Marcus Paradies, Elena Vasilyeva, Adrian Mocan, Wolfgang Lehner

Robust Cardinality Estimation for Subgraph Isomorphism Queries on Property Graphs

Erstveröffentlichung in / First published in:

Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa, 31.08-04.09.2015. Springer, S. 184-198. ISBN 978-3-319-41576-5.

DOI: http://dx.doi.org/10.1007/978-3-319-41576-5_14

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

Robust Cardinality Estimation for Subgraph Isomorphism Queries on Property Graphs

Marcus Paradies[1,2(✉)], Elena Vasilyeva[1,2], Adrian Mocan[2], and Wolfgang Lehner[1]

[1] Database Systems Group, TU Dresden, Dresden, Germany
{m.paradies,elena.vasilyeva}@sap.com, wolfgang.lehner@tu-dresden.de
[2] SAP SE, Walldorf, Germany
adrian.mocan@sap.com

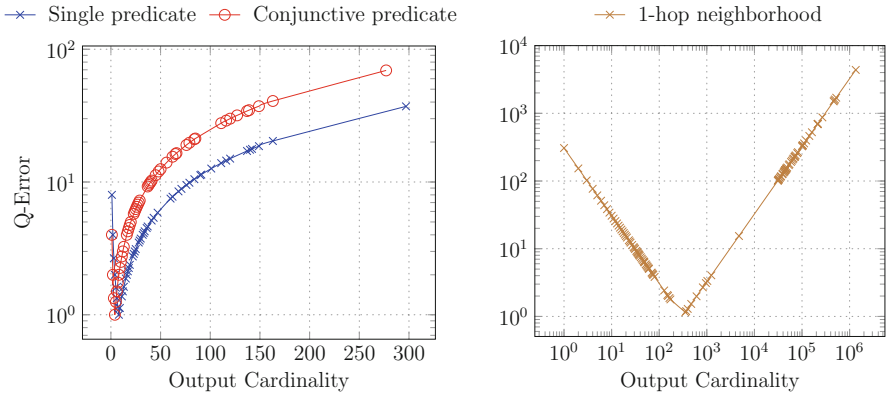
Abstract. With an increasing popularity of graph data and graph processing systems, the need of efficient graph processing and graph query optimization becomes more important. Subgraph isomorphism queries, one of the fundamental graph query types, rely on an accurate cardinality estimation of a single edge of a pattern for efficient query processing. State of the art approaches do not consider two important aspects for cardinality estimation of graph queries on property graphs: the existence of nodes with a high outdegree and functional dependencies between attributes. In this paper we focus on these two challenges and integrate the detection of high-outdegree nodes and functional dependency analysis into the cardinality estimation. We evaluate our approach on two real data sets and compare it against a state-of-the-art query optimizer for property graphs as implemented in Neo4j.

1 Introduction

The recent advent of graph-structured data and the ever-growing need to process graph data led to the development of a plethora of graph processing systems and frameworks [5]. While some of them target long-running offline, analytic graph queries like PageRank calculation and community detection, others focus on online transactional queries. A subgraph isomorphism query is one of the most fundamental graph query types represented in both graph query paradigms. It is supported in most commercial GDBMS either through a specific language such as Neo4j's declarative query language Cypher or can be emulated through a programming interface, such as in Sparksee [6]. A subgraph isomorphism query discovers all data subgraphs matching a query graph and belongs to queries with high computational costs [4]. To decrease the processing efforts, usually such queries rely on statistics about the underlying data graph and a cardinality estimation of each edge in a query graph. Therefore, to estimate a cardinality of a graph pattern, it is necessary to estimate precisely the cardinality of each single edge in a query. Query optimization based on the cardinality estimation of an edge(graph) becomes crucial for efficient subgraph isomorphism queries and adaptation of the online query processing.

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

Figure 1 shows the result of a micro benchmark evaluating the cardinality estimation quality of NEO4J[1] for the LDBC data set at scale factor 1. We conduct evaluation for simple filter operations on nodes—a predicate on *firstname* and a conjunctive predicate on the correlated attributes *firstname* and *gender*—and for 1-hop traversals based on the *knows* subgraph describing relationships between forum users. NEO4J assumes a uniform distribution of selectivities for both, attributes and neighborhoods, respectively. This results in a dramatic cardinality underestimation with a q-error of about 1000 and more for the predicate evaluation on correlated attributes and for neighborhood queries on high-outdegree nodes. It is not our intention to belittle NEO4J and its query optimizer, but instead to raise fundamental problems in dealing with real-world graphs that are prevalent in most graph database management systems.



