COLOR : A Framework for Applying Graph Coloring to Cardinality Estimation

Anonymous Author(s)

ABSTRACT

Graph workloads pose a particularly challenging problem for query optimizers in modern databases. While theoretically equivalent to traditional relational joins, they typically feature larger queries, more FK-FK joins, and fewer filter predicates (e.g. $R.X < 10$). This puts significant stress on traditional cardinality estimation methods which generally see catastrophic errors when estimating the size of queries with a handful of joins (~5). On the other hand, machine learning approaches have been proposed which often provide accurate inference at the cost of inference latency, summary construction time, and complexity. To overcome these problems, we propose a framework, COLOR, for improving traditional estimators by incorporating theoretical insights from the graph coloring literature to produce a compact summary of the data graph. Further, we identify several key optimizations that enable tractable estimation over this summary even for large query graphs ($\geq$ 30 vertices). We then evaluate several designs within this framework and find that they improve accuracy by up to $10^2\times$ over 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

ACM Reference Format:

Anonymous Author(s). 2018. COLOR : A Framework for Applying Graph Coloring to Cardinality Estimation. In *Proceedings of Special Interest Group on the Management of Data (SIGMOD '25).* ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The core operation of graph databases is *subgraph matching* where instances of a small query graph, Q, are found within a larger data graph, G. In theory, this operation is equivalent to joins in a traditional relational database system, but the "typical" workloads are quite different. In graph databases, large join queries with tens of tables are common and complex selection predicates are either rare or disallowed. These larger queries make join ordering and query planning particularly crucial and challenging. Further, recent work has highlighted the necessity of accurate cardinality estimates for effective query planning [5, 14, 18].

However, cardinality estimation techniques are required to satisfy a challenging set of metrics in order to be used in a practical system: 1) They must be accurate enough to distinguish good plans from bad ones 2) They must be fast enough (~ms) to be run hundreds of times during query planning 3) They must have a small enough memory footprint to not restrict the usage during query evaluation (~MBs) 4) Their statistics must be either fast to update (~ms) or very fast to reconstruct (~sec). Correctly balancing these four metrics is necessary to produce an estimator that can function in a real system.

Prior work on this problem can generally be categorized into pattern-based summaries, online sampling techniques, and machine learning approaches. The former generally count the frequency of a small set of pattern graphs offline then combine these frequencies at estimation time to calculate the frequency of a larger query graph. [17, 23] are examples of this approach, and for small queries they are generally fast and accurate. However, as the size of the pattern graphs grows (and the number of labels grows), the size of their statistics increases exponentially. This makes it challenging for these methods to handle larger query graphs whose size can't be effectively inferred from small pattern graphs.

Online sampling methods such as [3, 12, 15] instead use various techniques to sample from the database at inference time. The most effective methods of this kind apply optimizations to infer cardinalities from random walks on the data graph. Generally, this requires pre-computing indices on the data graph to guide the random walk more efficiently. While these methods are often quite accurate, directly obtaining samples from the database can be an expensive operation, becoming intractable in distributed or disk-based settings. Further, these methods are prone to sampling failure (i.e. not finding any valid samples) on more complex or selective queries which poses a challenge for adoption.

Recently, many machine learned approaches have been proposed in both the relational and graph setting to handle cardinality estimation. Query-driven methods like [13, 25] use a set of query graphs and their exact cardinalities to train a model while data-driven methods like [24, 26] aim to identify patterns in the data graph. Though these methods often achieve good accuracy, they are generally slow to train and slow to perform inference on. Further, query-driven methods require the execution of a large number of expensive subgraph counting queries to produce their training data.

In this paper, we propose a new approach based on graph summarization; we call this method COLOR. During the offline phase, our method takes the data graph and produces a compact summary, called a *lifted graph*, which captures the topology (Sec. 3). This is done by grouping vertices with similar local topological properties into *colors*, then recording information about the number and kind of edges which pass between these colors. During the

A note.

SIGMOD '25, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

online phase, the cardinality estimate is computed on the (much smaller) lifted graph by performing a weighted version of subgraph counting which we call *lifted subgraph counting* (Sec. 4). In order to handle 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).

