Color: A Framework for Applying Graph Coloring to Subgraph Cardinality Estimation

Kyle Deeds
University of Washington
kdeeds@cs.washingon.edu

Diandre Sabale
University of Washington
dmbs@uw.edu

Moe Kayali
University of Washington
kayali@cs.washington.edu

Dan Suciu
University of Washington
suciu@cs.washington.edu

ABSTRACT

Graph workloads pose a particularly challenging problem for query optimizers. They typically feature large queries made up of entirely many-to-many joins with complex correlations. This puts significant stress on traditional cardinality estimation methods which generally see catastrophic errors when estimating the size of queries with only a handful of joins. To overcome this, we propose COLOR, a framework for subgraph cardinality estimation which applies insights from graph compression theory to produce a compact summary that captures the global topology of the data graph. Further, we identify several key optimizations that enable tractable estimation over this summary even for large query graphs. We then evaluate several designs within this framework and find that they improve accuracy by up to $10^3\times$ over all competing methods while maintaining fast inference, a small memory footprint, efficient construction, and graceful degradation under updates.

KEYWORDS

Cardinality Estimation, Graph Databases, Graph Summarization, Query Optimization

PVLDB Reference Format:
Kyle Deeds, Diandre
Sabale, Moe Kayali, and Dan Suciu. Color: A Framework for Applying Graph Coloring to Subgraph Cardinality Estimation. PVLDB, 18(2): 130 - 143, 2024.
doi:10.14778/3705829.3705834

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/uwdb/color.

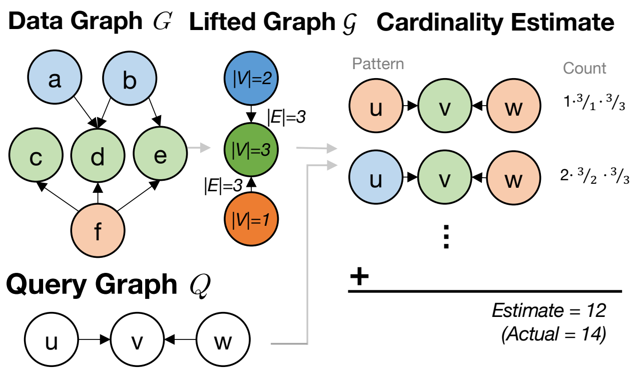

Figure 1: Lifted counting example. We estimate the number of occurrences of the query graph Q pattern ($u \rightarrow v \leftarrow w$) in the data graph G. First, the data graph is partitioned offline: the resulting summary is stored as the *lifted graph* $\mathcal{G}$. At runtime, the cardinality estimate is computed on the lifted graph $\mathcal{G}$ without reference to the underlying data graph.

1 INTRODUCTION

The core operation of queries over graphs is *subgraph matching* where instances of a query graph pattern, Q, are found within a larger data graph, G. This is the main primitive in graph query languages like GQL, SQL/PGQ, and SPARQL which are implemented by graph database management systems such as Neo4J, TigerGraph, Virtuoso, QLever, and Amazon Neptune [3, 4, 10, 11, 27]. On critical workloads like financial fraud detection, subgraph matching is a part of virtually all analysis pipelines [33]. For example, money laundering often manifests as cycles of transactions in financial networks [17, 31].

Estimating the count, or *cardinality*, of subgraphs is crucial to the optimization of graph queries. Specifically, subgraph matching algorithms are generally either search-based or join-based [5, 37]. In both cases, the query optimizer attempts to choose a query plan (a query-vertex order or join order, respectively) which minimizes the size of intermediate results. However, graph workloads commonly contain large query graphs with ten or more edges; these queries may produce much larger intermediates which makes finding a good query plan particularly crucial for execution [6, 25, 34]. Historically, cardinality estimation has been the key obstacle in identifying good query plans over relational data [9, 22, 29]. Applying these lessons to the graph setting, the database community has begun a concerted effort to improve subgraph cardinality estimation, with many recent methods and benchmarks [8, 21, 28, 29, 36]. In this vein, our paper applies novel insights from graph theory to improve cardinality estimation for subgraph matching.

Proceedings of the VLDB Endowment, Vol. 18, No. 2 ISSN 2150-8097.
doi:10.14778/3705829.3705834

The first two authors contributed equally to this work.

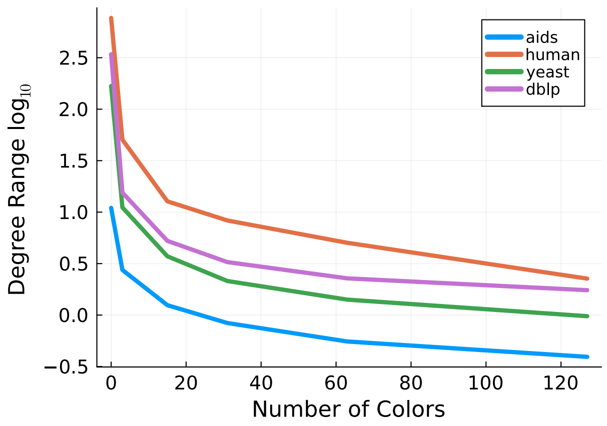

Figure 2: Accuracy of coloring as the number of colors increases

Prior work on this problem generally follows one of three approaches. *Pattern-based methods* [28, 41] count the frequency of a small, pre-defined set of patterns offline, combining them at runtime to produce estimates by assuming independence between their counts. Simple queries can be effectively captured by the combination of one or two patterns—complex ones cannot, as they require combining many correlated patterns. For this reason, performance is adequate on simple queries but degrades rapidly for workloads with larger queries: resulting in median underestimates of up to $\mathbf{10^{10}}\times$ (Sec. 8). *Online-sampling methods* [8, 21, 23, 38, 43] perform runtime sampling, generally random walks, to estimate cardinalities. Unfortunately, these methods often suffer from sampling failure. While generating samples for smaller, less selective query patterns is feasible, finding even a single match for complex patterns can be challenging [24]. We later show **60−99%** failure rate for online sampling methods on the most challenging workloads (Sec. 8). Further, online sampling in disk-based or distributed settings can incur a prohibitive latency as it relies on fast random reads over the whole graph. *Summarization methods* [7, 36] group nodes in the data graph into a super structure and store summary statistics. However, the grouping is done by predefined rules or hashing, without regard to the edge distribution between these groups. Because these summaries are not tailored to the graph structure, *i.e.* make a uniformity assumption, they produce median error of up to $\mathbf{10^{12}}\times$, and we find that they timeout on larger queries (Sec. 8).

In this paper, we propose a new approach to subgraph cardinality estimation based on graph compression; we call this method Color. By taking advantage of recent advances in lossy graph representations such as quasi-stable coloring [19], we approximately capture the topology of the graph in a small summary. We then directly estimate cardinalities on the summary without needing to access the data graph.

The key idea is to color the graph G such that nodes of the same color have a similar number of edges to each other color. This mitigates the effect of the uniformity and independence assumptions. In Figure 1 for example, the coloring assigns the *c*,*d*,*e* nodes to the same green color. This is because green nodes have a similar number of incoming edges from blue and orange nodes, and no out edges. Meanwhile, *a*,*b* are assigned to the blue color as they have a similar number of outgoing edges to green nodes and none to orange nodes. This helps mitigate the uniformity assumption because, within nodes of a fixed color, the edge distribution is nearly uniform. Further, because high and low degree nodes tend to be placed in different colors, correlations in the connections between them can be identified, mitigating the independence assumption.

Real world graphs are more complex, but it turns out that our approach can meaningfully capture their topology with a small number of colors, typically just 32 in our experiments. Figure 2 shows the maximum difference in edge counts from nodes in one color to nodes in another, averaged over all pairs of colors for four of our benchmark datasets. Lower values indicate a better coloring and smaller differences between the two most different nodes in each color. A handful of colors is sufficient to capture most of the graph topology and reduce non-uniformity by 1-2 orders of magnitude.

With these colorings in mind, we return to the example in Figure 1. During the offline phase, our method takes the data graph, G, and produces a compact summary, $\mathcal{G}$ called a *lifted graph*, with one supernode for each color (Sec. 3). We keep statistics on the number and kinds of edges which pass between these colors. During the online phase, the cardinality estimate is computed on the lifted graph by performing a weighted version of subgraph counting which we call *lifted subgraph counting* (Sec. 4). To extend this to cyclic queries, which occur frequently in graph databases, we also propose a technique based on a novel statistic called the *path closure probability* (Sec. 5).

To enable efficient inference on a more detailed, and therefore accurate, lifted graph, we propose three critical optimizations. Tree-decompositions and partial aggregation, introduced in Sec. 7.1, reduce the inference latency by over **100**×. Importance sampling and Thompson-Horowitz estimation over the lifted graph, the subject of Sec. 7.2, ensure a linear latency w.r.t. the size of the query while maintaining a **6x** lower error than a naive sampling approach. Lastly, we demonstrate how to maintain the lifted graph under updates in Sec. 7.3, reducing the need to rebuild the summary by providing reasonable estimates even when over 1/2 of the graph is updated.

In summary, we make the following contributions:

- Develop the COLOR framework for producing lifted graph summaries from colorings (Sec. 3) and evaluate six possible coloring schemes (Sec. 6).
- Define a general formula for performing inference over a lifted graph, show its optimality for acyclic queries (Sec. 4), and extend it to cyclic ones (Sec. 5).
- Develop optimizations that allow for efficient and accurate inference and robust handling of updates (Sec. 7). These optimizations are: tree-decomposition, importance sampling, and Thompson-Horowitz estimation.
- Empirically validate COLOR's superior performance on eight standard benchmark datasets and against nine comparison methods (Sec. 8).

