D_i \in D_L$ as ordered lists. Also, for ease of exposition, we use $\mathrm{P}(v{:}\ell)$ instead of $\mathrm{P}[\mathcal{R}](v{:}\ell)$.

Having ordered every cluster $D_i \in D_L$, for each labeled node n matched by v we determine its *representative label* as the label $\ell_r \in \lambda(n)$ that occurs first in D_i. The degree of each node is then estimated as the average degree $\overline{\deg}_{\ell_r\ {}^{T'}_{\alpha}\ *}$ of all nodes having the same representative label.

Subsequently, we divide the set of input subgraphs into two subsets. In the first subset, we put those subgraphs where the nodes matched by v have at least one label. In the second subset, we put the subgraphs where the nodes matched by v have no label. Hence, the exact cardinality of the Expand operator is

$$\left|\varepsilon_{v\ {}^{r:T'}_{\alpha}\ v'}(\mathcal{R})\right| = \left|\varepsilon_{v\ {}^{r:T'}_{\alpha}\ v'}\left(\bigcup_{\ell \in D_L} \sigma_{v:\ell}(\mathcal{R})\right)\right| + \left|\varepsilon_{v\ {}^{r:T'}_{\alpha}\ v'}\left(\mathcal{R} \setminus \bigcup_{\ell \in D_L} \sigma_{v:\ell}(\mathcal{R})\right)\right|. \tag{3}$$

In order to estimate the first summand of Equation 3 using the available statistics, we rewrite it as a sum over the clusters D_i of the label partition D_L. Subsequently, for each label $\ell_{ij} \in D_i$, we count the number of subgraphs produced by the Expand operator applied on the nodes in $\mathcal{R}$ that have label ℓ_{ij}, but do not have any of the labels $\ell_{i1}, \ldots, \ell_{i(j-1)}$. Therefore, the degree of nodes that have ℓ_{ij} as their representative label is estimated as $\overline{\deg}_{\ell_{ij}\ {}^{T'}_{\alpha}\ *}$.

For the first summand of Equation 3 we have

$$\begin{aligned}
&\left|\varepsilon_{v\ {}^{r:t}_{\alpha}\ v'}\left(\textstyle\bigcup_{\ell \in L} \sigma_{v:\ell}(\mathcal{R})\right)\right| \\
&\approx \sum_{i=1}^{|D_L|} \left|\varepsilon_{v\ {}^{r:T'}_{\alpha}\ v'}\left(\textstyle\bigcup_{j=1}^{|D_i|} \sigma_{v:\ell_{ij}}(\mathcal{R})\right)\right| \\
&= \sum_{i=1}^{|D_L|} \sum_{j=1}^{|D_i|} \left|\varepsilon_{v\ {}^{r:T'}_{\alpha}\ v'}\left(\sigma_{v:\ell_{ij}}\left(\mathcal{R} \setminus \textstyle\bigcup_{k=1}^{j-1} \sigma_{v:\ell_{ik}}(\mathcal{R})\right)\right)\right| \\
&\overset{(1)}{\approx} \sum_{i=1}^{|D_L|} \sum_{j=1}^{|D_i|} \left|\sigma_{v:\ell_{ij}}\left(\mathcal{R} \setminus \textstyle\bigcup_{k=1}^{j-1} \sigma_{v:\ell_{ik}}(\mathcal{R})\right)\right| \cdot \overline{\deg}_{\ell_{ij}\ {}^{T'}_{\alpha}\ *} \\
&= \sum_{i=1}^{|D_L|} \sum_{j=1}^{|D_i|} |\mathcal{R}| \cdot \mathrm{P}(v{:}\ell_{ij} \cap \overline{\textstyle\bigcup_{k=1}^{j-1} v{:}\ell_{ik}}) \cdot \overline{\deg}_{\ell_{ij}\ {}^{T'}_{\alpha}\ *} \\
&= |\mathcal{R}| \cdot \sum_{i=1}^{|D_L|} \sum_{j=1}^{|D_i|} \mathrm{P}(v{:}\ell_{ij} \cap \textstyle\bigcap_{k=1}^{j-1} \overline{v{:}\ell_{ik}}) \cdot \overline{\deg}_{\ell_{ij}\ {}^{T'}_{\alpha}\ *}.
\end{aligned} \tag{4}$$

By treating D_i as an ordered list, we ensure that each node is considered exactly once, as identified by its representative label. The ordering also enables us to keep track of the proportion of nodes that have not yet been covered in each iteration.

Next, we estimate the fraction of mappings $\mu \in \mathrm{res}(\mathcal{R})$ where node $\mu(v)$ has the label ℓ_{ij} but none of the labels $L_i^{j-1} := \{\ell_{i1}, \ldots, \ell_{i(j-1)}\}$. For every label $\ell \in D_i \setminus L_i^{j-1}$ it holds that

$$\begin{aligned}
&\mathrm{P}(v{:}\ell \cap \textstyle\bigcap_{\ell' \in L_i^{j-1}} \overline{v{:}\ell'}) \\
&= \mathrm{P}(v{:}\ell \mid \textstyle\bigcap_{\ell' \in L_i^{j-1}} \overline{v{:}\ell'}) \cdot \mathrm{P}(\textstyle\bigcap_{\ell' \in L_i^{j-1}} \overline{v{:}\ell'}) \\
&= \left(1 - \mathrm{P}(\overline{v{:}\ell} \mid \textstyle\bigcap_{\ell' \in L_i^{j-1}} \overline{v{:}\ell'})\right) \cdot \mathrm{P}(\textstyle\bigcap_{\ell' \in L_i^{j-1}} \overline{v{:}\ell'}) \\
&= \left(1 - \frac{\mathrm{P}(\bigcap_{\ell' \in L_i^{j-1} \cup \{\ell\}} \overline{v{:}\ell'})}{\mathrm{P}(\bigcap_{\ell' \in L_i^{j-1}} \overline{v{:}\ell'})}\right) \cdot \mathrm{P}(\textstyle\bigcap_{\ell' \in L_i^{j-1}} \overline{v{:}\ell'}) \\
&= \mathrm{P}(\textstyle\bigcap_{\ell' \in L_i^{j-1}} \overline{\ell'}) - \mathrm{P}(\bigcap_{\ell' \in L_i^{j-1} \cup \{\ell\}} \overline{v{:}\ell'})
\end{aligned} \tag{5}$$

For all labels $\ell' \in L_i^{j-1}$ that are sublabels of ℓ, the probability of drawing a node that has neither ℓ nor ℓ' is $\mathrm{P}(\overline{v{:}\ell} \cap \overline{v{:}\ell'}) = \mathrm{P}(\overline{v{:}\ell})$.