The primary sources of error in traditional cardinality estimation are the uniformity and independence assumptions. By grouping nodes with similar local topology, our method reduces the errors introduced by these assumptions. For example, consider a large graph where most nodes have small degrees, say 2-3, with the exception of a rather large clique, where the nodes have high degrees, say 100. The simplest lifted graph partitions these nodes into two colors, consisting of the nodes in the clique (red), and the nodes outside the clique (blue). This helps mitigate the uniformity assumption because the degree of nodes in each color is closer to uniform. It also mitigates the independence assumption as it captures the high frequency of edges between high-degree red nodes. A real world graph is more complex, but our experiments show that we can meaningfully capture the topology with a relatively small number of colors, typically just 32. This ensures that the lifted graph has a small memory footprint (<10 MB).

COLOR offers a trade-off between accuracy and latency. By using more colors, the lifted graph better captures the topology of the data graph and produces more accurate cardinality estimates, but it then requires doing lifted subgraph counting over a larger input graph. To improve this trade-off and enable efficient inference with more colors, we propose several critical optimizations: 1) Tree-decompositions and partial aggregation (Sec. 7.1) 2) Importance sampling and Thompson-Horowitz estimation (Sec. 7.2) 3) Maintaining the lifted graph under updates (Sec. 7.3). The former reduces the inference latency by up to **100x** on smaller queries, while preventing timeouts on large ones. Importance sampling allows for linear latency w.r.t. the size of the query while maintaining a **6x** lower median q-error than uniform sampling. Lastly, our update method allows for graceful degradation under updates, greatly reducing the need to rebuild the summary and providing consistent estimates even when over $\frac{1}{2}$ of the graph is updated.

While the idea of applying graph summarization to cardinality estimation has been discussed in the literature, to the best of our knowledge COLOR is the first system to 1) apply summarization based on the local topology of the nodes and 2) make this approach scale beyond small query graphs. SumRDF [20] summarizes RDF graphs based on the types of the RDF resources, then compresses the summary using Jaccard similarity, but it does not account for the degrees of nodes or their structure. BoundSketch [2] uses a randomized hash partitioning: as we show in Sec. 8, random partition has a much poorer accuracy/performance tradeoff than the toplogy-based partitioning in COLOR. Unlike previous approaches, COLOR partitions the nodes based on a combination of their topology and labels through the use of different coloring methods. The prototypical coloring method is *quasi-stable coloring* (reviewed in Sec. 3), which was introduced in [10] as a relaxation of *stable coloring*, a technique used in graph isomorphisms [6, 8]. Further, with the optimizations mentioned above, COLOR reduces the latency of these methods from exponential to linear in the query size.

In summary, we make the following contributions:

Contributions

- We present a framework for applying arbitrary colorings and choices of statistics to produce graph summaries for cardinality estimation (Sec. 3)
- We define a general formula for performing inference over a lifted graph and show its optimality for acyclic queries and stable colorings (Sec. 4).
- We extend this formula to handle cyclic queries and propose additional techniques to tackle their additional complexity (Sec. 5)
- We explore a variety of methods for coloring a data graph (Sec. 6).
- We introduce crucial optimizations for this framework that allow for efficient and accurate inference over a dynamic lifted graph (Sec. 7).
- We perform a thorough experimental analysis and show that graph coloring can increase the accuracy of traditional cardinality estimators by up to 10^{13}x, with median q-errors less than 10 across all datasets (Sec. 8).

2 PROBLEM SETTING

Labeled Graphs. The data model that we use for this work is labeled graphs which may have labels on both edges and vertices. This captures the most common graph datasets such as knowledge graphs, RDF graphs, and social graphs. Formally, we define labeled graphs as follows:

Definition 1. *A labeled graph, $G(V, E, \lambda, \chi)$, is a directed graph with vertices V, edges E, label set $\mathbb{L}$, and two additional functions,*

- $\lambda : \quad V \rightarrow 2^{\mathbb{L}}$ *which associates vertices to a set of vertex labels*[1]
- $\chi : \quad E \rightarrow 2^{\mathbb{L}}$ *which associates edges to a set of edge labels*

Note, this data model does not cover the broader class of property graphs which allow numeric or string properties, e.g. age = 73, to be associated with vertices or edges and included in queries. Standard techniques which bucketize these values and treat them as labels could be applied directly in our framework; more sophisticated approaches to handling property predicates are left for future work.