2 PROBLEM SETTING

Property Graphs. The data model that we use for this work is property graphs. These are directed graphs where each edge and vertex is associated with a set of attributes. These attributes can be simple labels (e.g. $:friendOf$) or key-value pairs (e.g. $Age=72$). This is the most general data model for graphs; it matches the model of GQL, Cypher, and GraphQL, and it captures the RDF model [2, 12, 18].

Formally, we define property graphs as follows:

Definition 1. *A property graph, $G(V,E,\lambda,\chi)$, is a directed graph with vertices V, edges E, attribute domain $\mathbb{A}$, and two annotation functions,*

- $\lambda: \quad V \rightarrow 2^{\mathbb{A}}$ *which maps a vertex to a set of attributes* [1]
- $\chi: \quad E \rightarrow 2^{\mathbb{A}}$ *which maps an edge to a set of attributes*

Subgraph Counting. The goal of subgraph counting is to find the number of occurrences of a query graph Q in a larger data graph G. On a property-less graph, an occurrence is defined as any mapping from the vertices in Q to the vertices in G such that all edges in Q are mapped to edges in G, i.e. a homomorphism from Q to G. To account for properties, each vertex and edge of the query graph is associated with a predicate, P, that returns true or false based on the attributes (e.g. "*hasLabel:friendOf*" or "*age>60*"). Formally, this is defined as follows,

Definition 2. *For a property-less query graph Q and data graph G, we define the set of subgraph matches as,*

$$\text{hom}(Q,G) = \{\pi : V_Q \rightarrow V_G \mid \pi(E_Q) \subseteq E_G\}$$

Each match, π, is a function from V_Q to V_G. When Q and G are property graphs, we add the natural conditions for each π,

$$P_v(\pi(v)) = \mathbf{1} \quad \forall v \in V_Q \qquad P_e(\pi(e)) = \mathbf{1} \quad \forall e \in E_Q \tag{1}$$

The subgraph count is then $|\text{hom}(Q,G)|$.

This work studies the problem of *cardinality estimation.* Recent work in both graph and relational databases has demonstrated the importance of cardinality estimation for producing efficient query plans [22, 30]. This problem consists of two phases: 1) a preprocessing phase where the statistics, denoted $\mathbf{s}$, are computed and 2) an online phase where query graphs come in and approximate subgraph counts are returned. Formally, we can view it as follows,

Definition 3. *A cardinality estimation method $\mathcal{E}$ consists of two algorithms: 1) computing statistics during the preprocessing phase, $\mathbf{s} \overset{def}{=} \mathcal{E}_{pre}(G)$ and 2) estimating the cardinality during the online phase, $c \overset{def}{=} \mathcal{E}_{on}(Q,\mathbf{s}) \in \mathbb{R}_+$. The goal is to produce an estimate where $c \approx |\text{hom}(Q,G)|$.*

The primary metrics for these algorithms are: 1) the accuracy of $c \approx |\text{hom}(Q,G)|$ 2) the latency of $\mathcal{E}_{on}$ and 3) the size of $\mathbf{s}$.

Traditional Estimators. The classic System R approach to cardinality estimation in relational databases combines the number tuples in the joining relations, the number of unique values in the joining columns, and assumptions (uniformity, independence, preservation of values) to produce a basic cardinality estimate [16, 22]. In the graph setting, this estimation method looks like the following,

Definition 4. *Given a query graph $Q(V_Q,E_Q)$ and data graph $G(V,E)$, the traditional estimation method is,*

(1) $\mathcal{E}^{trad}_{pre}(G) = (|V|,|E|)$ *(number of vertices and of edges)*

(2) $\mathcal{E}^{trad}_{on}(Q,\mathbf{s}) = \prod_{v \in V_Q} |V| \cdot \prod_{e \in E_Q} \frac{|E|}{|V|^2}$

Intuitively, the estimation formula calculates the number of possible embeddings of the query graph in the data graph, $\prod_{v \in V_Q} |V|$, then scales this by the probability of any embedding having the correct set of edges, $\prod_{e \in E_Q} \frac{|E|}{|V|^2}$. In effect, this estimation procedure assumes that the data graph is distributed like an Erdos-Renyi random graph with $|E|$ edges and $|V|$ vertices and produces an accurate estimate given this assumption. However, the structure of most real world graphs is much more complex. This results in very different subgraph counts from those on Erdos-Renyi random graphs and motivates the use of more complex estimators.

Example 1. *Recall that the standard estimate of a join $Q(x,y,z) = R(x,y) \wedge S(y,z)$ is $\frac{|R| \cdot |S|}{\max(|R.y|,|S.y|)}$. When both R,S are the edge relation E, then $R.y = S.y = V$ (assuming no isolated vertices) and the traditional estimator becomes $\frac{|E| \cdot |E|}{|V|}$, which is the same as the formula above, $|V|^3 \frac{|E|^2}{|V|^4}$.*

3 COLORINGS & LIFTED GRAPHS

Colorings and lifted graphs are the core of our framework, so we start by formally defining them here. For clarity of presentation, we will begin by ignoring predicates and reintroduce them later. Given a graph $G(V,E)$, a *coloring* σ is a function from V to C where C is a small set of colors, $|C| \ll |V|$. Under a *quasi-stable* coloring [19], two vertices in the same color will have similar distributions of outgoing edges to different colors, i.e. any two red vertices should have nearly the same number of edges to blue vertices. Formally,

Definition 5. *A coloring, σ, is quasi-stable if the following properties hold for all pairs of vertices v_1,v_2. If $\sigma(v_1) = \sigma(v_2)$, then:*

$$\forall c \in C : |\{(v_1,v) \in E | \sigma(v) = c\}| \approx |\{(v_2,v) \in E | \sigma(v) = c\}| \tag{2}$$

In English, σ is quasi-stable if for any two colors c_0,c, any two vertices v_1,v_2 colored c_0 have approximately the same number of neighbors colored c. If we replace $\approx$ with $=$ in (2), then σ is called a *stable coloring.* Stable colorings are commonly used in graph isomorphism algorithms, and have elegant theoretical properties [13, 15]. However, they are unsuitable for our purpose, because stable colorings of real-world graphs require a very large number of colors [19]. In fact, in a random graph, every vertex has a distinct color with high probability [14]. Instead, we relax equality $=$ to approximation $\approx$ in Definition 5 in exchange for using a much smaller number of colors. To do this, we apply a variety of coloring algorithms (Sec. 6) which produce a dramatic reduction in the number of colors with only a small relaxation of $=$ to $\approx$. We demonstrate this in Fig. 2. This graph shows the average range of degrees from nodes in one color to nodes in another as the number of colors varies. Across all graphs, a coloring with just 32 colors (using the Quasi-Stable coloring method from [19]) lowers the average degree range by over 2 orders of magnitude as compared to the initial graph.

For any color $c \in C$, we denote the set of vertices colored c by $V_c \overset{\text{def}}{=} \{v \in V \mid \sigma(v) = c\}$. For any two colors c_1,c_2 we denote the set of edges between them by $E_{c_1c_2} \overset{\text{def}}{=} E \cap (V_{c_1} \times V_{c_2})$. The *lifted graph* consists of a quasi-stable coloring, together with statistics for each pair of colors:

[1] 2^X denotes the set of subsets of X.

Definition 6. *Fix a directed graph $G=(V,E)$, and a coloring σ with colors C. A* lifted graph *is a triple, $\mathcal{G}=(F,\psi,\tau)$, consisting of the following parts.*

- *$F=(V_F,E_F)$ is a graph where $V_F=C$ and $E_F=\{(c_1,c_2)\mid E_{c_1c_2}\neq\emptyset\}$.*
- *$\psi:C\rightarrow\mathbb{N}$ where $\psi(c)=|V_c|$*
- *$\tau:E_{c_1c_2}\rightarrow\mathbb{R}_+$ maps edges of the graph to statistics about the edges in $E_{c_1c_2}$*

In this paper, we consider the following choices for the edge statistics function τ:

$$\tau_{\min}(c_1,c_2)=\min\{|\{(v_1,v_2)\mid v_2\in V_{c_2}\}|\mid v_1\in V_{c_1}\}$$
$$\tau_{\text{avg}}(c_1,c_2)=\frac{|E_{c_1c_2}|}{|V_{c_1}|}$$
$$\tau_{\max}(c_1,c_2)=\max\{|\{(v_1,v_2)\mid v_2\in V_{c_2}\}|\mid v_1\in V_{c_1}\}$$

They represent the minimum degree, the average degree, and the maximum degree of a vertex in c_1 to vertices in c_2 respectively. In the following sections, it will be sufficient to assume that τ means τ_{avg}, unless otherwise noted.

Thus, ψ is a function that returns the number of vertices with the color c, and τ is a function that returns statistics about the set of edges between a pair of colors. The lifted graph forms a fuzzy compression of the data graph, and is computed offline, during preprocessing. In our approach, this is the statistic, $\mathbf{s}=\mathcal{E}_{pre}$, as defined in Def. 3.