We can simplify the subexpression $\mathrm{P}(\bigcap_{\ell' \in L_i^{j-1} \cup \{\ell\}} \overline{v{:}\ell'})$ by only considering labels ℓ' that do not have any superlabels in $L_i^{j-1} \cup \{\ell\}$. For the resulting set $S_i^j := L_i^{j-1} \setminus \{\ell' \in L_i^{j-1} \mid \exists \ell \in L_i^{j-1} : \ell' \preccurlyeq \ell\}$ we assume independence and obtain

$$\mathrm{P}\Big(\bigcap_{\ell' \in S_i^j} \overline{v{:}\ell'}\Big) \approx \prod_{\ell' \in S_i^j} 1 - \mathrm{P}(v{:}\ell').$$

Hence, for Equation 5 we have

$$\mathrm{P}(v{:}\ell \cap \bigcap_{\ell' \in L_i^{j-1}} \overline{v{:}\ell'}) = \left(\prod_{\ell' \in S_i^{j-1}} 1 - \mathrm{P}(v{:}\ell')\right) - \left(\prod_{\ell' \in S_i^j} 1 - \mathrm{P}(v{:}\ell')\right).$$

Similarly, the second summand of Equation 3 is estimated as

$$\begin{aligned}
\left|\varepsilon_{v\ {}^{r:T'}_{\alpha}\ v'}(\mathcal{R} \setminus \textstyle\bigcup_{\ell \in L} \sigma_{v:\ell}(\mathcal{R}))\right| &\overset{(2)}{\approx} \left|\mathcal{R} \setminus \textstyle\bigcup_{\ell \in L} \sigma_{v:\ell}(\mathcal{R})\right| \cdot \overline{\deg}_{*\ {}^{T'}_{\alpha}\ *} \\
&= |\mathcal{R}| \cdot \mathrm{P}(\overline{\textstyle\bigcup_{\ell \in L} v{:}\ell}) \cdot \overline{\deg}_{*\ {}^{T'}_{\alpha}\ *} \\
&= |\mathcal{R}| \cdot \left(1 - \mathrm{P}(\textstyle\bigcup_{\ell \in L} v{:}\ell)\right) \cdot \overline{\deg}_{*\ {}^{T'}_{\alpha}\ *}.
\end{aligned}$$

New label probabilities. For the newly introduced variable v' and all labels $\ell \in L$ we have:

$$\mathrm{P}[\varepsilon_{v\ \alpha\ v'}^{r:T'}(\mathcal{R})](v':\ell) = \frac{\left|\sigma_{v':\ell}(\varepsilon_{v\ \alpha\ v'}^{r:T'}(\mathcal{R}))\right|}{\left|\varepsilon_{v\ \alpha\ v'}^{r:T'}(\mathcal{R})\right|}$$

The denominator is the result of Equation 3, while the numerator can be estimated analogously, i.e.,

$$\begin{aligned}
&\left|\sigma_{v':\ell}\left(\varepsilon_{v\ \alpha\ v'}^{r:T'}(\mathcal{R})\right)\right| \\
&= \left|\sigma_{v':\ell}\left(\varepsilon_{v\ \alpha\ v'}^{r:T'}\left(\textstyle\bigcup_{\ell' \in L}\sigma_{v:\ell'}(\mathcal{R})\right)\right)\right| \\
&\quad + \left|\sigma_{v':\ell}\left(\varepsilon_{v\ \alpha\ v'}^{r:T'}\left(\mathcal{R} \setminus \textstyle\bigcup_{\ell' \in L}\sigma_{v:\ell'}(\mathcal{R})\right)\right)\right| \\
&\overset{(1)}{\approx} |\mathcal{R}| \cdot \Big(\sum_{i=1}^{|D_L|}\sum_{j=1}^{|D_i|} \mathrm{P}(v{:}\ell_{ij} \cap \textstyle\bigcap_{k=1}^{j-1}\overline{v{:}\ell_{ik}}) \cdot \overline{\deg}_{\ell_{ij}\ \alpha\ \ell}^{\ T'} \\
&\qquad + \left(1 - \mathrm{P}(\textstyle\bigcup_{\ell' \in L} v{:}\ell')\right) \cdot \overline{\deg}_{*\ \alpha\ \ell}^{\ T'} \Big)
\end{aligned}$$

Finally, for node variable v and all labels $\ell \in L$ we have

$$\mathrm{P}[\varepsilon_{v\ \alpha\ v'}^{r:T'}(\mathcal{R})](v{:}\ell) = \frac{\left|\sigma_{v:\ell}\left(\varepsilon_{v\ \alpha\ v'}^{r:T'}(\mathcal{R})\right)\right|}{\left|\varepsilon_{v\ \alpha\ v'}^{r:T'}(\mathcal{R})\right|} = \frac{\left|\varepsilon_{v\ \alpha\ v'}^{r:T'}\left(\sigma_{v:\ell}(\mathcal{R})\right)\right|}{\left|\varepsilon_{v\ \alpha\ v'}^{r:T'}(\mathcal{R})\right|}$$

which can be computed in a similar fashion to Equation 4.

5.5 Estimation for MergeOn

To estimate $\triangledown_{v=v'}(\mathcal{R})$, i.e., how many matches of an input $\mathcal{R}$ have the same graph node bound to two distinct node variables v and v', we assign each labeled node bound to v or v' in the input to a single most representative label to estimate its contribution to the overall result cardinality only once.