(a) Filter query. **(b) Neighborhood query.**

Fig. 1. Output cardinality estimation with Q-error of filter evaluation on nodes and simple 1-hop neighborhood queries in Neo4j.

Although cardinality estimation for graph pattern matching on property graphs is a rather new research field, there have been extensive studies on graph pattern matching in SPARQL—the declarative query language of the RDF data model—and cardinality estimation techniques in RDFDBMS [7,8,16]. While some techniques can be directly applied to the graph pattern matching on property graphs, there are some fundamental differences caused by the different underlying data models like the RDF data model does not provide natural support for attributes on edges. Moreover, these techniques focus only on cardinality estimation of a star topology, which is a typical pattern in the RDF graphs. In opposite, property graphs can include arbitrary topologies and nodes and edges can be described by multiple attributes.

[1] We use the latest available version: Neo4j 2.3.0-M2.

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

As a single edge estimation is crucial for estimating the cardinality of a graph query, we improve it by addressing two important challenges—handling of correlated attributes with a skewed value distribution and cardinality estimation for neighborhood queries in the presence of high-outdegree nodes. We summarize our contributions as follows:

- We reuse a technique well-known from the relational world to detect soft functional dependencies between attributes and apply it to the property graph model.
- We propose *degree histograms*, a concise representation of the degree distribution with a dedicated handling of vertices with a large in/outdegree.
- We evaluate our approach on two realistic graph data sets with a rich set of attributes, correlations between attributes, and a skewed degree distribution. We show that our techniques achieve cardinality estimations that are up to a factor of 50 better than a naive solution.

In the following we present the state of the art of cardinality estimation for graphs in Sect. 2. Then we provide the general description of cardinality estimation on graphs in Sect. 3. We enhance it by considering correlated attributes in Sect. 4 and detecting high-outdegree nodes in Sect. 5. We evaluate our solution on two data sets in Sect. 6 before we conclude in Sect. 7.

2 Related Work

Most graph queries require high processing efforts caused by the flexible schema of a graph model and the complexity of graph queries themselves. The optimization of such queries is a challenging task due to the irregularity of the graph topology, the exposed attributes, and the value-based and topology-based correlations exhibited in real-world graphs. Usually graph query optimization focuses on establishing a traversal path through the query and answers the question: 'Which edge has to be processed next?' by estimating the cardinality of each candidate edge. In this section we discuss the state of the art work on cardinality estimation with the focus on graph data.

2.1 RDF Cardinality Estimation

In the graph database research community, cardinality estimation of patterns [7,8,16] is critical for RDF (Resource Description Framework) graphs, where each query can be represented as a join of several stars describing specific entities. The RDF data representation enables storing and processing of schema-free structured information. A typical pattern for RDF data is a star, therefore, it is taken as the base for query optimization.

Basic Graph Pattern (BGP) in a form of (un)bounded triples are used as a static optimization [16] to determine a join order for pattern calculation. The proposed system constructs a query plan by traversing edges according to the

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

minimum estimated selectivity principle by passing first visiting triples to prevent a system from having to compute the Cartesian product of two intermediate result sets. To increase the estimation quality, cardinality for a bounded object is calculated deterministically, while an upper bound of the size of joined triples for unbounded objects is supported by domain/range information. In comparison to our approach, the cardinality estimation of BGP does not support the reduction of intermediate results based on the neighboring relations. Moreover, the underlying model differs from the property graph model and therefore does not support modeling of attributes on edges.

Cardinality estimation for the schema-free representation is provided by frequent path calculation [8] and characteristic sets identification [7]. In both cases these statistics are precomputed and they describe two kinds of patterns specific for the RDF data representation: chains and stars. In comparison to these solutions, we work with entities and calculate their cardinalities based on the estimated schema information and additional graph-specific characteristics like in - and outdegrees and their modifications for particular edge types. In addition, we provide more precise cardinality estimation by overcoming the independence assumption.

2.2 Cardinality Estimation in DBMS

Query optimization in DBMS is a long-term established research field. Leveraging statistics about data distributions, correlations, and selectivities have been around in the DBMS market since the very beginning. Cardinality estimation is used for example for establishing an optimal join order. For this purpose, the join selectivity and selectivity of a selection operator are calculated based on the data distribution and multidimensional histograms. To increase the performance of estimation, sampling techniques are used. We refer an interesting reader to the survey [2]. In our work we focus on eliminating predicates based on functional dependencies between them [10,12]. This allows us discarding dependent predicates from the cardinality estimation and thereby increasing the quality of the estimation.