Subgraph Counting. Throughout this paper, we will assume that the vertices and edges of the query graph have a single label, which we denote $\lambda_Q(x)$ and $\chi_Q(x, y)$ respectively. The goal of subgraph counting is to find the number of occurrences of a query graph Q in a larger data graph G. However, there are different definitions of an "occurrence" which result in slightly different problems; isomorphism counting, induced subgraph counting, and homomorphism counting. In this work, as in [15, 17, 18, 20], we focus on homomorphism counting where 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.

Definition 2. *Given an unlabeled graph Q and a data graph G, we define the set of subgraph matches as,*

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

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

Each match, π, is a function from V_Q to V_G. When Q and G are labeled graphs, we add two natural conditions,

$$\lambda_Q(v) \subseteq \lambda_G(\pi(v)) \,\forall\, v \in V_Q \qquad \chi_Q(e) \subseteq \chi_G(\pi(e)) \,\forall\, e \in E_Q \quad (1)$$

Therefore, the subgraph count is $|\hom(Q, G)|$.

In keeping with prior work [18], we adopt the semantics of homomorphism counting. But, we note that counting isomorphisms or induced subgraphs can be reduced to the homomorphism counting problem using inclusion/exclusion formulas [4], and we leave it to future work to explore more direct adaptations to other semantics.

This work studies the problem of *cardinality estimation* which can be viewed as approximate subgraph counting done through the lens of statistics gathered on the data graph. 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, $\mathbf{s} \stackrel{def}{=} \mathcal{E}_{pre}(G)$ and 2) estimating the cardinality during the online phase, $c \stackrel{def}{=} \mathcal{E}_{on}(Q, \mathbf{s}) \in \mathbb{R}_+$. The goal is to produce an estimate where $c \approx |\hom(Q, G)|$.*

The primary metrics for these algorithms are: 1) the accuracy of $c \approx |\hom(Q, G)|$ 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 [9, 14]. 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 both labels and edge directionality, and introduce them again 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, 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:*

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

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 [6, 8]. However, they are unsuitable for our purpose, because stable colorings of real-world graphs require a very large number of colors [10]. In fact, in a random graph, with high probability every vertex has a distinct color [7]. It has been shown [10] that, by relaxing $=$ to $\approx$ in (2), a quasi-stable coloring of a graph (V, E) has significantly fewer colors than vertices, $|C| \ll |V|$. In this paper we do not need to define formally the approximation $\approx$ in (2), but instead rely on the coloring algorithm in [10] and refer the reader to the discussion there.

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

Definition 6. *Fix a directed, unlabeled 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 $E_F = \{(c_1, c_2) \mid E_{c_1 c_2} \neq \emptyset\}$.*
- *$\psi : C \to \mathbb{N}$ where $\psi(c) = |V_c|$*
- *$\tau : E_{c_1 c_2} \to \mathbb{R}_+$ maps edges of the labeled graph to statistics about the edges in $E_{c_1 c_2}$*

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

$$\tau_{\min}(c_1, c_2) = \min\{|\{(v_1, v_2) \mid v_2 \in V_{c_2}\}| \mid v_1 \in V_{c_1}\}$$

$$\tau_{\text{avg}}(c_1, c_2) = \frac{|E_{c_1 c_2}|}{|V_{c_1}|}$$

$$\tau_{\max}(c_1, c_2) = \max\{|\{(v_1, v_2) \mid v_2 \in V_{c_2}\}| \mid v_1 \in V_{c_1}\}$$

They represent the minimum degree, the average degree, and the maximum degree of a vertex in c_1 to vertices in c_2 respectively. Unless otherwise stated, we will assume that τ means τ_{avg}.