Result cardinality. The MergeOn operator can be interpreted as a *filter* on the mappings of the input, retaining only those that map the same node of the graph to both specified node variables. The result cardinality can therefore never increase and the task is to estimate the reduction factor. To this end, we first split labeled and unlabeled nodes into two separate summands.

$$\begin{aligned}
|\triangledown_{v=v'}(\mathcal{R})| &= \left|\{\,\mu \mid \mu \in \mathrm{res}(\mathcal{R}) \wedge \mu(v) = \mu(v')\,\}\right| \\
&= \overbrace{\left|\{\,\mu \mid \mu \in \mathrm{res}(\mathcal{R}) \wedge \mu(v){=}\mu(v') \wedge \lambda(\mu(v)){\neq}\emptyset\}\right|}^{C_L} \\
&\quad + \underbrace{\left|\{\,\mu \mid \mu \in \mathrm{res}(\mathcal{R}) \wedge \mu(v){=}\mu(v') \wedge \lambda(\mu(v)){=}\emptyset\}\right|}_{C_{\overline{L}}}
\end{aligned}$$

We begin with the first summand C_L. To determine representative labels, all labels $\ell \in L$ are ranked according to the formula described in the estimation for Expand, but this time by computing the maximum of the probabilities from both node variables. Accordingly, we rewrite C_L as a sum over the clusters D_i of D_L.

$$\begin{aligned}
C_L &= \sum_{i=1}^{|D_L|} \left|\{\mu | \mu \in \mathrm{res}(\mathcal{R}) \wedge \mu(v) = \mu(v') \wedge \lambda(\mu(v)) \subseteq D_i\}\right| \\
&= \sum_{i=1}^{|D_L|} \left|\textstyle\bigcup_{i=j}^{|D_i|} \{\mu | \mu \in \mathrm{res}(\mathcal{R}) \wedge \mu(v) = \mu(v') \wedge \ell_{ij} \in \lambda(\mu(v))\}\right|
\end{aligned}$$

Next the nodes in each D_i are partitioned by representative label.

$$\begin{aligned}
&= \sum_{i=1}^{|D_L|}\sum_{j=1}^{|D_i|} |\{\mu | \mu \in \mathrm{res}(\mathcal{R}) \wedge \mu(v) = \mu(v') \wedge \ell_{ij} \in \lambda(\mu(v)) \\
&\qquad\qquad \wedge \nexists k \in \{1..., j-1\} : \ell_{ik} \in \lambda(\mu(v))\}|
\end{aligned}$$

Let $\mathrm{P}[\mathcal{R}](a \equiv b)$ denote the probability that $\mu(v) = \mu(v')$ for any $\mu \in \mathrm{res}(\mathcal{R})$. We can express the cardinality of the set in terms of the cardinality of $\mathcal{R}$ and the probability that its resulting mappings satisfy the given conditions.

$$\begin{aligned}
&= \sum_{i=1}^{|D_L|}\sum_{j=1}^{|D_i|} |\mathcal{R}| \cdot \mathrm{P}(v \equiv v' \cap \overbrace{v{:}\ell_{ij} \cap \textstyle\bigcap_{\ell \in L_i^{j-1}} \overline{v{:}\ell}}^{R_{ij}(v)}) \\
&= |\mathcal{R}| \sum_{i=1}^{|D_L|}\sum_{j=1}^{|D_i|} \mathrm{P}(v \equiv v' \mid R_{ij}(v)) \cdot \mathrm{P}(R_{ij}(v))
\end{aligned}$$

$\mathrm{P}(R_{ij}(v))$ is the probability of the node bound to v having representative label ℓ_{ij} and can be estimated directly using Equation 5.

It remains to estimate the conditional probability of how likely it is for any node bound to v that has representative label ℓ_{ij} to match with some node bound to v'. If v and v' map to the same node, $\mu(v')$ must also have most representative label ℓ_{ij}. Assuming statistical independence between mappings of v and v', we have

$$\begin{aligned}
\mathrm{P}(v \equiv v' \mid R_{ij}(v)) &= \mathrm{P}(v \equiv v' \cap R_{ij}(v') \mid R_{ij}(v)) \\
&= \frac{\mathrm{P}(v \equiv v' \cap R_{ij}(v) \cap R_{ij}(v'))}{\mathrm{P}(R_{ij}(v))} \\
&= \frac{\mathrm{P}(v \equiv v' \cap R_{ij}(v) \cap R_{ij}(v')) \cdot \mathrm{P}(R_{ij}(v'))}{\mathrm{P}(R_{ij}(v)) \cdot \mathrm{P}(R_{ij}(v'))} \\
&\approx \frac{\mathrm{P}(v \equiv v' \cap R_{ij}(v) \cap R_{ij}(v'))}{\mathrm{P}(R_{ij}(v) \cap R_{ij}(v'))} \cdot \mathrm{P}(R_{ij}(v')) \\
&= \mathrm{P}(v \equiv v' \mid R_{ij}(v) \cap R_{ij}(v')) \cdot \mathrm{P}(R_{ij}(v')) \\
&\approx \frac{1}{NC(\ell_{ij})} \cdot \mathrm{P}(R_{ij}(v'))
\end{aligned}$$

As we do not have statistics on how many unlabelled nodes exist in the database, we proceed similar to the Expand operator above and estimate the second summand $C_{\overline{L}}$ as follows.

$$C_{\overline{L}} \approx \frac{|\mathcal{R}|}{NC(*)} \cdot \left(1 - \mathrm{P}\left(\textstyle\bigcup_{\ell \in L} v{:}\ell\right)\right) \cdot \left(1 - \mathrm{P}\left(\textstyle\bigcup_{\ell \in L} v'{:}\ell\right)\right)$$

New label probabilities. Since the result of $\triangledown_{v=v'}(\mathcal{R})$ does not contain v' any more, we only have to estimate new label probabilities for the remaining node variable v. We also know that for every label $\ell \in L$ the probability $\mathrm{P}[\triangledown_{v=v'}(\mathcal{R})](v{:}\ell)$ can only be greater than zero if there are both nodes bound to v and v' in $\mathcal{R}$ that have label ℓ, otherwise no matching pair of bound nodes could be found. We therefore estimate the new label probabilities as the minimum among the probabilities for both variables v and v' in the input, scaled by the inverse of the reduction factor.

$$\mathrm{P}[\triangledown_{v=v'}(\mathcal{R})](v{:}\ell) \approx \frac{\min(\mathrm{P}[\mathcal{R}](v{:}\ell), \mathrm{P}[\mathcal{R}](v'{:}\ell)) \cdot |\mathcal{R}|}{|\triangledown_{v=v'}(\mathcal{R})|}$$

6 EXPERIMENTAL EVALUATION

In our experimental evaluation, we first examine the individual contributions of label probability propagation, label hierarchy and label disjointness on the overall accuracy of our technique to identify the best configuration (Section 6.1). Then, we study the accuracy and efficiency of our proposed cardinality estimation technique w.r.t. state-of-the-art approaches (Section 6.2). In terms of accuracy, we report the q-error [22], while in terms of efficiency, we measure the average runtime of each algorithm and the memory requirements for the statistical information.