Example 2. *Consider again the example data graph and coloring in Figure 1. First, the data graph G is colored with a quasi-stable coloring. This produces three different colors, which reflect that topologically there are three different "kinds" of vertices in the data graph. In particular, orange vertices have an 3 out-degree, no in-degree; the green vertices no out-degree, 2 out-degree; and blue 1.5 average out-degree, no in-degree. Due to the arrangement of these edges, we can produce a "good" coloring where vertices a and b are assigned to the blue partition. Further, we can produce the lifted graph $\mathcal{G}$ by choosing the degree sum as our edge statistic. In this figure, the vertex labels are the partition cardinalities, i.e. the values of ψ. The label edges represent the sum of edges between two colors: for example, green vertices have a total of 3 edges into the orange vertices, i.e. the values of τ_Σ. Note that this lifted graph closely captures the distribution of edges in the data graph while being half the size.*

Lifted Property Graphs. To account for attributes and predicates in the lifted graph, we adjust the definition of ψ and τ to accept predicates in addition to colors. Suppose that a query has a vertex predicate P, then we define $V_{c,P}$ as the set of data vertices in the color c which pass the predicate P. Similarly, E_{c_1,c_2,P_e,P_v} is the set of edges starting in color c_1, matching predicate P_e, and landing on a node in color c_2 which matches P_v. With this, we can then redefine ψ and τ_{avg},

$$\psi(c,P)=|V_{c,p}| \qquad \tau_{avg}(c_1,c_2,P_e,P_v)=\frac{|E_{c_1,c_2,P_e,P_v}|}{|V_{c_1}|}$$

We allow these functions to be exact or approximate in order to accommodate more complex predicates. If the predicates are all of the form *hasLabel*$:X$ and there are few edge label/vertex label combinations, then this can be calculated and stored explicitly. However, predicates like range or LIKE benefit from approximating $|E_{c_1,c_2,P_e,P_v}|$ using standard techniques like histograms or n-grams. This can then be extended to t_{min} and t_{max} using techniques similar to those in [9].

Table 1: Notation Dictionary

Symbol	Meaning
$G(V,E,\lambda,\chi)$	Property graph with vertices V, edges E, vertex attributes λ, and edge attributes χ.
$\mathcal{G}(F,\psi,\tau)$	Lifted graph with color graph F, color cardinalities ψ, and color edge statistics τ.
$W(\pi,Q,\mathcal{G})$	Estimate of the subgraph count for query Q with coloring π based on lifted graph $\mathcal{G}$.
$\Phi(Q,\mathcal{G})$	Estimate of the subgraph count for query Q based on lifted graph $\mathcal{G}$.
$\gamma(C_1,C_2,D)$	Probability of a path closing a cycle from C_1 to C_2 with directionality D

4 LIFTED SUBGRAPH COUNTING

We have described the lifted graph, a small weighted graph which approximately captures the topology of the data graph. Next, we show how to use the lifted graph for cardinality estimation, which we call the *lifted subgraph counting problem*,

Definition 7. *Fix a lifted graph $\mathcal{G}=(F,\psi,\tau)$ and a query graph Q. An estimation procedure is a function*

$$W:\hom(Q,F)\times Q\times\mathcal{G}\rightarrow\mathbb{R}_+$$

The lifted subgraph count is,

$$\Phi(Q,\mathcal{G})=\sum_{\pi\in\hom(Q,F)}W(\pi,Q,\mathcal{G}) \tag{3}$$

A homomorphism $\pi:Q\rightarrow F$ associates to each query vertex x a color $\pi(x)\in C$; we will also call π a *coloring* of the query Q. The estimator $W(\pi,Q,\mathcal{G})$ approximates the number of outputs of the query with coloring π. Note, we generally drop Q and $\mathcal{G}$ when obvious from context. The total estimate, $\Phi(Q,\mathcal{G})$ is simply the sum over all colorings π. To see the intuition, recall that errors in traditional cardinality estimation come from correlation and skew. For example, the former could mean that high degree vertices are more likely to be connected to high degree vertices (or vice versa), while the latter means that the degrees of vertices have a wide range. By grouping vertices into colors based on their local topology (their degrees, their neighbors' degrees, etc), and fixing a particular coloring π of the query vertices, we reduce the variance of the estimate $W(\pi)$, leading to a reduced error overall. In the rest of this section we define the estimate function W assuming that the query graph is acyclic. We will extend it to arbitrary query graphs in Sec. 5.

4.1 Acyclic Query Graphs

For acyclic queries, we use the following function W:

Definition 8 (Lifted Estimator for Acyclic Queries). *Let $Q=(V_Q,E_Q)$ be an acyclic query graph, and let $x_1,\ldots,x_{|V_Q|}$ be a topological ordering of its vertices: in other words, every vertex x_j is connected to* *some x_i for $i<j$. For any homomorphism $\pi: Q \to F$, we define:*

$$W(\pi) \overset{def}{=} \psi(\pi(x_1), P_{x_1}) \prod_{(x_i,x_j)\in E_Q} \tau(\pi(x_i),\pi(x_j),P_{x_j},P_{(x_i,x_j)}) \quad (4)$$

We adopt the convention that if x_i and x_j are connected by a reverse edge, i.e. E_Q contains (x_j,x_i) rather than (x_i,x_j), then this is reflected in the query graph with a predicate, $dir = \leftarrow$, on the edge. Intuitively, we process the edges in the topological order, so we always multiply with the in/outdegree of the color assigned to the topologically earlier vertex. In this paper we choose the topological order such as to minimize the time needed to compute $\hom(Q,G_F)$, see Sec. 7.

Example 3. *Here we show that the lifted graph estimator generalizes the traditional estimator in Def. 4. We illustrate this with a graph $G=(V,E)$ and the 2-edge query $Q(x,y,z) = E(x,y) \wedge E(y,z)$ from Example 1. Assume that we use a lifted graph, $\mathcal{G}(F,\psi,\tau)$, consisting of a single color c and a single edge: $F = (\{c\},\{(c,c)\})$. Then the statistics are $\psi(c) = |V|$ and $\tau(c,c) = \frac{|E|}{|V|}$ (recall that we assumed τ refers to τ_{avg}), there is a single homomorphism $\pi: G \to F$, and our estimate is $W(\pi) = \psi(c)\,(\tau(c,c))^2 = |V|\frac{|E|^2}{|V|^2} = \frac{|E|^2}{|V|}$. This is the same as the traditional estimator in Example 1.*

Example 4. *We illustrate now how a better designed lifted graph can lead to an improved estimator. Assume that the data graph is the disjoint union of two graphs, $G = G_1 \cup G_2$, where $G_1(V_1,E_1)$ is a 2-regular graph on 10000 vertices, and $G_2(V_2,E_2)$ is a clique of size 100. Suppose Q is a path of length k. Since the average degree in G is ≈ 4, a traditional estimate for Q is $10100 \cdot 4^k$, which vastly underestimates the true count, because it doesn't capture the skew and correlation introduced by the clique sub-graph. Suppose we pre-compute a lifted graph consisting of two colors: green contains all vertices V_1 and red color contains all vertices V_2. There are only two edges (green,green) and (red,red), and therefore only two colorings π of the query Q. We compute W separately for each of the two colorings, then return their sum, $100 \cdot 99^k + 10000 \cdot 2^k$, which, for our simple data graph, is an exact count.*

4.2 Special Case: Stable Colorings

As a theoretical justification of our method, we prove that if the lifted graph is a *stable* coloring (meaning: $\approx$ is $=$ in Def. 5), then our estimate for acyclic queries is exact, although we defer the formal proof to the technical report [1] for space.

Theorem 1. *Let $\mathcal{G}$ be a lifted graph defined by a stable coloring σ. Then $\tau_{min} = \tau_{avg} = \tau_{max}$, and, for any acyclic query Q, the lifted graph estimator is exact:*

$$|\hom(Q,G)| = \Phi(Q,\mathcal{G}) \quad (5)$$

This theorem states that stable colorings are a perfect statistic for cardinality estimation of acyclic query graphs. However, we cannot use them in practice, because the number of stable colors needed to represent real graphs is close to the number of vertices in the data graph [19, 26]. Fortunately, we can prove an additional corollary for quasi-stable colorings:

Corollary 1. *Let ϵ be the approximation error of the lifted graph $\mathcal{G}$ defined as,*

$$\epsilon = \max_{c_1,c_2\in C} \frac{\tau_{\max}(c_1,c_2)}{\tau_{\min}(c_1,c_2)} \quad (6)$$

Then, we can bound the error of our subgraph estimator as,

$$max\left(\frac{\Phi(Q,\mathcal{G})}{|hom(Q,G)|}, \frac{|hom(Q,G)|}{\Phi(Q,\mathcal{G})}\right) \leq \epsilon^{(|V_Q|-1)} \quad (7)$$

Intuitively, as colorings approach stability, the estimate converges to the true subgraph count.

5 HANDLING CYCLES

Cyclic queries are a fundamentally different challenge for cardinality estimators because they allow complex dependencies between vertices within the query graph. Practically, they require estimating the probability that an edge between two vertices exists, conditioned on the fact the these vertices are already connected in the query graph. This section outlines new techniques to estimate this *cycle closure probability*.

5.1 Cycle Closure Probability

To ground this discussion, we begin by defining the probability space and random variables. The former is defined by a uniform random selection of $|V_Q|$ vertices from V_G with replacement. This is equivalent to a random mapping from V_G to V_Q. The set of vertices selected by this process is denoted with the random variables $V_1,\ldots,V_{|V_Q|}$. Further, each edge of the query graph, $(v_i,v_j) = e_i \in E_Q$, is associated with a binary random variable E_i which is true when $(V_i,V_j) \in E_G$. In other words, E_i is true iff the data vertices mapped to that edge of the query have an edge between them.

As a basic example, we can calculate the unconditioned probability of E_i for any edge e_i as follows,

$$P(E_i) = \frac{|E_G|}{|V_G|^2}$$

Further, we can express the cardinality of an arbitrary query as,

$$|Hom(Q,G)| = |V_G|^{|V_Q|} \cdot P(\cap_{e_i\in E_Q} E_i)$$

With conditional probability, we can expand the probability as,

$$P(\cap_{e_i\in E_Q} E_i) = \prod_{e_i\in E_Q} P(E_i|E_1,\ldots,E_{i-1})$$

When the endpoints of an edge are contained within the previous edges, $e_i \in \cap_{j=1}^{i-1} e_j$, the probability within the product is a cycle closure probability. It is this probability which we try to estimate in this section, and the crucial challenge is estimating the effect of the previous edges, $E_1,\ldots,E_{i-1}$.