2.3 Graph Cardinality Estimation

The property graph model is a natural graph data model where vertices represent entities and edges describe relationships between them. This data model is used in modern graph processing systems, e.g. Neo4j. By estimating a graph pattern of a property graph, we need to consider statistics from DBMS and graph databases. However, for the property graph data model, only a few works [1,9,15] exist that try to apply relational/RDF results to property graphs.

In graph databases statistics natural for graphs can be based on neighborhood relationships [1,9]. In this case, as statistics we can use the complete "neighborhood" function of a graph. The exact calculation of a neighborhood function is expensive, which is why multiple approximation methods are considered.

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

These statistics can be used to determine the similarity between two graphs or to calculate the diameter of a graph.

For pattern detection, to reduce the complexity of matching, nodes and edges are filtered based on the label similarity [15]. For this purpose, the algorithm Quick-SI pre-processes a data graph and computes frequencies of vertex labels and frequencies of triples (source, edge, and target labels). The algorithm joins edges starting with the low-frequency edges.

3 Graph Data Model and Cardinality Estimation

Our system uses the property graph model [13] as an underlying data model. It represents a graph as a directed multigraph, where nodes are entities and edges are relationships between them. Each edge and node can be described by multiple types of attributes that can differ among edges and nodes—even if they have the same semantic type.

Definition 1 (Property Graph). *We define a property graph as a directed graph* $G = (V, E, u, f, g)$ *over an attribute space* $A = A_V \dot{\cup} A_E$*, where: (1)* V, E *are finite sets of nodes and edges; (2)* $u : E \to V^2$ *is a mapping between edges and nodes; (3)* $f : V \to A_V$ *and* $g : E \to A_E$ *are attribute functions for nodes and edges; and (4)* A_V *and* A_E *are their attribute spaces.*

Typical queries supported by graph databases include subgraph isomorphism queries, reachability queries, etc. Graph databases rely on accurate cardinality estimation to process such queries efficiently and usually support multiple indices specific for each query type. Before estimating the cardinality for any graph query, we must take into account the specifics of property graphs expressed by notation and topology.

3.1 Notation: Attribute Histograms

A property graph can have attributes on nodes and on edges. For each attribute we construct a frequency histogram, where the x-axis represents values of the attribute domain and the y-axis shows the number of occurrences for the specific value in the data graph. We support numerical as well as categorical attributes. An important parameter of an attribute histogram is the bucket width, which allows controlling the size of the histogram and consequently the quality of the cardinality estimation. Dividing a one-dimensional histrogram into equi-width buckets can be easily applied to numerical value domains, but is not meaningful for categorical attribute domains. To tackle this problem we apply the creation of buckets not directly on the values, but instead on the value codes that stem from a dictionary encoding. Dictionary encoding is frequently used for compressing categorical values by replacing the variable-length value with a fixed-length value code. For one-dimensional histograms with a bucket size larger than one, the value frequency describes the number of occurences of values from an interval of value codes. While increasing the interval width allows reducing the memory consumption of the histogram, it also decreases the precision of the cardinality estimation.

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

3.2 Topology: Degree Histograms

The graph topology is a unique property of a graph model. It can be represented by an in - or outdegree of a node and its neighbors, typical graph topologies like triangles, stars etc.

General node in - or outdegree describes the maximum number of its direct neighbors. For cardinality estimation we define the in- and outdegree separately for each edge type. An edge type is a special kind of edge attributes allowing to extract subgraphs like a friend-of-friend network. An edge can have only a single edge type. Typically, there is only a small number of different edge types in a data graph ($N(types) << N(edges)$). For example, LDBC data set with scale factor 1 has 16 edge types, while the total number of edges exceeds 21Mio. We specialize a degree histogram for each available edge type and thereby increase the quality of the cardinality estimation. For each kind of degree and edge type we construct degree histograms that map node identifiers (or intervals) to the number of adjacent edges with a specific edge type.

3.3 Cardinality Estimation

To estimate the cardinality of a query graph, we have to estimate the cardinality of each node and edge in the query graph, and combine them together into a single query graph.

Vertex Cardinality Estimation. To estimate the cardinality of a single node in a query graph, we have to consider selectivities of the predicates.