Data Sets. We evaluate our technique on one synthetic data set, i.e., the LDBC Social Network Benchmark 0.2.8 (SNB) with a scale factor of 0.1, and two real-world data sets, i.e., the Cineasts 2.1.6 movie data set [12], and DBpedia 3.6 [6] that contains structured data extracted from Wikipedia. For SNB and Cineasts, we constructed the optional statistical information H_L and D_L manually, This is easily accomplished as the number of labels and, therefore, the number of queries to verify the information is very small. For DBpedia we extracted this information from the ontology provided together with the data set. DBpedia's label partition only contains one component since almost all nodes have the common label Thing. Table 1 shows the characteristics of our data sets.

State-of-the-Art Techniques. We compare our approach to three state-of-the-art techniques that were included in a recent study of cardinality estimation for subgraph matching [27]. The authors of that study classify techniques into two broad categories based on whether they were developed for graph or relational data. From the former category we include *Characteristic Sets* (CSets) [24] and SumRDF [35] in our evaluation, which were both proposed in the context of SPARQL query processing. For CSets we use our own implementation without compressing the collected statistics in order to maximize accuracy. For SumRDF, we use the authors' implementation [34] and we optimize for performance by compressing with default parameters and the smallest target size that results in a usable summary. From the latter category, we adapted the implementation of *Wander Join* (WJ) [19] provided by Park et al. [27], which can be configured to trade off accuracy and efficiency by varying the number of sampling attempts. WJ-1 and WJ-100 denote configurations with 1 and 100 attempts, respectively, whereas WJ-R denotes the configuration used by Park et al. [27]. We also compare our approach to the cardinality estimator of Neo4j.

Most of the state-of-the-art approaches come with limitations. CSets, WJ, and SumRDF do not support undirected or untyped relationships. WJ also does not support patterns that have nodes with more than one label or properties. Furthermore, while all competing techniques except for Neo4j assume homomorphism semantics, we use them under Cyphermorphism semantics.

Query Sets. To generate a variety of graph patterns of different sizes without introducing subjective bias, we first match undirected patterns generated from graphlets with 3–7 nodes [31] against the data set, anchored at randomly selected nodes. The resulting subgraphs are then transformed into more general patterns by randomly removing some of their information, e.g., labels, properties or directedness of relationships. Following the work of Bonifati et al. [5] and Park et al. [27], we classify the resulting patterns into four main categories of shapes: *chains*, *stars*, *trees*, and *cyclic*. The cyclic patterns are subdivided into *circles*, *petals*, *flowers*, and *other* cyclic patterns. We then apply multiple rounds of stratified sampling to obtain a set of patterns that is diverse w.r.t. shape, size and frequency of their components (nodes, labels, properties etc.). We generate two query sets per data set, one with patterns that contain up to three properties and one with no properties present. The first set was generated in such a way that all methods but WJ support all queries (patterns containing nodes with multiple labels are included). Table 2 gives the total number of generated query patterns.

Environment. All experiments were conducted on a server-grade machine with a 12-core Intel Xeon E5-2650 v4 CPU and 256 GiB 2666 MHz DDR4 memory, running Windows 10 Pro 20H2. Our source code is available under MIT license [41].

Table 1: Data sets

	SNB	**Cineasts**	**DBpedia**
nodes	327,588	72,542	2,362,452
relationships	1,477,965	106,651	7,046,160
properties	2,132,911	561,520	8,344,029
node labels	14	5	252
rel. types	15	4	584
prop. keys	23	27	601
H_L height	2	2	5
D_L components	7	2	1

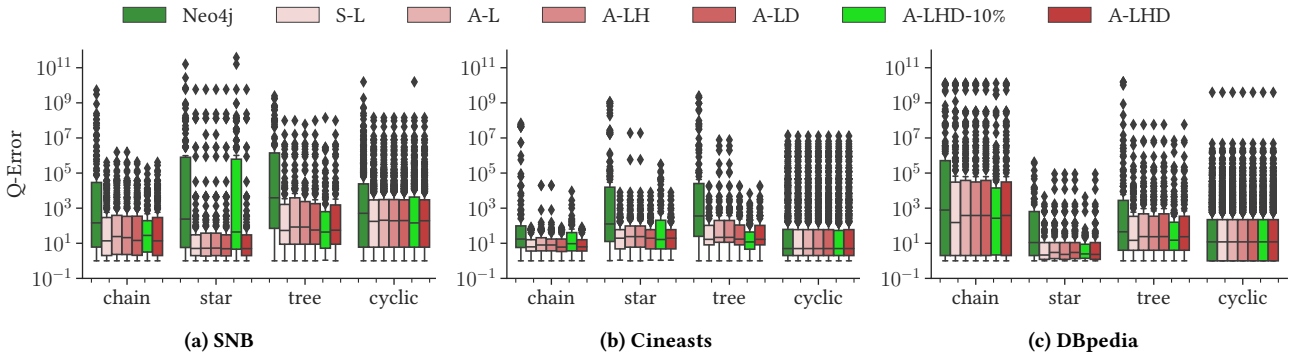
Table 2: Number of patterns in the six query sets

	Chain	**Star**	**Tree**	**Cyclic**			
				Circle	Petal	Flower	*other*
SNB	182	148	130	154	151	176	189
(w/o prop)	830	652	411	94	131	107	442
Cineasts	155	161	146	148	163	163	170
(w/ prop)	270	270	267	71	153	35	11
DBpedia	175	145	169	154	199	234	305
(w/o prop)	119	156	141	186	143	161	184

6.1 Detailed Evaluation of Our Technique

We first examine how the different types of statistical information described in Section 4 affect the accuracy of our approach. We label different configurations of our technique as follows. First, we indicate whether Neo4j's **S**imple or our more **A**dvanced statistics are used for relationship counts (Section 4.1). Then we denote whether **L**abel probability propagation (Section 5) and information about **H**ierarchical or **D**isjoint labels is used (Section 4.2). For example, A-LHD is the configuration that uses label probability propagation together with both required and all of the optional statistical information. All configurations use property statistics except for A-LHD-10% which, similar to many real-world systems including Neo4j, follows the classical approach to assume 10% selectivity for all properties [33]. As a point of reference, we include the results of the cardinality estimator of Neo4j, which is our most direct competitor as it is the only other technique specifically designed for property graphs and supports all graph patterns in our query sets.

Figure 5 shows the accuracy (q-error) of six different configurations of our technique and Neo4j for subgraph matching queries with chain, tree, star, and cyclic graph patterns. For the synthetic SNB data set (Figure 5a), all configurations of our technique consistently outperform Neo4j, even if they rely on the same simple statistical information (S-L). Furthermore, we observe that the more advanced statistical information either improves cardinality estimates or does not affect them at all. For cyclic patterns our estimates are generally less accurate. Because our technique applies the independence assumption more frequently when MergeOn is involved, underestimation is more pronounced, masking some of the effects that can be observed in the other types of patterns. With regard to the optional statistical information, D_L improves estimation accuracy, H_L has almost no effect, and the information on properties improves estimation accuracy only for star patterns.