The naive solution is to consider all patterns of a fixed size (i.e. the pattern induced by $E_1,\ldots,E_{i-1}$) and calculate the probability of an edge occurring between two nodes of that pattern in the data graph. However, the number of patterns increases super-exponentially in their size due to the choices of basic graph pattern, edge direction, and predicates, so this approach is infeasible for all but the smallest queries. Our approach attempts to tackle this by considering a smaller set of patterns and composing them smartly.

5.2 Path Closure Probability

We introduce *path closure probabilities* which represent the probability that a path in the data graph is "closed", i.e. there is an edge from the starting vertex to the ending vertex. To limit the number of probabilities that we store, paths are grouped by their directionality, e.g.

$D = \{\leftarrow, \rightarrow\}^k$, and by the color of their starting and ending vertices. We denote this probability as,

Definition 9. *Let $\mathcal{P}_{c_1,c_2,D}(G)$ be the set of paths in the data graph G with directions matching D and starting/ending color c_1/c_2. Let $\mathcal{C}_{c_1,c_2,D}(G)$ be the subset of paths in $\mathcal{P}_{c_1,c_2,D}(G)$ which are closed. The path closure probability is then,*

$$\gamma(c_1,c_2,D) = \frac{|\mathcal{C}_{c_1,c_2,D}(G)|}{|\mathcal{P}_{c_1,c_2,D}(G)|}$$

Given this statistic, we can define an expression for the cycle closure probability, $P(E_i|E_1,...,E_{i-1})$. Let $\mathcal{S}_{i-1}$ be the set of simple paths in $E_1,...,E_{i-1}$ which start at the source of E_i and end at its destination. Note that the closure of any path within S implies that E_i is true. Based on this, we treat the closure of each of these paths as an independent event (a conservative assumption), and calculate the cycle closure probability as,

$$P(E_i|E_1,...,E_{i-1}) = 1 - \prod_{(c_1,c_2,D)\in\mathcal{S}_{i-1}} (1-\gamma(c_1,c_2,D))$$

We can now explain how we extend the definition of the acyclic estimator (4) to handle arbitrary query graphs. Fix a query graph $Q = (V_Q, E_Q)$, and consider a topological edge ordering, $e_1,...,e_{|E_Q|}$, which means that every edge e_j has a vertex in common with some previous edge e_i, $i < j$. This ordering defines a spanning tree T, consisting of the subset of edges that introduce a new vertex, i.e. $T = \{e_j \mid e_j \not\subseteq \bigcup_{i<j} e_i\}$. If $e_i = (x,y)$ is not a tree edge, then both vertices x, y are already connected in the subgraph consisting of $e_1,...,e_{i-1}$. The modified definition of the estimator (4) is:

$$W(\pi) = \psi(\pi(x_1, \lambda_Q(x_1))) \prod_{e_i \in E_Q} \omega(e_i) \tag{8}$$

$$\omega(e_i) = \begin{cases} \tau(\pi(x),\pi(y),\lambda_Q(x),\chi_Q(x,y))) & \text{if } e=(x,y)\in T \\ 1-\prod_{(c_1,c_2,D)\in\mathcal{S}_{i-1}}(1-\gamma(c_1,c_2,D)) & \text{if } e=(x,y)\notin T \end{cases} \tag{9}$$

To keep the construction of our statistics tractable, we do not calculate γ exactly. Instead, we sample paths from the lifted graph and use these to calculate probabilities. As a default, we use 100,000 sampled paths when calculating these statistics in our experiments. If a particular color combination doesn't occur in our samples, we fall back to the probability just conditioned on the sequence of directions.

6 ALTERNATE COLORING METHODS

Because a coloring can be any mapping from vertices to colors, there is a wide design space of algorithms for creating colorings. The goal of this section is to find colorings which facilitate improved cardinality estimations. In this work, we focus on divisive colorings where all vertices begin in the same color and then the following steps proceed iteratively: 1) identify a color, c, to split into two colors 2) for each vertex in c, determine whether it should stay in c or join the new color. The benefit of this approach is that arbitrary coloring methods can be composed, allowing for more robustness and accuracy.

Quasi-Stable Coloring [19]. As explained earlier, this is a generalization of the traditional color-refinement algorithm. Rather than producing a stable coloring, this algorithm softens the requirements and instead requires vertices in each color to have a "similar" number of edges to each other color. At each iteration, it selects the color with the widest range of degrees w.r.t. another color and splits it into two colors with more uniform counts relative to the other color.

Degree Coloring. This coloring simply separates vertices into colors based on their overall degree. The intuition is that vertices with high degree are generally occupying similar positions in the data graph and vice versa with low degree vertices. It begins by selecting the color with the largest range of degrees to split, then separates vertices into two colors depending on whether they are above or below the average degree.

Neighbor Label Coloring. An alternative to quasi-stable coloring, this method attempts to reduce the variance introduced by label predicates (e.g. "hasLabel:X") by grouping vertices based on their neighbors' label attributes. First, it selects the color whose vertices have the widest range of degrees w.r.t. the vertex labels of their neighbors. It then splits it into two colors which have more uniform connections to vertices with each vertex label.

Vertex Label Coloring. A more direct version of the previous approach, this coloring also aims to reduce the effect of label predicates. This time it aims to make the distribution of vertex labels within colors more uniform. To do this, it first identifies the color, c_1, with the most even distribution of a particular label, weighted by size. The nodes in c_1 which have that label attribute are then put in a new color and the ones which do not remain in c_1.

Mixture Coloring. The previous colorings generally target a particular source of variance related to either topology or attribute distributions, and they divide the color where this kind of variance appears most strongly. So, it makes sense to layer these colorings in order to jointly manage these different concerns, and we call this a mixture coloring. In the experiments (Fig. 9), we show that this is the most accurate coloring across a range of workloads.

Hash Coloring. For completeness, we consider the naive hash coloring which uniformly randomly sorts vertices into colors. This corresponds to the partitioning used by [7] to tighten their cardinality bounds. This method is convenient because construction is linear in the size of $|G|$, and it does not require coordination in a distributed setting. However, it offers limited improvement to the estimator because it doesn't take the specific topology or attributes of the graph into account.

7 OPTIMIZATION

7.1 Partial Aggregation

Naively, the runtime of inference on the lifted graph is exponential in the size of the query graph, making it intractable for even moderately sized query graphs. The lifted graph is dense so the size of $Hom(Q,F)$ is approximately $|C|^{|Q|}$. Fortunately, we can rephrase Eq. 3 to drastically reduce this runtime via aggregate push-down. To do this, we express the set of lifted graph matches, $\hom(Q,F)$, as $(c_1,...,c_{|Q|}) \in C^{|Q|}$. We then use the definition of W from 8 and, as before, we assume a topological ordering on the edges, $e_1,...,e_{|E|}$, and vertices, $v_1,...,v_{|Q|}$.

$$\Phi(Q,F,W_{\psi,\tau}) = \sum_{c_1,...,c_{|Q|}\in C} \psi(c_1) \prod_{i=1}^{|E_Q|} \omega(\pi_{c_1,...,c_{|Q|}}, e_i) \tag{10}$$

Algorithm 1 Optimized Inference Algorithm

```
Require: $F_{G,\sigma,I}(G_F(C,E_V),\psi,\tau)$, $Q(V_Q,E_Q)$, $v_1,...,v_{|V_Q|}$    // Lifted
    Graph, Query Graph, Vertex Order
 1: $PC = \{(\{\},1)\}$ // Partial Colorings
 2: $V_F = \{\}$
 3: for $i \in [1,...,|Q|]$ do
 4:     $PC' = \{\}$
 5:     $E_i = \{e \in E_Q | v_i \in e\}$
 6:     for $\pi, w \in PC$ do
 7:         for $c \in C$ do
 8:             $\pi' = \pi \cup (v_i \to c)$
 9:             $w' = w \cdot \prod_{e \in E_i} \omega(\pi', e)$ ← Estimator Sec. 5.1
10:             $PC' = PC' \cup \{\pi', w'\}$
11:         end for
12:     end for
13:     $PC = PC'$
14:     $V_S = \{v \in V | (v,v_j),(v_j,v) \notin E_Q \forall j > i\} \setminus V_F$
15:     $V_F = V_F \bigcup V_S$
16:     $PC = \sum_{V_S} PC$ ← Partial Aggregation Sec. 7.1
17:     $PC = Sample(PC)$ ← Sampling Sec. 7.2
18: end for
19: return $PC$
```

At this point, we can identify this as a *Functional Aggregate Query* [20] and apply the techniques there to solve it efficiently.[2] As an example, suppose that the query graph Q is a line graph $v_1 \to v_2 \to v_3$. The naive expression would be the following $O(|C|^3)$ expression,

$$\Phi(Q,F,W_{\psi,\tau}) = \sum_{c_1,c_2,c_3 \in C} \psi(c_1)\tau((c_1,c_2))\tau((c_2,c_3))$$

However, by pushing down the summation over c_1, we can produce an expression which requires $O(|C|^2)$ time to evaluate.

$$f(c_2) = \sum_{c_1 \in C} \psi(c_1)\tau((c_1,c_2))$$
$$\Phi(Q,F,W_{\psi,\tau}) = \sum_{c_2,c_3 \in C} \tau((c_2,c_3))f(c_2)$$

This version materializes a a vector of intermediate values and then uses that vector in the second line. Doing this allows us to avoid performing an unnecessary summation over 3 variables at once.

More generally, we can apply this strategy by choosing a variable order and at each step summing out the next variable in the order. The efficacy depends on the maximum number of variables present in any intermediate product. If an intermediate product involves k variables, then we need to compute a relation of size $|C|^k$ which dominates the runtime. We defer the details of the proof to the technical report [1], but this intuition can be formalized as follows using the theory of tree decompositions and treewidth,

Theorem 2. *Given a query graph, Q, a lifted graph, F, and a decomposable estimator W, $\Phi(Q,F,W_{\psi,\tau})$ can be computed in time $O(|C|^{tw(Q)+1})$ where $tw(Q)$ is the treewidth of Q.*

Of course, this relies on finding a good ordering of the query vertices, and finding the optimal one is naively NP-Hard with respect to the size of Q. Fortunately, there are very effective heuristics for identifying good tree decompositions, and we apply these in our system, using the min-fill heuristic [32]. To accommodate our sampling techniques, we restrict these tree decompositions to path decompositions and get results relative to the pathwidth.