For a data graph with N nodes the selectivity of a node without any predicate is $sel(v_i) = 1$. If a node v_i has predicate p_k then its selectivity is determined by the selectivity of its predicate

$$sel(v_i|p_k) = sel(p_k) = \frac{N(p_k)}{N} \tag{1}$$

The number of nodes matching a predicate p_k can be taken from the corresponding attribute histogram. The selectivity of a node v_i is defined as the selectivity over an attribute space $A = A_V$, where attributes are assumed to be independent:

$$sel(v_i|p_k, p_l) = sel(p_k) * sel(p_l) \tag{2}$$

Edge and Path Cardinality Estimation. The cardinality estimation of an edge is similar to the cardinality estimation of a node: we use predicate selectivities to estimate the number of edges matching the edge description as in Eq. 2. This estimation is node-irrespective. To estimate the cardinality of a path(1) $C(s-e-t)$ that represents *source(s)-edge(e)-target(t)*, we have to consider source and target nodes of an edge as follows. First, we estimate the selectivity and cardinality of a source and then we multiply it with the average outdegree for an edge. Finally, we multiply it by the selectivity of the target:

$$C(s-e-t) = sel(s) * N * avg.outdeg(e(type)) * sel(e) * sel(t) \tag{3}$$

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

The estimation of a path cardinality is commonly used in graph queries, for example: subgraph isomorphism queries for establishing the join order [15]. The estimation is crucial for enabling efficient graph processing. In the following we improve the above presented standard estimation by focusing on two challenges, namely: (1) notation: considering functional dependencies between attributes and (2) topology: handling nodes with a high outdegree.

4 Considering Functional Dependencies Between Attributes

Considering dependency between attributes of the same query edge or node can increase the quality of cardinality estimation. Inspiring by work on determining functional dependencies between columns [12], we detect soft functional dependencies between attributes of nodes or edges as follows. The dependency between attributes can be expressed by the uncertainty coefficient, also called entropy coefficient as

$$U(Y|X) = \frac{I_{XY}}{H_X} \tag{4}$$

where Y, X are two attributes, $I_{XY} = H_X + H_Y - H(XY)$ is their dependence information, $H(X), H(Y), H(XY)$ are the entropy values of attributes X, Y, and the coentropy of a joint distribution XY, respectively. The uncertainty coefficient $U(Y|X)$ shows how well an attribute value from X defines an attribute value from Y and varies between $[0; 1]$. While $U(Y|X) = 0$ indicates value independence of the two attributes, $U(Y|X) = 1$ expresses a strong dependency. The measure is not symmetric: $U(Y|X) \neq U(X|Y)$.

Example. Assume a data graph represents a social network, where some typical attributes for persons are a firstname and a gender. The functional dependency $U(gender|firstname)$ will be high, while $U(firstname|gender)$ is rather small. In practice we can almost always derive a gender of a person from his firstname. For example, Bob should be a male, while Alice is a female. The converse is not true: if a gender is a female, the firstname cannot be easily derived.

Functional dependencies are static statistics and are calculated offline. To reduce the overhead of determining functional dependencies between all pairs of attributes, we group attributes into several sets and calculate functional dependencies only between attributes of the same set. We create sets based on semantical relatedness between attributes. In many scenarios, for example, social networks, such sets are already defined in the form of an attribute *type* for nodes like a person, a web page, or a city. If the nodes do not have any attribute type, we create characteristic sets [7]. Originally, a characteristic set describes a typical star relationship for the RDF data.

In Fig. 2 we show the transformation of a characteristic set from RDF into the property graph model. As we can see, a characteristic set in the property graph represents only the typical schema of a node. To describe typical outgoing connections for a specific characteristic set in a property graph, we can create an edge characteristic set. It allows describing the topology and the notation

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

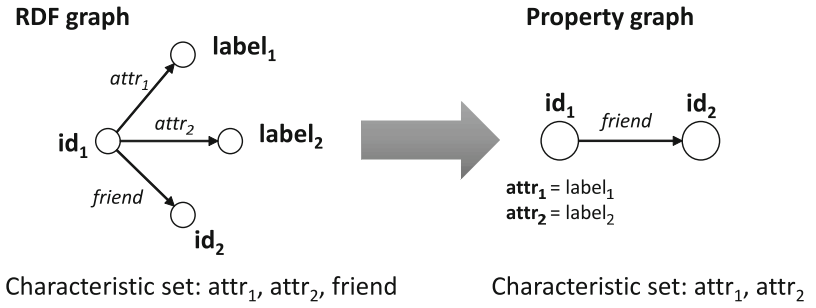


Fig. 2. Characteristic set for a property graph.

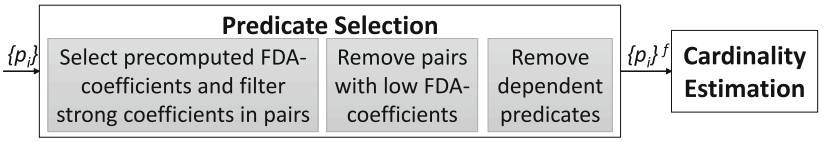


Fig. 3. Predicate selection for cardinality estimation.