(a) SNB (b) Cineasts (c) DBpedia

Figure 5: Accuracy of different configurations of our cardinality estimation technique in comparison with Neo4j

The results on the real-world data sets in Figures 5b and 5c clearly show that all configurations of our technique outperform Neo4j in all cases, apart from cyclic patterns where the accuracy is essentially the same. For Cineasts, estimation accuracy for cyclic patterns is better than on SNB. This can be explained by the more restricted structure of the Cineasts graph, which contains very few triangles in comparison to SNB. Therefore, the cardinality of cyclic pattern queries is low, bounding the q-error. For DBpedia, we observe that S-L and A-LHD-10%, which tend to overestimate cardinalities, demonstrate higher accuracy. This behavior can be explained by the fact that the DBpedia data set contains a large number of distinct concepts. As a result, the use of the independence assumption introduces strong underestimation. With regard to optional information, we observe that label disjointness information improves estimation accuracy in Cineasts, while label hierarchy information seems to have no effect in either data set. Comparing A-LHD and A-LHD-10%, the information about properties seems to benefit cardinality estimation only for chain and star patterns in Cineasts, while in DBpedia the accuracy is getting worse.

Insights. The label probability propagation technique proposed in this paper avoids the sometimes significant underestimations made by Neo4j, even if it uses the same statistical information. Also, collecting and using more advanced statistical information leads to more accurate cardinality estimates provided that the structure of the underlying data is not overly complex. The impact of statistics designed to capture the (implicit) schema of the property graph, i.e., information about label hierarchies (H_L) and label disjointness (D_L), highly depends on the structure of the graph and the availability of such information. Between these two types of optional information, D_L can improve accuracy substantially, while the effect of H_L is generally more minor. With regard to the optional information about properties, we observe a large benefit in patterns in which our approach can effectively limit underestimations, e.g., chain and star patterns in SNB and Cineasts data sets.

6.2 Comparison to the State of the Art

We now study how our approach compares to the selected state-of-the-art approaches w.r.t. accuracy and efficiency.

Runtime and Memory Requirements. Figure 6 shows the runtime of the cardinality estimation approaches. We present the results only on the query set with properties, since all methods but WJ support all queries. Also, due to lack of space, we only present the results on the SNB dataset. We excluded Neo4j from

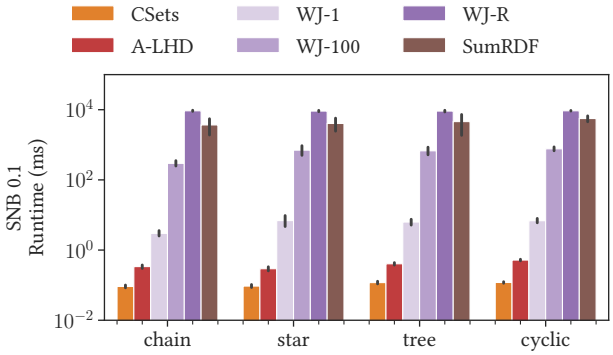


Figure 6: Runtime for patterns with properties (SNB)

the evaluation as we could not isolate the runtime for the cardinality estimation. However, based on our measurements, we expect Neo4j to be the fastest technique. Also, due to their very large runtime, we set a timeout of 10 seconds for SumRDF and for WJ-R. Our results in Figure 6 show that CSets is the fastest technique. Our A-LHD technique is slower than CSets but still offers comparable runtime performance. WJ-1 comes third, while all other techniques, i.e., SumRDF, WJ-100, and WJ-R are at least one order of magnitude slower than our technique.

Table 3 reports the (approximate) memory requirements of all cardinality estimation techniques. Neo4j inflicts the smallest memory overhead in all data sets. Our A-LHD approach (with and without the optional statistical information on properties) has slightly greater memory requirements than Neo4j being second in Cineasts and DBpedia, and third in SNB, behind CSets. SumRDF has clearly the highest memory requirements while WJ does not require any stored information by definition. Note that CSets and SumRDF can be configured to have different memory requirements. However, no configuration of these approaches dominates our A-LHD. On the one hand, CSets can be tuned to require less memory, but then its accuracy will drop even further increasing the gap from A-LHD. On the other hand, SumRDF is more accurate with less compression, but then it will also become even slower and consume more memory.

Pattern Size. Figure 7 shows the results of our experiments studying the impact of *pattern size*, i.e., the sum of the number of labels, relationships, and properties in the pattern, on the q-error. Since queries with properties usually have much smaller result cardinality than queries without properties, which is expected to affect accuracy, we present the results for both query sets with

Table 3: (Approximate) Sizes of summaries

	CSets	Neo4j	A-LHD		WJ	SumRDF
			basic	+props		
SNB	1.8 kB	1.2 kB	3.1 kB	13.3 kB	n/a	192.1 MB
Cineasts	7.9 kB	0.3 kB	0.8 kB	7.7 kB	n/a	4.9 MB
DBpedia	10.3 MB	0.2 MB	1.4 MB	1.9 MB	n/a	291.3 MB

and without properties (see Table 2). For the query sets with properties, all methods support all queries, apart from WJ which is excluded. For the query sets without properties, only our A-LHD approach and Neo4j support all queries; CSets supports 48% of the queries on SNB, 23% of the queries on Cineasts, and 38% of the queries on DBpedia; WJ supports approximately 28% of the queries on SNB, 17% of the queries on Cineasts, and 17% of the queries on DBpedia; and SumRDF supports approximately 16% of the queries on SNB, 23% of the queries on Cineasts, and 37% of the queries on DBpedia.

Figure 7a presents the q-error of all approaches on the query sets that contain patterns with properties. For the SNB and Cineasts data sets, we observe a trend that the q-error increases with the size of the query pattern for all techniques. In these two datasets, our A-LHD approach clearly outperforms CSets and Neo4j. For SNB in particular, A-LHD offers comparable accuracy to SumRDF. For the DBpedia data set the pattern size does not seem to affect the q-error at all. We also observe that all methods offer comparable accuracy. This result can be explained by the fact that due to the structure of the DBpedia data set all methods tend to underestimate greatly.