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

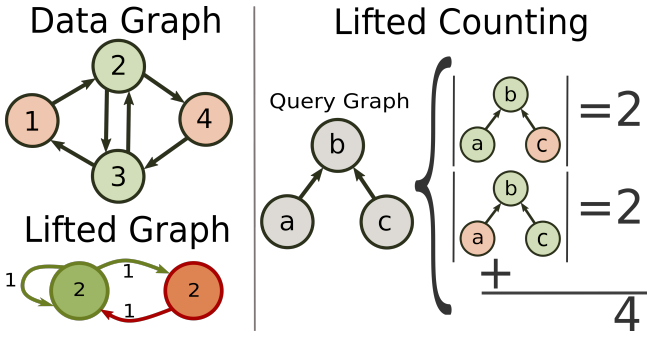

Figure 1: Lifted Counting Example

compression of the data graph, and is computed offline, during preprocessing. It represents the statistics $\mathbf{s} = \mathcal{E}_{pre}$ in Def. 3.

Example 2. *For clarity, we provide an example (unlabeled) data graph and lifted graph in Figure 1. Topologically, there are two different "kinds" of vertices in the data graph. One set of vertices has out/indegree of 1 and the other has out/indegree of 2. Due to the arrangement of these edges, we can produce a good coloring where vertices* 2 *and* 3 *are green and vertices* 1 *and* 4 *are red. Further, we can produce the lifted graph shown below by choosing to keep the average degree as our edge statistic. In this figure, the* 2*'s which the colors are tagged with represent their cardinalities, i.e. the values of* ψ*. The* 1*'s which the edges are tagged with represent that each green vertex has (typically) 1 edge to red vertex(s) and 1 edge to green vertex(s) and similarly for red vertices, i.e. the values of* τ*. Note that this lifted graph precisely captures the distribution of edges in the data graph while being half the size.*

Labeled Lifted Graphs. We explain now how to account for labels, and for edge directionality in the lifted graph. Recall that the set of labels is $\mathbb{L}$, and for simplicity assume that vertex labels are disjoint from edge labels. For each edge label $b \in \mathbb{L}$ we create a copy, b^-. For each edge $(v, v') \in E$ we create a reversed edge (v', v), and for every label $b \in \chi(v, v')$ we add the label b^- to $\chi(v', v)$. Given a data graph $G = (V, E)$, we modify the quasi-stable Definition 5 as follows: for every two vertices v_1, v_2 and label a, if $c(v_1) = c(v_2)$ and a is a label of both v_1 and v_2, then we require that for every color c and edge label b, the number of b-labeled edges from the two vertices into c must be approximately equal. For colors c, c_1, c_2, vertex label a and edge label b, we denote by $V_c = \{v \in V \mid \sigma(v) = c\}$, $V_{c,a} = \{v \in V_c \mid a \in \lambda(v)\}$ and $E_{c_1,c_2,a,b} = \{(v_1, v_2) \mid v_1 \in V_{c_1,a}, v_2 \in V_{c_2}\}$. We modify the definition of ψ, τ to account for labels, as follows:

$$\psi(c, a) = |V_{c,a}| \qquad \tau(c_1, c_2, a, b) = \frac{|E_{c_1,c_2,a,b}|}{|V_{c_1,a}|}$$

Here, we assumed that τ is τ_{avg}; the definitions of $\tau_{\min}$ and $\tau_{\max}$ are adapted similarly and omitted. While these updates increases the size of the lifted graph, this is ameliorated by storing it as a sparse graph: when $E_{c_1,c_2,a,b} = \emptyset$ then we do not store the value $\tau(c_1, c_2, a, b)$, and assume implicitly that it is 0.