[2]Note, this is closely related to the variable elimination algorithm for probabilistic graphical models as well as tensor contraction algorithms.

7.2 Sampling Techniques

In real systems, cardinality estimation needs to be extremely fast and consistent because its overhead is seen by every query. While partial aggregation speeds up inference for simple query graphs with low treewidth, larger and denser query graphs still pose a problem. To avoid this, we use a sampling procedure that ensures linear inference w.r.t query size (see Fig. 13). This is similar to a weighted version of the WanderJoin algorithm from [23]. We apply a Thompson-Horowitz estimator to random paths within the lifted graph. However, we adjust the method in two important ways: 1) we incorporate the sampling into the aggregation framework from Sec. 7.1 2) we apply importance sampling to account for the fact that different paths within the lifted graphs contribute more or less to the cardinality estimate.

Sampling During Aggregation. At each step, we process a vertex of the query graph and materialize an intermediate result consisting of partial colorings and their weights. After materialization, we apply sampling in order to reduce the amount of partial colorings that we extend in the next step. By doing this at each step, we can maintain a constant number of partial colorings at all times and ensure a linear runtime w.r.t. the size of the query graph. Because we still apply aggregation, we reduce the amount of sampling required.

Importance Sampling. This is a classical technique for approximating the value of an integral, and we adapt it here by noting that our summation in 10 is a discrete integral over a product. The core idea is to sample points of the integrand which contribute more heavily to the result with higher probability in order to reduce the variance. Because determining the contribution of a partial coloring to the final result is challenging, we approximate this contribution via its partial count. This assumes that a high partial count leads to a high contribution to the final sum. To keep our estimator unbiased, we apply Thompson-Horowitz estimation and scale each sampled partial count by the inverse of its selection probability. Finally, we scale the total sampled weight to set it to the total weight prior to sampling.

7.3 Handling Updates

Updates pose a challenge to summary-based estimators because the statistics which they collect become stale over time as updates are applied to the database. Traditional estimators simply rebuild the summary intermittently[16]. This leads to severe decreases in accuracy under even modest updates because the estimator is entirely "blind" to them. On the other hand, recalculating the summary before each query is very costly. In this section, we demonstrate how COLOR supports a middle ground approach that applies fast, basic updates to the lifted graph, allowing it to maximize the time between full rebuilds.

First, we formally define updates in our setting,

Definition 10. *Given a data graph G, an update θ can either add an edge between existing vertices or a new vertex with attributes A:*

- $\theta_V = (v,A)$ *where* $v \in G_V, A \in \mathbb{A}$
- $\theta_E = (v_1,v_2,A)$ *where* $v_1 \in G_V, v_2 \in G_V, A \in \mathbb{A}$

Table 2: Estimator Failure Rates per dataset. Only Color and Characteristic Sets (cset) succeed on all queries. Later we will see Color outperforms cset's accuracy substantially.

Dataset\Method	cs	wj	jsub	impr	cset	alley	alleyTPI	LSS	BSK++	sumrdf	Color
human	0.67	**0.00**	0.22	0.63	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
aids	0.69	0.07	0.14	0.28	**0.00**	0.04	0.01	**0.00**	0.02	0.39	**0.00**
lubm80	0.83	0.17	0.67	0.67	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
yeast	1.00	0.97	0.97	0.11	**0.00**	0.63	0.60	**0.00**	0.63	0.88	**0.00**
dblp	1.00	0.99	0.94	0.15	**0.00**	0.14	0.14	**0.00**	0.70	0.85	**0.00**
youtube	0.99	0.93	1.00	0.22	**0.00**	0.10	0.05	**0.00**	0.63	0.78	**0.00**
eu2005	0.95	0.90	0.91	0.55	**0.00**	**0.00**	**0.00**	**0.00**	0.22	0.44	**0.00**
patents	0.98	0.88	0.98	0.08	**0.00**	0.13	0.13	**NA**	0.67	0.79	**0.00**

This definition allows for adding a single edge or vertex to the graph at a time. We then define a summary update function to incorporate these updates without accessing G.

Definition 11. *Given a data graph G, a lifted graph $F=(G_F,\psi,\tau)$, and update θ, we define the summary update function as follows where F' is the updated lifted graph,*

$$F' = \delta(F,\theta)$$

Depending on the type of θ, the functionality of δ can change:

$$\delta = \begin{cases} \delta_V, & \theta \in \theta_V \\ \delta_E, & \theta \in \theta_E \end{cases}$$

Depending on the estimator, the correct definition of δ will change. Here, we focus on the average degree estimator.

Vertex Updates. The vertex update function δ_V affects the stored edge statistics τ and color sizes ψ in the lifted graph. Because a new vertex has no edges, we have no knowledge about which color it should be placed once its edges are added. Conservatively, we simply add it to the largest existing color which dilutes the impact on the average degree. In this way, we preserve the high quality information in other colors while the largest color gracefully degrades to a traditional estimator as in Def. 4. After choosing the color for the new vertex, we adjust $\psi(c)$ by incrementing its value by one, and we scale down τ to account for the new vertex.

Edge Updates. Given an update $\theta_E = (v_1,v_2,A)$, the edge update function δ_E marginally increases τ for the combination of attributes and colors in the edge update. However, the edge update doesn't directly contain the colors of v_1 and v_2 or the attributes of v_2. To retrieve the colors, c_1 and c_2, associated with v_1 and v_2, we look up their values in π which we store compactly (and approximately) as a series of cuckoo filters. To calculate the attributes of v_2, we leverage statistics about the attribute distribution in c_2 to sample a set of attributes.

Path Closure Probabilities. The lifted graph contains statistics about the cycle-closing probability for existing nodes and edges, but additions to the graph change this probability. To account for this, we make an adjustment to the ω°_{CCP} function. We calculate the probability that either the path was originally closed, $\gamma(c_1,c_2,D)$, or is closed by an update edge. For a set of edge updates, $\mathbb{S}_{\theta E}$:

$$\gamma'(c_1,c_2,D) = 1-(1-\gamma(c_1,c_2,D))(1-\frac{|\mathbb{S}_{\theta E}|}{|V_F|^2})$$

Table 3: Experimental Datasets. $|\ell_V|$ and $|\ell_E|$ are the number of unique vertex and edge labels.

Dataset	|V|	|E|	$\lvert\ell_V\rvert$	$\lvert\ell_E\rvert$
human	4674	86282	89	1
aids	254000	548000	50	4
lubm80	2.6M	12.3M	35	35
yeast	3112	12519	71	1
dblp	317080	1M	15	1
youtube	1.1M	3M	25	1
eu2005	862664	16M	40	1
patents	3.8M	16.5M	20	1

Deletions. To handle deletions, COLOR simply performs the inverse of the update logic.

8 EVALUATION

In this section, we provide a detailed experimental analysis of our framework on eight benchmark datasets and against nine competitive baselines. Compared to existing methods, COLOR exhibits competitive accuracy, speed, and scalability. We also demonstrate the significance of our performance optimizations for building and maintaining graph summaries. Overall, we show that COLOR:

(1) Never experiences estimation failure on any query across all workloads.
(2) Has a median error $<\mathbf{10}$ across all workloads and is up to $\mathbf{10^3}\times$ more accurate than competing methods.
(3) Produced up to $\mathbf{10^7}\times$ tighter bounds when using τ_{max} than BoundSketch while being many orders of magnitude faster.
(4) Requires up to $\mathbf{80-800}\times$ less space than competing methods and is up to $\mathbf{10-100}\times$ faster to construct.
(5) Handles updates gracefully with only $\mathbf{3}\times$ worse median error when half the graph is updated.

Datasets & Workload. We consider datasets from [29] and [37] for our analysis. These datasets come from a variety of domains. Those from [37] are undirected graphs whose queries are larger and vary in density. Those from [29] are directed graphs whose queries are smaller and vary in shape. We adapt undirected workloads to directed methods by including reverse edges in the data graph but not in the query graphs. Both of these sources include label predicates in their queries with the former using both edge and vertex labels and the