In Figure 7b, we present the results of our experiments on the query sets that contain patterns without properties. For the SNB and Cineasts data sets, we observe a much clearer trend for all approaches that the q-error increases with the pattern size than on the query set with properties. In contrast, similar to the previous query set, for DBpedia we do not observe such a connection between pattern size and q-error. With regard to our technique, A-LHD outperforms CSets and Neo4j in all cases, and is comparable to SumRDF for small pattern sizes. The results for WJ show the expected behavior: increasing the sampling rate increases accuracy. Our approach offers higher accuracy than WJ-1 in all cases, except for very large patterns in DBpedia. In comparison to WJ-R and WJ-100 in SNB, our approach demonstrates slightly lower accuracy. Nevertheless, our technique outperforms CSets, Neo4j, and WJ in almost all cases, while SumRDF achieves clearly the best accuracy.

In Figure 7 we also observe that the presence of properties in queries has a great effect on the accuracy of all methods. Since the result cardinality of queries with properties is usually low, drastically underestimating cardinalities leads to lower q-error. Furthermore, the difference in accuracy between competitors is less pronounced on queries with properties than without. As the existence of properties in patterns affects the estimations greatly and not all techniques support patterns with properties, to gain additional insight on the performance of the cardinality estimation methods, in what follows we focus on patterns without properties.

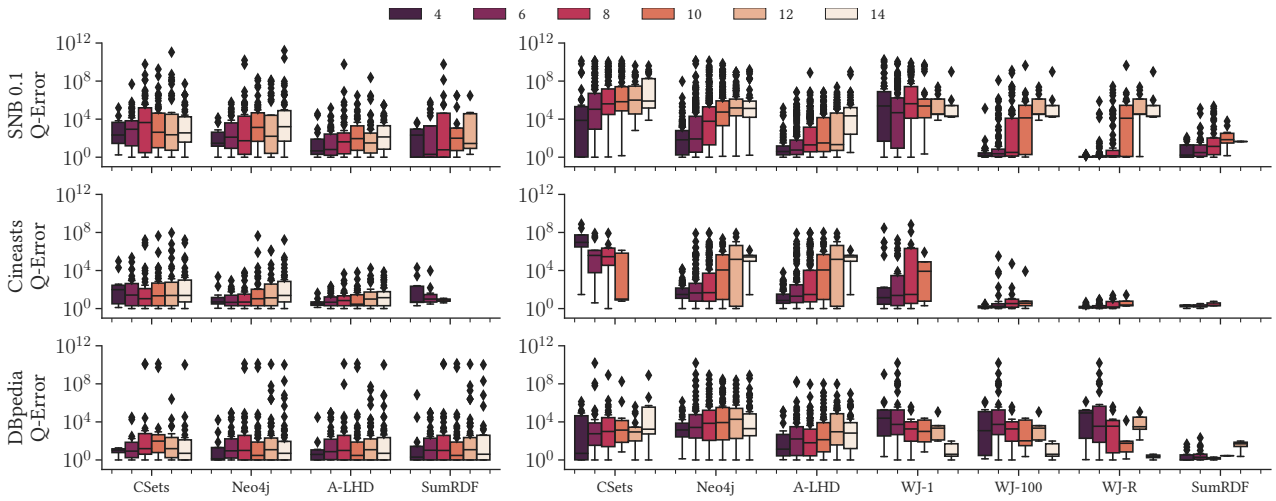
Detailed Evaluation on Patterns without Properties. Figure 8a shows the results of our experiments studying the impact of pattern shape on the q-error. For SNB, we observe that our technique (A-LHD) consistently outperforms CSets and Neo4j, while it shows similar accuracy as SumRDF, albeit with more pronounced outliers. For WJ we observe once again that the sampling rate increases accuracy. We note that overall cyclic patterns present the greatest challenge to cardinality estimation for most approaches. For the smaller Cineasts data set, we observe the same but less pronounced results. The results of WJ-1 and WJ-100 are slightly better in this data set due to the fact that their relative sampling rate increases as the size of the data set decreases. For DBpedia, SumRDF offers the highest accuracy. Our A-LHD technique comes second behind SumRDF, outperforming all remaining techniques.

Figure 8b reports on the effect that the label density $d = \frac{\#\text{labels}}{\#\text{nodes}}$ has on cardinality estimation errors. We classify label density into three categories: *low* ($0 \leq d \leq 0.\overline{3}$), *medium* ($0.\overline{3} < d \leq 0.\overline{6}$), and *high* ($0.\overline{6} < d$). The results for both data sets confirm the observations made in the previous experiments: A-LHD consistently outperforms CSets as well as Neo4j and, on average, is comparable to SumRDF. Again, WJ performs as expected: accuracy is increasing with the number of samples. In general, we observe that the label density has no clear effect on the q-error.

Figure 8c gives the results of our study into the accuracy of cardinality estimation w.r.t. true cardinality, i.e., the result size of the corresponding query. We group results into buckets according to orders of magnitude: 1–99, 100–9,999, 10,000–999,999, etc. For all data sets, our A-LHD technique outperforms CSets and Neo4j for all result sizes. While the accuracy of CSets and Neo4j degrades with increasing result size, the accuracy of our technique and that of SumRDF seems to be unaffected. WJ exhibits similar behavior to the previous experiments, with the exception of SNB where WJ-100 and WJ-R show an increased q-error for small patterns.

Homomorphism vs. Cyphermorphism Semantics. While in this paper we focus on cyphermorphism semantics, some of the approaches we evaluate, i.e., CSets and SumRDF, have been designed for SPARQL, which uses homomorphism semantics instead. This means that any node or relationship in the subgraph may be mapped to more than one node or relationship in the pattern. In order to evaluate the effect that the difference in semantics has on the quality of the estimations, we compared the ground truth cardinalities for all queries in our query sets using both semantics. We observed that in all data sets, for most queries the cardinality computed using homomorphism semantics did not exceed the cardinality computed using cyphermorphism semantics by a factor larger than 1.5, with only very few patterns demonstrating an increase by a factor larger than 10. As the q-error is a ratio between true and estimated cardinality, the factor by which the true cardinality changes is an upper bound for the change in the resulting q-error. Since the differences between the studied techniques exceed the observed factor by which the cardinalities differ, we do not expect the semantics to alter the ranking of the techniques.