Algorithm 1. Predicate Selection.

```
1: function selectPredicates(p[])
2:     predicateMap
3:     for all p_i ∈ p[] do
4:         for all p_i.next ∈ p[] do
5:             if FDA(p_i, p_i.next) ≥ FDA(p_i.next, p_i) then      ▷ select precomputed
   FDA-coefficients and filter the strongest ones in pairs
6:                 predicateMap ← FDA(p_i, p_i.next)
7:             else if FDA(p_i.next, p_i) > FDA(p_i, p_i.next) then
8:                 predicateMap ← FDA(p_i.next, p_i)
9:     for all FDA ∈ predicateMap do
10:        if FDA < threshold then          ▷ remove pairs with low FDA-coefficients
11:            remove FDA
12:    startPredicate ← max(predicateMap)
13:    remove startPredicate from p[]
14:    for all pair ∈ predicateMap do
15:        if pair[2] ∈ p then                          ▷ remove dependent predicates
16:            remove pair[2] from p
17:    p[] ← startPredicate return p[]
```

of a graph more precisely and in such a way we can separate edges from the analyzing attribute dependencies for nodes.

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

Predicate Selection

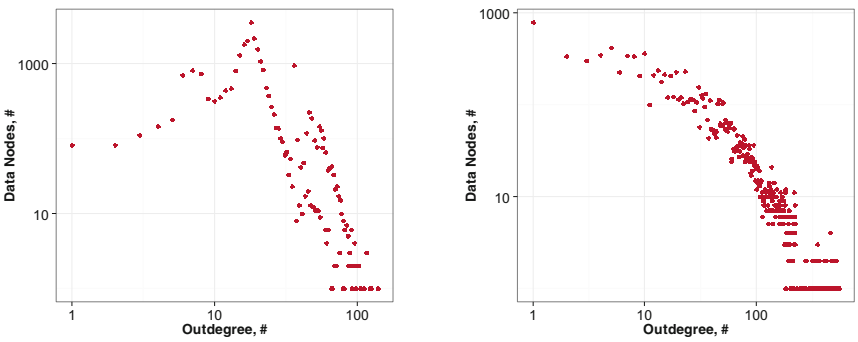
To calculate the selectivity of a node, we first filter out those predicates, whose attributes are functionally dependent from others and therefore can be derived from the already considered attributes. In Fig. 3 and Algorithm 1 we present the process of selecting the predicates. As an input, Algorithm 1 receives a set of node predicates. For each pair of predicates we query their precomputed pair of functional dependency coefficients and choose the largest one (Lines 5-8). Afterwards, we remove all dependencies below a threshold in Lines 9-11, dependent predicates, which can be derived by their pair partners (Lines 14-17), and return filtered predicates.

5 Handling High-Outdegree Nodes

One of the problems in cardinality estimation of graph queries that has been mostly ignored by the research community so far is the special handling of nodes with a large number of outgoing edges. Usually the number of such nodes is much smaller than the total number of nodes in a data graph and ignoring them can lead to dramatic cardinality underestimations.

In Fig. 4 we present the outdegree distributions for the DBpedia and LDBC data sets for a single edge type that we use later in the experimental evaluation in Sect. 6. The tail on the right-hand side of each figure represents nodes with a high outdegree. If we do not treat them separately, the cardinality estimation can produce an estimation error of several orders of magnitude.

For a correct handling of nodes with a high outdegree, we need to answer two questions: (1) how to discover such nodes and (2) how to efficiently store and process them.



(a) DBpedia data set. **(b) LDBC data set.**

Fig. 4. Outdegree diagrams.

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

5.1 Discovery of Nodes with a High Outdegree

Nodes with a high outdegree can be interpreted as outliers. To detect them, we have to study the degree distribution of nodes in a data graph and define a node v_i as an outlier, whose degree is much larger than the average degree ($avg.degree << degree_{v_i}$). For this purpose, we use the algorithm based on the calculation of the modified z-score for a univariate data set [3]. The calculation requires two components: the median and the median of the absolute deviation of the median that is calculated as

$$MAD = median|x_i - \widetilde{x}|, \tag{5}$$

where $\widetilde{x}$ is the sample median. As a consequence, the modified z-score can be computed as

$$M_i = \frac{0.6745 * (x_i - \widetilde{x})}{MAD}, \tag{6}$$

where $E(MAD) = 0.675\sigma$ for large normal data. The authors suggested $M_i > |3.5|$ to be outliers. In our case x_i is an outdegree of node v_i, which has at least one edge of a specific *type*. Therefore, we call a node an outlier, if its modified z-score is $M > |3.5|$. We refer the interested user to the survey of existing outlier detection methods [14].