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*

$$W : \hom(Q, F) \to \mathbb{R}_+$$

The lifted subgraph count is,

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

A homomorphism $\pi : Q \to F$ 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 $W(\pi)$ estimates the number of outputs of the query labeled according to π. 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:*

$$W(\pi) \stackrel{def}{=} \psi(\pi(x_1), \lambda_Q(x_1)) \prod_{(x_i,x_j) \in E_Q} \tau(\pi(x_i), \pi(x_j), \lambda_Q(x_i), \chi_Q(x_i, x_j)) \tag{4}$$

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 the label $\chi_Q(x_j, x_i)$ is a "reverse label", b^{-1}. 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. *Our first example shows that the lifted graph estimator generalizes the traditional estimator in Def. 4. We illustrate with an unlabeled 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 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:*

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

In essence, this theorem states that a stable coloring is a perfect statistic for cardinality estimation of acyclic query graphs. However, we cannot use it in practice, because the number of stable colors of real graphs is usually very large, close to the number of vertices in the data graph [10, 16]. However, the theorem demonstrates that, 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 contain 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 labels, 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 a direct edge from the starting vertex to the ending vertex. Further, instead of considering all edge and vertex labels that might define a path, we group paths by their sequence of directions, $D = \{\leftarrow, \rightarrow\}^k$, 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 $C_{c_1,c_2,D}(G)$ be the subset of paths in $\mathcal{P}_{c_1,c_2,D}(G)$ which are closed. The path closure probability is then,*

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

Given this statistic, we can define an expression for the cycle closure probability, $P(E_i | E_1, \ldots, E_{i-1})$. Let $\mathcal{S}_{i-1}$ be the set of simple paths in $E_1, \ldots, E_{i-1}$ 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,

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

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, $e_1, \ldots, e_{|E_Q|}$, 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 $e_1, \ldots, e_{i-1}$. The modified definition of the estimator (4) is:

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

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

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. 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 [10]. 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. Note, this coloring method does not take edge or vertex labels into account.

Neighbor Label Coloring. An alternative to quasi-stable coloring, this method instead 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.

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.

Vertex Label Coloring. This coloring approach aims to make the distribution of vertex labels within colors more uniform. It first selects the color to split by finding the largest color with the most even distribution of a particular label. The nodes within that color which possess that label are then put in the new color and the ones which do not will remain in their color.

Hash Coloring. Lastly, we consider the naive hash coloring which uniformly randomly sorts vertices into colors. This corresponds to the partitioning used by [2] 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 does not take the topology of the graph into account and, therefore, offers limited improvement to the accuracy of the estimator. To match our framework, we select colors to split by iterating through them sequentially, and we split a color by randomly dividing its vertices.

Mixture Coloring. The previous colorings generally target a particular source of variance related to either topology or label 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. 7), we show that this is the most accurate coloring across a range of workloads.

Algorithm 1 Optimized Inference Algorithm

Require: $F_{G,\sigma,l}(G_F(C, E_V), \psi, \tau)$, $Q(V_Q, E_Q)$, $v_1, \ldots, v_{|V_Q|}$
// Lifted Graph, Query Graph, Vertex Order
$PC = \{(\{\}, 1)\}$ // Partial Colorings
$V_F = \{\}$
for $i \in [1, \ldots, |Q|]$ **do**
  $PC' = \{\}$
  $E_i = \{e \in E_Q \mid v_i \in e\}$
  for $\pi, w \in PC$ **do**
    for $c \in C$ **do**
      $\pi' = \pi \cup (v_i \to c)$
      $w' = w \cdot \prod_{e \in E_i} \omega(\pi', e) \leftarrow$ Def. **??**
      $PC' = PC' \cup \{\pi', w'\}$
    end for
  end for
  $PC = PC'$
  $V_S = \{v \in V | (v, v_j), (v_j, v) \notin E_Q \ \forall\ j > i\} \setminus V_F$
  $V_F = V_F \bigcup V_S$
  $PC = \sum_{V_S} PC \leftarrow$ Partial Aggregation Sec. 7.1
  $PC = Sample(PC) \leftarrow$ Sampling Sec. 7.2
end for
return PC