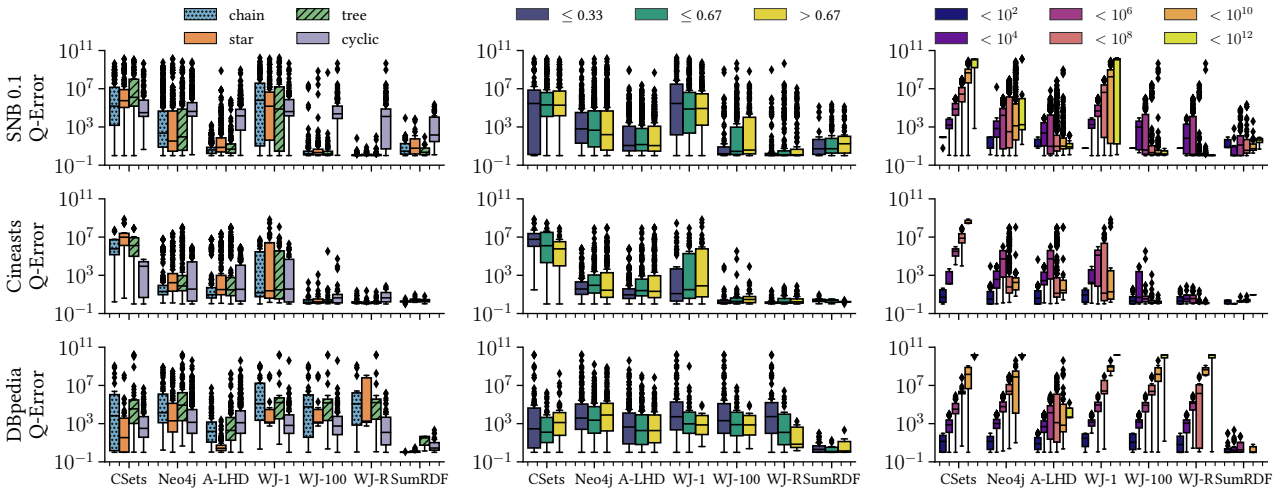
Insights. The results presented in this section show that our approach is considerably faster than techniques that are more accurate, and considerably more accurate than techniques that are faster. More specifically, our approach consistently outperforms CSets and Neo4j, i.e., the only two techniques proposed for graph query optimization. Additionally, our approach is, on average, comparable to SumRDF in terms of accuracy, while requiring only a fraction of its runtime and memory. Furthermore,



(a) Query sets with properties

(b) Query sets without properties

Figure 7: Accuracy for query patterns with and without properties varying pattern size



(a) Pattern shape

(b) Label density

(c) Result size

Figure 8: Accuracy for different pattern shapes

our approach provides a better trade-off between accuracy and efficiency than WJ, being consistently faster and more accurate than the fastest WJ configuration, i.e., WJ-1, and significantly faster with comparable accuracy to the most accurate configurations, i.e., WJ-100 and WJ-R. Finally, we note that our experimental results are consistent with the findings of Park et al. [27] w.r.t. the state-of-the-art techniques.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented label probability propagation, a novel cardinality estimation technique for subgraph matching queries in property graphs. We have also studied how different types of required and optional statistical information w.r.t. node labels, relationship types and properties impact estimation error. Through a comprehensive experimental evaluation we have shown that, to the best of our knowledge, no state-of-the-art technique exists that dominates our technique in terms of the trade-off between accuracy and efficiency. Our results also show that our technique delivers the trade-off between accuracy and efficiency required in the setting of query optimization. In the future, we plan to study the effect of the operator sequence order on estimation quality in depth. We also plan to investigate whether more sophisticated graph statistics, e.g., triangle counts, etc., would benefit the accuracy of our technique. Last but not least, we plan to extend our approach to support more types of graph query patterns such as patterns with optional subpatterns and variable length paths.

ACKNOWLEDGMENTS

This work is partially supported by Grant No. GR 4497/2-1 and Grant No. CH 2464/1-1 of the Deutsche Forschungsgemeinschaft (DFG). Part of this work is based on the BSc thesis of Moritz Renftle at the University of Konstanz under the supervision of Jürgen Hölsch.

REFERENCES

[1] Oracle and/or its affiliates. 2021. Property Graph Query Language. https://pgql-lang.org.

[2] Renzo Angles. 2018. The Property Graph Database Model. In *Proc. Alberto Mendelzon Intl. Workshop On Foundations Of Data Management (AMW)*. http://ceur-ws.org/Vol-2100/paper26.pdf

[3] Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. 2018. G-CORE: A Core for Future Graph Query Languages. In *Proc. SIGMOD*. 1421–1432. https://doi.org/10.1145/3183713.3190654

[4] Angela Bonifati, George Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. *Querying Graphs*. Morgan & Claypool Publishers, Chapter Cardinality Estimation, 128–135. https://doi.org/10.2200/S00873ED1V01Y201808DTM051

[5] Angela Bonifati, Wim Martens, and Thomas Timm. 2020. An analytical study of large SPARQL query logs. *The VLDB Journal* 29, 2 (2020), 655–679. https://doi.org/10.1007/s00778-019-00558-9

[6] DBpedia Association. 2021. DBpedia Datasets. http://wikidata.dbpedia.org/develop/datasets. Accessed: 2021-09-28.

[7] Stefania Dumbrava, Angela Bonifati, Amaia Nazabal Ruiz Diaz, and Romain Vuillemot. 2019. Approximate querying on property graphs. In *Proc. Intl. Conf. on Scalable Uncertainty Management (SUM)*. 250–265. https://doi.org/10.1007/978-3-030-35514-2_19

[8] Graph Query Language GQL. 2020. GQL Standard. https://www.gqlstandards.org/.

[9] Andrey Gubichev. 2015. *Query Processing and Optimization in Graph Databases*. Ph. D. Dissertation. Technische Universität München. http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20150625-1238730-1-7

[10] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. 2010. Data summaries for on-demand queries over linked data. In *Proc. WWW*. 411–420. https://doi.org/10.1145/1772690.1772733

[11] Jürgen Hölsch and Michael Grossniklaus. 2016. An Algebra and Equivalences to Transform Graph Patterns in Neo4j. In *Proc. EDBT/ICDT Workshops*. http://nbn-resolving.de/urn:nbn:de:bsz:352-0-326779

[12] Neo4j Inc. 2014. Cineasts. http://example-data.neo4j.org/files/cineasts_12k_movies_50k_actors_2.1.6.zip. Accessed: 2021-09-28.

[13] Neo4j Inc. 2021. openCypher. https://opencypher.org.