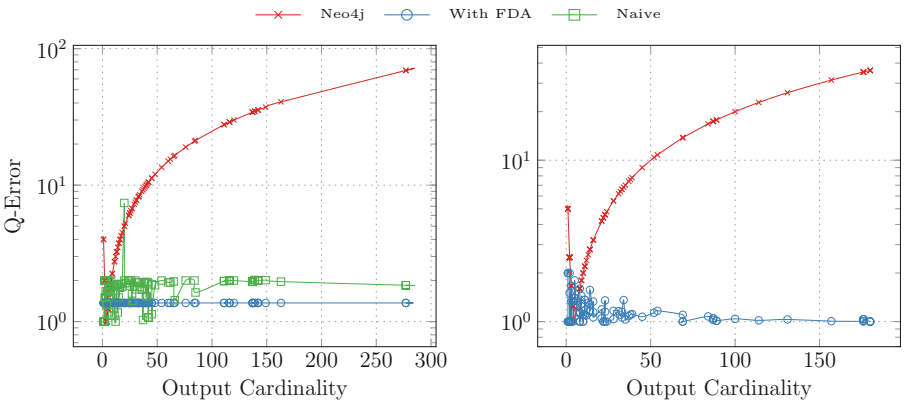
5.2 Processing of High-Outdegree Nodes

Based on the selectivity estimation of the predicates on nodes, we distinguish between nodes with an average outdegree and those with a high outdegree. To identify efficiently, whether a given graph pattern matches one or multiple high-outdegree nodes, we use a lightweight data structure to partially index attributes for high-outdegree nodes. For each attribute, we use an ordered tree structure to map attribute values to high-outdegree nodes. The value of a node in the tree structure provides a reference to the corresponding cardinality of the node in the degree histogram. In a final step, we union the general cardinality estimation of nodes with the cardinality estimation of matched high-outdegree nodes.

6 Experimental Evaluation

In this section we provide an experimental evaluation of our techniques for detecting and leveraging functional dependencies during cardinality estimation and for detecting nodes with a high outdegree. We implemented the proposed techniques in Graphite [11]—a columnar graph processing system—and conducted all experiments on a two socket Linux based system with Intel Xeon X5650 CPUs equipped with 6 cores @2.67 GHz and 48 GB RAM. We use two data sets in our experiments—ldbc scale factor 1 (3.7 Mio. vertices with 17 attributes, 21.7 Mio. edges with 16 types and 4 attributes) and DBpedia (0.2 Mio. vertices with 1543 attributes, 0.8 Mio. edges with 829 types)—and initially

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14



(a) Strong correlation ($\rho = 0.76$) **(b) No correlation** ($\rho = 0.09$)

Fig. 5. Cardinality estimation quality of conjunctive queries on correlated and uncorrelated attributes.

populate them into GRAPHITE and NEO4J. For NEO4J, we created secondary indices on each vertex attribute to allow the system collecting additional statistics about the attributes.