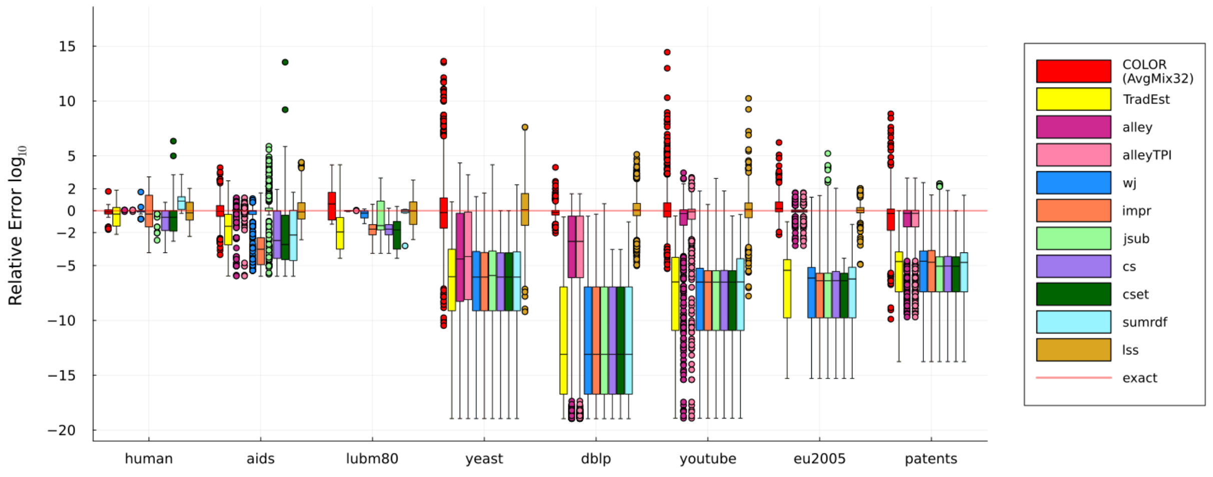

Figure 3: Relative Error by Estimator

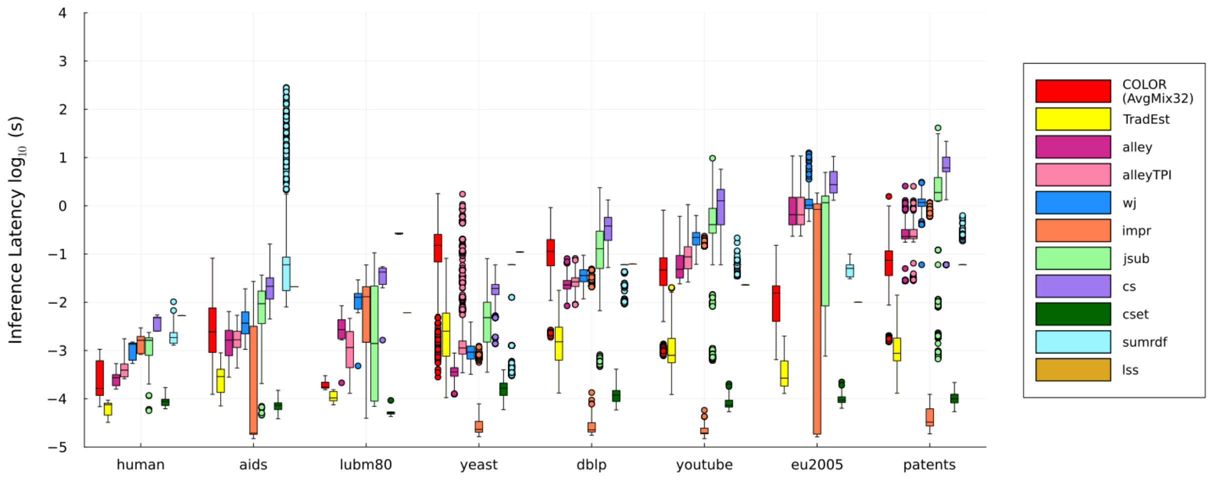

Figure 4: Inference Time by Estimator

latter using only vertex labels. Table 3 shows their different characteristics where $|\ell_V|$ and $|\ell_E|$ are the number of edge and vertex labels.

Comparison Methods. For comparison, we use a superset of the methods considered in [29] and additionally apply them to the larger, more complex datasets from [37]. These methods include: 1) Correlated Sampling (CS) [38] 2) Characteristic Sets (CSet)[28] 3) Wander Join (WJ) [23] 4) Alley (alley) and alleyTPI [21] 5) Join Sampling with Upper Bounds (JSUB), an adaptation of [43] 6) Bound Sketch (BSK) [7] which corresponds to a hash-based coloring and using the max degree estimator. We further apply our partial aggregation optimization, and we call this improved version BSK++. 7) IMPR [8] and 8) SumRDF [36] 9) Learned Sketch for Subgraph Counting (LSS) [42], a query-driven deep learning approach. Due to hardware compatibility issues, this method was run on a different machine than the others with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz CPU. This machine had 32GB of memory, resulting in an OOM error for the patents dataset which we treat as NA rather than estimator failure. 10) We also include a traditional independence-based estimator (IndEst) corresponding to Def. 4. For the sampling based estimators, we apply the default sampling ratios from [29] and [21] (i.e. .03 for all methods except for Alley which uses .001). AlleyTPI uses a maximum pattern length (MAX_L) and a maximum number of stored label groups (NUM_GROUPS) when building its index, with default values of MAX_L=4 or MAX_L=5 depending on the dataset, and NUM_GROUPS=32. To prevent excessive index build times on AlleyTPI (> 12 hours) for eu2005, dblp, and patents, we used MAX_L=4. We decreased NUM_GROUPS to 16 for eu2005 and dblp and 8 for patents. We adjusted NUM_GROUPS more than MAX_L because small adjustments to the maximum pattern length greatly decrease the domain of patterns stored by the index [21]. As in the

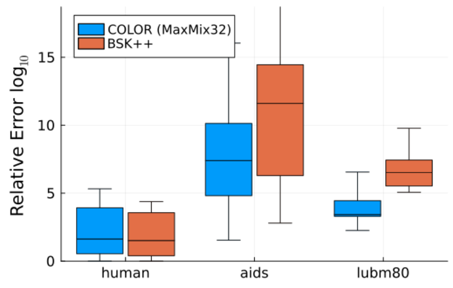

Figure 5: Relative Error by Cardinality Bound Method

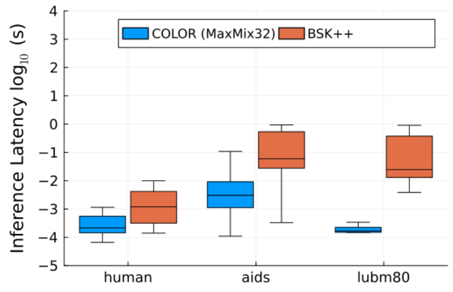

Figure 6: Inference Time by Cardinality Bound Method

original work, we get estimates for LSS via a 5-fold cross-validation where 4/5ths of the queries are trained on and 1/5 is estimated during each fold. Lastly, recently, there has been work on the isomorphism variant of subgraph cardinality estimation from both the deep learning and sampling perspective[35, 39]. Unfortunately, this difference in problem setting makes them incomparable with the other methods considered here, so we exclude them from our evaluation.

Additionally, we experiment with several instantiations of our framework which use the following naming convention; first, we note the kind of degree statistic (Min/Avg/Max), then we describe the coloring scheme, e.g. $Q64$ as 64 colors from the quasi-stable coloring method. The mixed coloring scheme, $Mix32$, that we use as the default involves 8 divisions from degree coloring, quasi-stable coloring, neighbor labels coloring, and node label coloring, in that order. Unless otherwise noted, we use 500 samples during inference and keep track of cycle probabilities for cycles up to length 6.

Experimental Setup. To reduce noise, we repeat all inference results 3 times and report the median inference time. We do not do this for our cardinality estimates because this would unfairly reduce the impact of our sampling approach. These experiments are run on a server with an Intel(R) Xeon(R) CPU E7-4890 v2 @ 2.80GHz CPU, and all summary building and inference is done using a single thread, unless stated otherwise. The reference implementation is available at: https://github.com/uwdb/color

8.1 Estimator Failure

There are two ways that an estimator can fail to provide meaningful results: 1) time outs, which we define as taking longer than 1 minute to report a result 2) sampling failure, not finding any qualifying samples. In Table 2, we show the proportion of queries that result in estimation failure for each dataset and technique. Simpler sampling-based methods (CS, WJ, JSUB, IMPR) face estimation failure even on the smaller query workloads and fail to find any samples most queries on the larger workloads. Alley achieves much higher success rates due to its sophisticated sampling approach but still fails a significant portion of the time on four datasets. Summary-based methods (BSK++ and SumRDF), on the other hand, time out on over half of queries for four datasets. In contrast, because our approach applies sampling to a highly dense lifted graph, we never experience sampling failure on any query, across all workloads. As the data graph contains hundreds of thousands of edges or more, which are mapped into a lifted graph with at most 32 nodes (colors), the probability that any two colors have an edge connecting them is high.

8.2 Accuracy

In Fig. 3, we show the relative error of various methods and workloads[3]. Relative error is defined as the ratio of the estimate to the true cardinality. Scores greater than one indicate an overestimate, those under an underestimate. Across workloads, our method, AvgMix32, is unbiased and scales well to larger, more cyclic workloads (yeast, dblp, youtube, eu2005, patents). It even achieves a median error of less than 2 on human, aids, dblp, and eu2005. We also reproduce the high accuracy of WanderJoin and Alley on the G-Care datasets. However, we find that all methods from [29] fail to scale to the larger more cyclic workloads. In particular, we reproduce the finding in [21] that WanderJoin, IMPR, and JSUB overwhelmingly fail to find a positive sample in a reasonable time on these datasets, and that LSS achieves reasonable accuracy on a variety of workloads. Further, SumRDF times out on all larger queries.

Cardinality Bounds. When comparing the cardinality bounding methods, BSK and MaxQ64, we find that applying a mixture of coloring methods rather than hash coloring produces up to 10^6 times lower error. Notably, because we apply sampling to this method, we guarantee a linear runtime for all queries in exchange for a less principled cardinality bound. However, across all workloads, this never results in significant underestimation.

Coloring Methods. In Fig. 9, we examine the effect of choosing different colorings on the accuracy of the average degree estimator. The hash coloring performs the worst across all benchmarks which is expected because it does not take the labels or graph topology into account. On the other hand, the quasi-stable coloring algorithm from [19] works quite well on most datasets with the exception of dblp because it does not account for the distribution of labels. Overall, the mixed coloring performs well across datasets because it can supplement the topological colorings with a label-based colorings, accounting for both sources of error.

In this figure, we also vary the number of colors that we use for the lifted graph. Interestingly, increasing the number of colors kept does not straightforwardly improve accuracy. This is because we always use 500 samples during inference. As the lifted graph gets

[3]In these graphs, outliers (>2 std. deviations) are shown as points. The inner box shows quartiles, and the whiskers are the max/min non-outlier values. Further, sample failure is treated as an estimate of 1.