[14] Yannis Ioannidis. 2003. The History of Histograms (abridged). In *Proc. VLDB*. 19–30. https://doi.org/10.1016/B978-012722442-8/50011-2

[15] Yannis E. Ioannidis and Stavros Christodoulakis. 1991. On the Propagation of Errors in the Size of Join Results. In *Proc. SIGMOD*. 268–277. https://doi.org/10.1145/115790.115835

[16] Sebastian Kruse and Felix Naumann. 2018. Efficient Discovery of Approximate Dependencies. *PVLDB Endow.* 11, 7 (2018), 759–772. https://doi.org/10.14778/3192965.3192968

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

[18] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *PVLDB Endow.* 9, 3 (2015), 204–215. https://doi.org/10.14778/2850583.2850594

[19] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation for Joins. In *Proc. SIGMOD*. 2121–2124. https://doi.org/10.1145/2882903.2899413

[20] Cheng Luo, Zhewei Jiang, Wen-Chi Hou, Feng Yu, and Qiang Zhu. 2009. A sampling approach for XML query selectivity estimation. In *Proc. EDBT*. 335–344. https://doi.org/10.1145/1516360.1516400

[21] Angela Maduko, Kemafor Anyanwu, Amit Sheth, and Paul Schliekelman. 2008. Graph summaries for subgraph frequency estimation. In *Proc. ESWC*. 508–523. https://doi.org/10.1007/978-3-540-68234-9_38

[22] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *PVLDB Endow.* 2, 1 (2009), 982–993. https://doi.org/10.14778/1687627.1687738

[23] Magnus Müller, Guido Moerkotte, and Oliver Kolb. 2018. Improved Selectivity Estimation by Combining Knowledge from Sampling and Synopses. *PVLDB Endow.* 11, 9 (2018), 1016–1028. https://doi.org/10.14778/3213880.3213882

[24] Thomas Neumann and Guido Moerkotte. 2011. Characteristic Sets: Accurate Cardinality Estimation for RDF Queries with Multiple Joins. In *Proc. ICDE*. 984–994. https://doi.org/10.1109/ICDE.2011.5767868

[25] Thomas Neumann and Gerhard Weikum. 2008. RDF-3X: a RISC-style engine for RDF. *PVLDB Endow.* 1, 1 (2008), 647–659. https://doi.org/10.14778/1453856.1453927

[26] Marcus Paradies, Elena Vasilyeva, Adrian Mocan, and Wolfgang Lehner. 2015. Robust Cardinality Estimation for Subgraph Isomorphism Queries on Property Graphs. In *Proc. VLDB Workshops*. 184–198. https://doi.org/10.1007/978-3-319-41576-5_14

[27] Yeonsu Park, Seongyun Ko, Sourav S. Bhowmick, Kyoungmin Kim, Kijae Hong, and Wook-Shin Han. 2020. G-CARE: A Framework for Performance Benchmarking of Cardinality Estimation Techniques for Subgraph Matching. In *Proc. SIGMOD*. 1099–1114. https://doi.org/10.1145/3318464.3389702

[28] Neoklis Polyzotis, Minos Garofalakis, and Yannis Ioannidis. 2004. Approximate XML query answers. In *Proc. SIGMOD*. 263–274. https://doi.org/10.1145/1007568.1007599

[29] Viswanath Poosala and Yannis E. Ioannidis. 1997. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proc. VLDB*. 486–495.

[30] PostgreSQL 13 Documentation. 2021. 14.2.1. Single-Column Statistics. https://www.postgresql.org/docs/13/planner-stats.html#id-1.5.13.5.3m. Accessed: 2021-09-27.

[31] Nataša Pržulj, Derek G. Corneil, and Igor Jurisica. 2004. Modeling interactome: scale-free or geometric? *Bioinformatics* 20, 18 (2004), 3508–3515. https://doi.org/10.1093/bioinformatics/bth436

[32] 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, 2-3 (2020), 595–618. https://doi.org/10.1007/s00778-019-00548-x

[33] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. 1979. Access Path Selection in a Relational Database Management System. In *Proc. SIGMOD*. 23–34. https://doi.org/10.1145/582095.582099

[34] Giorgio Stefanoni, Boris Motik, and Egor V. Kostylev. 2017. ISG Tools: SumRDF. https://www.cs.ox.ac.uk/isg/tools/SumRDF/. Accessed: 2021-09-28.

[35] Giorgio Stefanoni, Boris Motik, and Egor V. Kostylev. 2018. Estimating the Cardinality of Conjunctive Queries over RDF Data Using Graph Summarisation. In *Proc. WWW*. 1043–1052. https://doi.org/10.1145/3178876.3186003

[36] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. 2008. SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation. In *Proc. WWW*. 595–604. https://doi.org/10.1145/1367497.1367578

[37] David Vengerov, Andre Cavalheiro Menck, Mohamed Zaït, and Sunil Chakkappen. 2015. Join Size Estimation Subject to Filter Conditions. *PVLDB Endow.* 8, 12 (2015), 1530–1541. https://doi.org/10.14778/2824032.2824051

[38] W3C. 2008. SPARQL 1.1 Overview. https://www.w3.org/TR/sparql11-overview/.

[39] Li Wang. 2017. *On histograms for path selectivity estimation in graph data*. Master's thesis. Eindhoven University of Technology.

[40] Andreas M. Weiner. 2011. Advanced Cardinality Estimation in the XML Query Graph Model. In *Proc. BTW*. 207–226. https://dl.gi.de/20.500.12116/19580

[41] Leonard Wörteler and Moritz Renftle. 2022. A-LHD Source Code. https://github.com/DBIS-UniKN/graph-card-label-probs. MIT license.

[42] Nikolay Yakovets. 2016. *Optimization of Regular Path Queries in Graph Databases*. Ph. D. Dissertation. York University. http://hdl.handle.net/10315/33392

[43] Nikolay Yakovets, Parke Godfrey, and Jarek Gryz. 2016. Query planning for evaluating SPARQL property paths. In *Proc. SIGMOD*. 1875–1889. https://doi.org/10.1145/2882903.2882944

[44] Nikolay Yakovets, Li Wang, George Fletcher, Craig Taverner, and Alexandra Poulovassilis. 2018. Histogram Domain Ordering for Path Selectivity Estimation. In *Proc. EDBT*. 493–496. https://doi.org/10.5441/002/edbt.2018.55

[45] Ning Zhang, M. Tamer Ozsu, Ashraf Aboulnaga, and Ihab F. Ilyas. 2006. XSEED: Accurate and Fast Cardinality Estimation for XPath Queries. In *Proc. ICDE*. 61–61. https://doi.org/10.1109/ICDE.2006.178

[46] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *Proc. SIGMOD*. 1525–1539. https://doi.org/10.1145/3183713.3183739