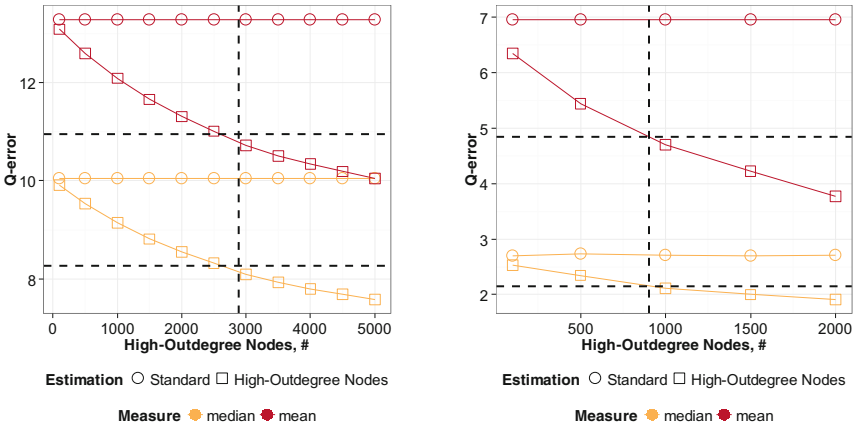
6.1 Correlated Attributes

In this experiment we evaluate the influence of (soft) functional dependencies between vertex attributes on the quality of the cardinality estimation and present our results in Fig. 5. We use the LDBC data set at scale factor 1 and selected two representative conjunctive predicates (cf. Table 1). We evaluate the estimation quality for NEO4J, a naive cardinality estimation (using equi-width histograms with bucket size 1 and attribute independence assumption), and our cardinality estimation that automatically detects functional dependencies between attributes and takes them into account during the estimation process. For a strong functional dependency (see Fig. 5a), our cardinality estimation outperforms the cardinality estimation quality of NEO4J by up to factor 53 and a naive cardinality estimation (under the independence assumption) by up to 50 % for large output cardinalities. For uncorrelated attributes (see Fig. 5b) we decide, based on the estimated functional dependency and a threshold, whether we estimate the conjunctive cardinality under the independence assumption or by exploiting the knowledge about functional dependencies. For a weak functional dependency we estimate the cardinality under the independence assumption and represent both, the naive and our solution with functional dependency analysis by the same blue plot line.

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

6.2 High-Outdegree Vertices

In this set of experiments we evaluate the automatic detection of high-outdegree vertices and compare it with manually chosen numbers of nodes with a high outdegree based on the top-k principle. We conducted our evaluation on two data sets, DBpedia and LDBC, and generated different query templates instantiated with different predicate values (cf. Table 1).



(a) DBpedia data set. **(b) LDBC data set.**

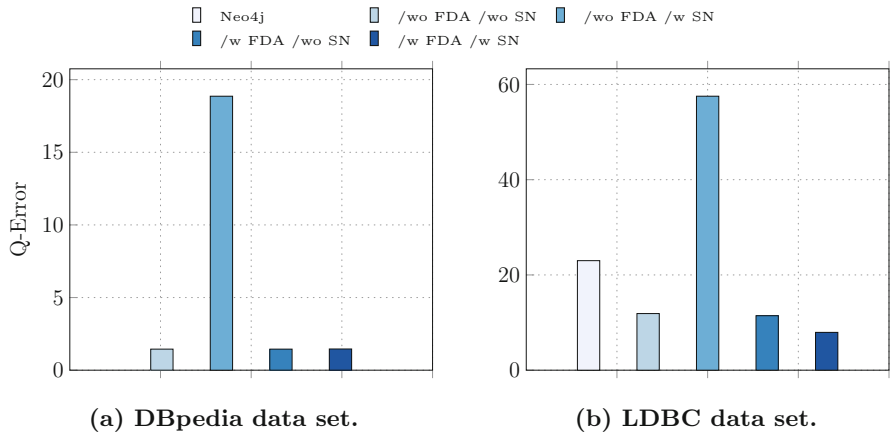
Fig. 6. Evaluating functional dependency analysis.

While for the DBpedia data set (cf. Fig. 6a) the system automatically detected the kink at around 2886, it determined a value of around 900 for the LDBC data set. For both data sets, we observed that by increasing the number of considered high-outdegree nodes, the q-error decreases more slowly than before the kink.

6.3 End-to-End Cardinality Estimation

In this test we evaluate the influence of functional dependencies between attributes and the presence of high-outdegree nodes on the quality of the estimation (cf. Fig. 7). We consider four different configurations combined from two features: with or without functional dependency analysis (FDA) and with or without automatic selection of high-outdegree nodes (SN). Neo4j provides a good estimation quality for the LDBC data set but fails estimating the cardinality of queries for the DBpedia data set which requires an estimation of selective predicates. Most queries match at most a single source vertex and Neo4j estimates these high-selective vertices to 0. Since the q-error cannot be computed for a cardinality (exact and estimated) of 0, we omit the results for Neo4j for the DBpedia data set in Fig. 7a.

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14



(a) DBpedia data set. **(b) LDBC data set.**

Fig. 7. Cardinality estimation quality for Neo4j and possible combinations of our two proposed techniques.

Since both query templates do not hit many high-outdegree nodes, the cardinality estimation improvement is only marginal. For both query templates, the FDA has the highest impact, and therefore, represents in conjunction with the handling of high-outdegree vertices the best results. We conclude that both techniques are important key components for efficient cardinality estimation on subgraph isomorphism queries and can reduce the cardinality estimation error significantly.

7 Conclusion

We tackled two important challenges that arise in graph pattern cardinality estimation caused by the skewedness of the degree distribution and the irregularity of exposed attributes in vertices and edges present in real-world graphs, namely: detection of functional dependencies between node attributes and consideration of nodes with a high outdegree. By analyzing two real-world graph data sets with a rich set of attributes and a power-law vertex degree distribution, we identified that these two aspects are important for the cardinality estimation of subgraph isomorphism queries over property graphs. With our solution for the cardinality estimation considering both aspects, we outperform a naive approach relying on average outdegree measures and the independence assumption in conjunctive predicates by up to 50 % and the cost-based query optimizer of Neo4j by up to a factor of 50.