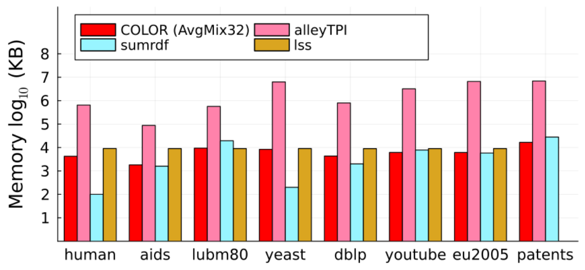

Figure 7: Statistics Size

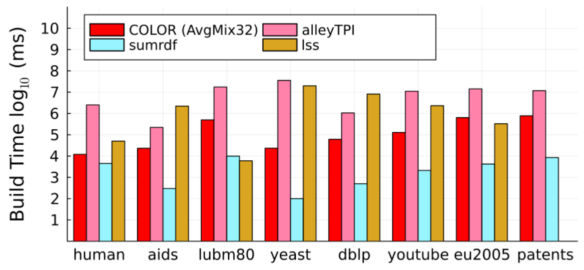

Figure 8: Build Time

larger, the sampling procedure has a larger impact. Given this, we find that the optimal coloring uses either 32 or 64 colors.

8.3 Inference Latency

Fig. 4 shows the distribution of inference latencies for each method across workloads. We can see that the inference latency of COLOR lies in the middle of the competing methods across workloads. On the smaller, less cyclic queries of human, it achieves a median latency of around 10^{-4} seconds due to partial aggregation, and on the more complex queries of patents it has a median latency of $\sim .05$ seconds via sampling. Similar to prior work, we select a one minute timeout because cardinality estimators may be called many times during the query planning phase [21, 30, 40, 42].

Compared to other graph summarization methods, SumRDF and BSK++, our framework scales far better to larger queries. The former methods timeout on queries of even moderate size as they consider the exponential number of potential colorings of the query. This occurs even when using partial aggregation (as in BSK++), demonstrating the necessity of sampling to achieve consistent latencies.

8.4 Statistics Size & Build Time

Graph summarization approaches allow for a smooth tradeoff between accuracy and size/build time; a more granular summary of the graph will take more space but more accurately capture the structure of the data graph. Even a compact summary ($<$ 20MB) can be highly accurate as shown by Fig. 7. This comes from the fact that we store our summary sparsely. If two colors do not share an edge with a particular attribute, then we don't explicitly store any degree statistic about this combination. A better coloring can actually result in a more compact summary because many of these combinations won't occur if the nodes have been properly partitioned. On Human and Lubm80, AlleyTPI's pattern index requires 600 and 300 MB, respectively. LSS takes a considerable time to train across all datasets. This is the average training time over all folds of the cross validation, and, crucially, it does not include the time necessary to collect the training data (i.e. to run the queries and calculate a true cardinality).

With respect to build time, our summary construction scales linearly in the size of the data graph. For smaller graphs like human, aids, and yeast, summary construction takes less than 20 seconds, and it smoothly increases as the data graph gets larger. In Fig. 10, we confirm this by generating erdos-renyi graphs of varying sizes and recording the average time to build the lifted graph over 20 trials.

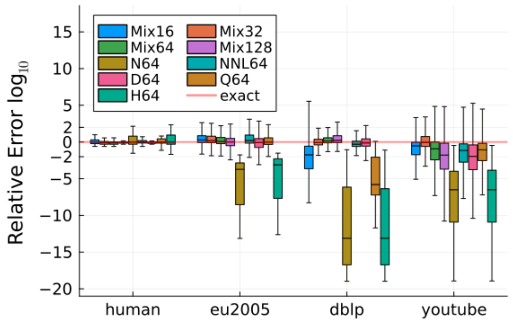

Figure 9: Relative Error by Coloring Method

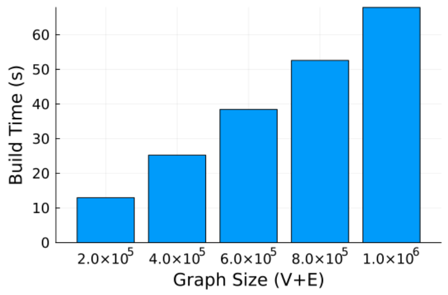

Figure 10: Construction Scaling

8.5 Updates

In Fig. 11, we evaluate the effectiveness of our method for handling updates (Sec. 7.3). To do this, we randomly partition the edges of the human dataset into an initial data graph and an ensuing set of updates, and we construct a lifted graph using the former then update it by adding one edge/vertex at a time based on the latter. Intuitively, when more of the graph is provided at the beginning, the lifted graph will be more accurate because it can take advantage of that knowledge when coloring the graph. We find that, as more of the lifted graph is updated, the accuracy remains very consistent and degrades to the traditional independence estimator. So, a full rebuilding of the lifted graph can occur very infrequently and be amortized over many updates. Due to the small size of the lifted

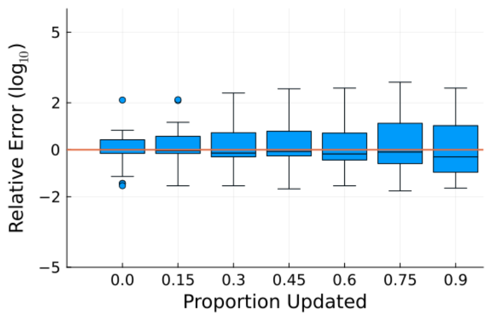

Figure 11: Relative Error vs Proportion Updated (Human)

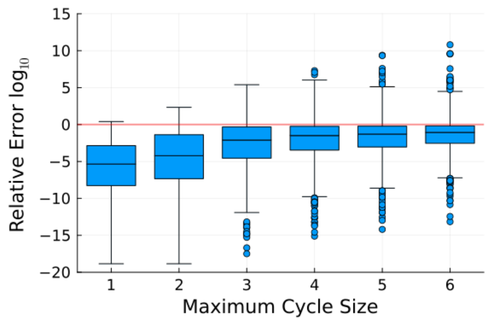

Figure 12: Relative Error vs Max Cycle Length Stored (Youtube)

graph, each update is very fast. The latency for vertex updates was roughly 0.4 ms while for edge updates it was roughly 0.1 ms.

8.6 Micro-Benchmarks

Path Closure Probabilities. To show the importance of tracking cycle probabilities, we show the relative error as we vary the size of cycles whose probabilities we track in Fig. 12. At size 1, we do not track any cycle closure probabilities. So, when we close a cycle in the query graph, we scale down the estimate based on the uniform probability of an edge existing, i.e. $|E|/|V|^2$. When we begin to store larger cycle sizes, we quickly see the error decrease, and underestimation, in particular, is significantly reduced. These results validate the necessity of handling cycle closure with a more complex method than the standard independence estimator.

Partial Aggregation. Fig. 13 demonstrates the effects of partial agg. and sampling. Using the Youtube dataset, we study the effect of partial agg. and sampling on inference latency across queries with different pathwidths. Recall that pathwidth is a measure of cyclicity where pathwidth 1 is acyclic and higher pathwidth queries are denser. Without partial agg., estimation times out (>1 min) when pathwidth exceeds 2. Higher pathwidth queries are larger and the naive approach is exponential w.r.t. the query size. Using only partial agg., estimation times out when pathwidths exceed 4. Lastly, when sampling is applied, the inference latency becomes linear in the size of the query and unrelated to the pathwidth. The results demonstrate that partial agg. achieves speedups without affecting accuracy and sampling can achieve consistent, fast inference.

Inference Sampling. In Fig. 14, we vary sample size to demonstrate the efficacy of our sampling method. Using a very small sample results in significant underestimation, but even a moderate number of samples quickly converges to the accuracy of a large number of samples. Further, this figure shows that importance sampling speeds up this convergence significantly over a naive uniform sample. For example, the performance at 250 samples for importance sampling is roughly equal to the accuracy of uniform sampling at 1000 samples.

9 CONCLUSION

We develop COLOR, a framework for producing lifted graph summaries from colorings. We define inference over the lifted graph for acyclic and cyclic queries, developing optimizations to accelerate estimation. We empirically validate COLOR's superior performance on eight benchmarks and compare to state-of-the-art methods. COLOR is up to $10^3\times$ more accurate than the baselines and stands out for never experiencing estimation failure. It gracefully handles updates, degrading to only 3× worse error when half of the data is replaced.

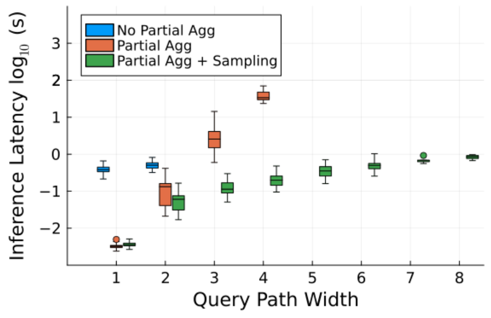

Figure 13: Inference Time vs Query Pathwidth (Youtube)

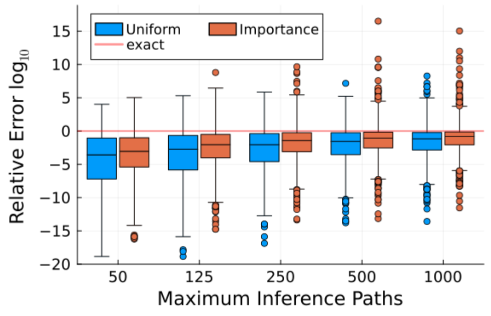

Figure 14: Relative Error vs Samples (Youtube)

ACKNOWLEDGMENTS

This work was partially supported by NSF-BSF 2109922, NSF IIS 2314527, and a gift from Amazon through the UW Amazon Science Hub.

REFERENCES

[1] 2024. *COLOR Tech Report & Repository.* Technical Report. https://anonymous.4open.science/r/Cardinality-with-Colors-4333/README.md

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

[3] Hannah Bast and Björn Buchhold. 2017. QLever: A Query Engine for Efficient SPARQL+Text Search. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, Ee-Peng Lim, Marianne Winslett, Mark Sanderson, Ada Wai-Chee Fu, Jimeng Sun, J. Shane Culpepper, Eric Lo, Joyce C. Ho, Debora Donato, Rakesh Agrawal, Yu Zheng, Carlos Castillo, Aixin Sun, Vincent S. Tseng, and Chenliang Li (Eds.). ACM, 647–656. https://doi.org/10.1145/3132847.3132921

[4] Bradley R. Bebee, Daniel Choi, Ankit Gupta, Andi Gutmans, Ankesh Khandelwal, Yigit Kiran, Sainath Mallidi, Bruce McGaughy, Mike Personick, Karthik Rajan, Simone Rondelli, Alexander Ryazanov, Michael Schmidt, Kunal Sengupta, Bryan B. Thompson, Divij Vaidya, and Shawn Wang. 2018. Amazon Neptune: Graph Data Management in the Cloud. In *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018 (CEUR Workshop Proceedings)*, Marieke van Erp, Medha Atre, Vanessa López, Kavitha Srinivas, and Carolina Fortuna (Eds.), Vol. 2180. CEUR-WS.org. https://ceur-ws.org/Vol-2180/paper-79.pdf

[5] Maciej Besta, Robert Gerstenberger, Emanuel Peter, Marc Fischer, Michal Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. 2024. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *ACM Comput. Surv.* 56, 2 (2024), 31:1–31:40. https://doi.org/10.1145/3604932

[6] Angela Bonifati, Wim Martens, and Thomas Timm. 2020. An analytical study of large SPARQL query logs. *VLDB J.* 29, 2-3 (2020), 655–679. https://doi.org/10.1007/S00778-019-00558-9

[7] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 18–35. https://doi.org/10.1145/3299869.3319894

[8] Xiaowei Chen and John C. S. Lui. 2018. Mining Graphlet Counts in Online Social Networks. *ACM Trans. Knowl. Discov. Data* 12, 4 (2018), 41:1–41:38. https://doi.org/10.1145/3182392

[9] Kyle B Deeds, Dan Suciu, and Magdalena Balazinska. 2023. SafeBound: A Practical System for Generating Cardinality Bounds. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[10] Alin Deutsch, Yu Xu, Mingxi Wu, and Victor E. Lee. 2019. TigerGraph: A Native MPP Graph Database. *CoRR* abs/1901.08248 (2019). arXiv:1901.08248 http://arxiv.org/abs/1901.08248

[11] Orri Erling and Ivan Mikhailov. 2009. Virtuoso: RDF Support in a Native RDBMS. In *Semantic Web Information Management - A Model-Based Perspective*, Roberto De Virgilio, Fausto Giunchiglia, and Letizia Tanca (Eds.). Springer, 501–519. https://doi.org/10.1007/978-3-642-04329-1_21

[12] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 international conference on management of data.* 1433–1445.

[13] Martin Grohe. 2017. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory.* Lecture Notes in Logic, Vol. 47. Cambridge University Press. https://doi.org/10.1017/9781139028868

[14] Martin Grohe and Daniel Neuen. 2020. Recent Advances on the Graph Isomorphism Problem. *CoRR* abs/2011.01366 (2020). arXiv:2011.01366 https://arxiv.org/abs/2011.01366

[15] Martin Grohe and Pascal Schweitzer. 2020. The graph isomorphism problem. *Commun. ACM* 63, 11 (2020), 128–134. https://doi.org/10.1145/3372123

[16] Laura M. Haas. 1999. Review - Access Path Selection in a Relational Database Management System. *ACM SIGMOD Digit. Rev.* 1 (1999). https://dblp.org/db/journals/dr/Haas99a.html

[17] László Hajdu and Miklós Krész. 2020. Temporal Network Analytics for Fraud Detection in the Banking Sector. In *ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium - International Workshops: DOING, MADEISD, SKG, BBIGAP, SIMPDA, AIMinScience 2020 and Doctoral Consortium, Lyon, France, August 25-27, 2020, Proceedings (Communications in Computer and Information Science)*, Ladjel Bellatreche, Mária Bieliková, Omar Boussaïd, Barbara Catania, Jérôme Darmont, Elena Demidova, Fabien Duchateau, Mark M. Hall, Tanja Mercun, Boris Novikov, Christos Papatheodorou, Thomas Risse, Oscar Romero, Lucile Sautot, Guilaine Talens, Robert Wrembel, and Maja Zumer (Eds.), Vol. 1260. Springer, 145–157. https://doi.org/10.1007/978-3-030-55814-7_12

[18] Olaf Hartig and Jorge Pérez. 2018. Semantics and complexity of GraphQL. In *Proceedings of the 2018 World Wide Web Conference.* 1155–1164.

[19] Moe Kayali and Dan Suciu. 2022. Quasi-stable Coloring for Graph Compression: Approximating Max-Flow, Linear Programs, and Centrality. *Proc. VLDB Endow.* 16, 4 (2022), 803–815. https://www.vldb.org/pvldb/vol16/p803-kayali.pdf

[20] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. 2016. FAQ: Questions Asked Frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Tova Milo and Wang-Chiew Tan (Eds.). ACM, 13–28. https://doi.org/10.1145/2902251.2902280

[21] Kyoungmin Kim, Hyeonji Kim, George Fletcher, and Wook-Shin Han. 2021. Combining Sampling and Synopses with Worst-Case Optimal Runtime and Quality Guarantees for Graph Pattern Cardinality Estimation. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 964–976. https://doi.org/10.1145/3448016.3457246

[22] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215.

[23] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2019. Wander Join and XDB: Online Aggregation via Random Walks. *ACM Trans. Database Syst.* 44, 1 (2019), 2:1–2:41. https://doi.org/10.1145/3284551

[24] Tianyu Liu and Chi Wang. 2020. Understanding the hardness of approximate query processing with joins. *arXiv preprint arXiv:2010.00307* (2020).

[25] Wim Martens and Tina Trautner. 2019. Bridging Theory and Practice with Query Log Analysis. *SIGMOD Rec.* 48, 1 (2019), 6–13. https://doi.org/10.1145/3371316.3371319

[26] Christopher Morris, Yaron Lipman, Haggai Maron, Bastian Rieck, Nils M. Kriege, Martin Grohe, Matthias Fey, and Karsten M. Borgwardt. 2021. Weisfeiler and Leman go Machine Learning: The Story so far. *CoRR* abs/2112.09992 (2021). arXiv:2112.09992

[27] Inc. Neo4j. 2007. https://neo4j.com/

[28] Thomas Neumann and Guido Moerkotte. 2011. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan (Eds.). IEEE Computer Society, 984–994. https://doi.org/10.1109/ICDE.2011.5767868

[29] 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 *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1099–1114. https://doi.org/10.1145/3318464.3389702

[30] 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 *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* 1099–1114.

[31] Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. Real-time Constrained Cycle Detection in Large Dynamic Graphs. *Proc. VLDB Endow.* 11, 12 (2018), 1876–1888. https://doi.org/10.14778/3229863.3229874

[32] Emma Rollon and Javier Larrosa. 2011. On Mini-Buckets and the Min-fill Elimination Ordering. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings (Lecture Notes in Computer Science)*, Jimmy Ho-Man Lee (Ed.), Vol. 6876. Springer, 759–773. https://doi.org/10.1007/978-3-642-23786-7_57

[33] 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. *VLDB J.* 29, 2-3 (2020), 595–618. https://doi.org/10.1007/s00778-019-00548-x

[34] Muhammad Saleem, Gábor Szárnyas, Felix Conrads, Syed Ahmad Chan Bukhari, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2019. How Representative Is a SPARQL Benchmark? An Analysis of RDF Triplestore Benchmarks. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia (Eds.). ACM, 1623–1633. https://doi.org/10.1145/3308558.3313556

[35] Wonseok Shin, Siwoo Song, Kunsoo Park, and Wook-Shin Han. 2024. Cardinality Estimation of Subgraph Matching: A Filtering-Sampling Approach. *Proc. VLDB Endow.* 17, 7 (2024), 1697–1709. https://www.vldb.org/pvldb/vol17/p1697-park.pdf

[36] Giorgio Stefanoni, Boris Motik, and Egor V. Kostylev. 2018. Estimating the Cardinality of Conjunctive Queries over RDF Data Using Graph Summarisation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 1043–1052. https://doi.org/10.1145/3178876.3186003

[37] Shixuan Sun and Qiong Luo. 2020. In-memory subgraph matching: An in-depth study. In *Proceedings of the 2020 ACM SIGMOD International Conference on*

Management of Data. 1083–1098.

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

[39] Hanchen Wang, Rong Hu, Ying Zhang, Lu Qin, Wei Wang, and Wenjie Zhang. 2022. Neural Subgraph Counting with Wasserstein Estimator. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 160–175. https://doi.org/10.1145/3514221.3526163

[40] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are We Ready For Learned Cardinality Estimation? *Proc. VLDB Endow.* 14, 9 (2021), 1640–1654.

[41] Xin Wang, Eugene Siow, Aastha Madaan, and Thanassis Tiropanis. 2018. PRESTO: probabilistic cardinality estimation for RDF queries based on subgraph overlapping. *arXiv preprint arXiv:1801.06408* (2018).

[42] Kangfei Zhao, Jeffrey Xu Yu, Qiyan Li, Hao Zhang, and Yu Rong. 2023. Learned sketch for subgraph counting: a holistic approach. *VLDB J.* 32, 5 (2023), 937–962. https://doi.org/10.1007/S00778-023-00781-5

[43] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 1525–1539. https://doi.org/10.1145/3183713.3183739