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

A Evaluated Queries

Table 1. Query templates used in the evaluation.

Figure	Data Set	Query Template
5a	LDBC	`MATCH (m:person)` `WHERE m.firstname=(?) AND m.gender=(?) RETURN m;`
5b	LDBC	`MATCH (m:person)` `WHERE m.lastname=(?) AND m.gender=(?) RETURN m;`
6a	DBpedia	`MATCH (m:)-[:type]->(n:)` `WHERE m.type=(?) RETURN n;`
6b	LDBC	`MATCH (m:person)-[:knows]->(n:person)` `WHERE m.id=(?) RETURN n;`
7b	LDBC	`MATCH (m:person)-[:knows]->(n:person)` `WHERE m.firstname=(?) AND m.gender=(?) RETURN n;`
7a	DBpedia	`MATCH (m:)-[:type]->(n:)` `WHERE m.airport=(?) AND m.short_name=(?) RETURN n;`

References

1. Boldi, P., Rosa, M., Vigna, S.: HyperANF: approximating the neighbourhood function of very large graphs on a budget. In: Proceedings of the WWW 2011, pp. 625–634 (2011)
2. Cormode, G., Garofalakis, M., Haas, P.J., Jermaine, C.: Synopses for massive data: samples, histograms, wavelets, sketches. Found. Trends Databases **4**(1–3), 1–294 (2012)
3. Iglewicz, B., Hoaglin, D.C.: How to Detect, Handle Outliers. ASQC Quality Press, Milwaukee (1993)
4. Lee, J., Han, W.-S., Kasperovics, R., Lee, J.-H.: An in-depth comparison ofsubgraph isomorphism algorithms in graph databases. In: Proceedings of the VLDB Endowment, vol. 6, pp. 133–144 (2012)
5. Lu, Y., Cheng, J., Yan, D., Wu, H.: Large-scale distributed graph computing systems: An experimental evaluation. Proc. VLDB Endow. **8**(3), 281–292 (2014)
6. Martínez-Bazan, N., Águila Lorente, M.A., Muntés-Mulero, V., Dominguez-Sal, D., Gómez-Villamor, S., Larriba-Pey, J.-L.: Efficient graph management based on bitmap indices. In: Proceedings of the IDEAS 2012, pp. 110–119 (2012)
7. Neumann, T., Moerkotte, G., Sets, C.: Accurate cardinality estimation for RDF queries with multiple joins. In: Proceedings of the ICDE 2011, pp. 984–994 (2011)
8. Neumann, T., Weikum, G.: RDF-3X: A RISC-style engine for RDF. Proc. VLDB Endow. **1**(1), 647–659 (2008)
9. Palmer, C.R., Gibbons, P.B., Faloutsos, C., ANF: a fast and scalable tool for data mining in massive graphs. In: Proceedings of the SIGKDD 2002, pp. 81–90 (2002)

Final edited form was published in "Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops. Waikoloa 2015", S. 184-198. ISBN: 978-3-319-41576-5
https://doi.org/10.1007/978-3-319-41576-5_14

10. Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J.-P., Schönberg, M., Zwiener, J., Naumann, F.: Functional dependency discovery: an experimental evaluation of seven algorithms. Proc. VLDB Endow. **8**(10), 217–228 (2015)
11. Paradies, M., Lehner, W., Bornhövd, C.: GRAPHITE: an extensible graph traversal framework for relational database management systems. In: Proceedings of the SSDBM 2015, pp. 29: 1–29: 12 (2015)
12. Paradies, M., Lemke, C., Plattner, H., Lehner, W., Sattler, K.-U., Zeier, A., Krueger, J.: How to juggle columns: an entropy-based approach for table compression. In: Proceedings of the IDEAS 2010, pp. 205–215 (2010)
13. Rodriguez, M.A., Neubauer, P.: Constructions from dots and lines. Bull. Am. Soc. Inf. Sci. Technol. **36**(6), 35–41 (2010)
14. Seo, S.: A review and comparison of methods for detecting outliers in univariate data sets. Master's thesis, Faculty of Graduate School of Public Health, University of Pittsburgh (2006)
15. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. Proc. VLDB Endow. **1**(1), 364–375 (2008)
16. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: SPARQL basic graph pattern optimization using selectivity estimation. In: Proceedings of the WWW 2008, pp. 595–604 (2008)