7 OPTIMIZATION

7.1 Partial Aggregation

Naively, the runtime of inference on the lifted graph would be exponential in the size of the query graph, making it intractable for even moderately sized query graphs. This is because the lifted graph is complete, or nearly complete, so the size of $Hom(Q, F)$ is approximately $|C|^{|Q|}$. Fortunately, when the estimator is decomposable, we can rephrase the sum in Eq. 3 to drastically reduce this runtime via aggregate push-down. We can see this rephrasing as follows,

$$\Phi(Q, F, W_{\psi,\tau}) = \sum_{c_1, \ldots, c_{|Q|} \in C} \psi(c_0) \prod_{i=1}^{|E_Q|} \omega(\pi_{v_k \to c_k}, e_i) \tag{6}$$

At this point, we can identify this problem as a *Functional Aggregate Query* as described in [11], and we can apply the techniques there

to solve it efficiently.[2] As an example, suppose that the query graph Q is a line graph $v_1 \rightarrow v_2 \rightarrow v_3$. The naive expression would be the following $O(|C|^3)$ expression,

$$\Phi(Q, F, W_{\psi,\tau}) = \sum_{c_1,c_2,c_3 \in C} \psi(c_0)\tau((c_1, c_2))\tau((c_2, c_3))$$

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 of this strategy depends on the variable order, and, in particular, it 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 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 [19]. Further, in order to accommodate the sampling techniques that we discuss next, we restrict these tree decompositions to path decompositions and get runtime results relative to the pathwidth.

7.2 Sampling Techniques

In real world systems, cardinality estimation needs to be an extremely fast and consistent process because it is an overhead incurred by every query. Specifically, a super-linear runtime with respect to the size of the query is generally viewed as problematic. While partial aggregation significantly speeds up inference for simple query graphs with low treewidth, larger and denser query graphs may still pose a problem under these requirements. To avoid this, we propose a sampling procedure which we integrate into the inference process along with the partial aggregation.

At a high level, this sampling procedure is similar to a weighted version of the WanderJoin algorithm from [15]. We apply a Thompson-Horowitz estimator to randomly chosen 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 of our algorithm, we process a single vertex of the query graph and materialize an intermediate result consisting of partial colorings and weights associated with them. After this 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. Further, because we can still apply aggregation where possible, we reduce the amount of sampling required to maintain a small number of partial colorings.

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 6 is simply 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 inherently challenging, we instead approximate this contribution via its associated weight. This reflects an assumption that partial colorings with high weight are likely to disproportionately contribute to the final sum. To keep our estimator unbiased, we apply Thompson-Horowitz estimation and simply multiply the weight of each sampled partial coloring by the inverse of its selection probability. Finally, we scale the total weight of the sampled colorings to make it equal to the weight of the partial colorings 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. Traditionally, summary based estimators simply recalculate the summary on a regular basis to accommodate updates [9]. This approach leads to severe decreases in accuracy under even modest updates because the estimator is "blind" to them. On the other hand, recalculating the summary before each query is far too costly for most systems. 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 a corresponding label set L:*

- $\theta_V = (v, L)$
- $\theta_E = (v_1, v_2, L)$ *where* $v_1 \in G_V, v_2 \in G_V$,

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,*

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

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

Table 1: Estimator Failure Rates

Dataset\Method	cs	wj	jsub	impr	cset	alley	alleyTPI	BSK++	sumrdf	COLOR (AvgMix32)
human	0.67	0.00	0.22	0.63	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	0.02	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
yeast	1.00	0.97	0.97	0.11	0.00	0.68	T/O	0.63	0.88	0.00
dblp	1.00	0.99	0.94	0.15	0.00	0.77	T/O	0.70	0.85	0.00
youtube	0.99	0.93	0.99	0.22	0.00	0.15	T/O	0.63	0.78	0.00
eu2005	0.95	0.90	0.91	0.55	0.00	0.00	T/O	0.22	0.44	0.00
patents	0.98	0.88	0.98	0.08	0.00	0.15	T/O	0.67	0.79	0.00

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 do not have any 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 $\theta_E = (v_1, v_2, L)$, the edge update function δ_E increases each entry in τ that corresponds to the combination of labels and colors in the edge update. To retrieve the colors, c_1 and c_2, associated with v_1 and v_2, we need to look up their values in π which we store compactly (and approximately) as a series of cuckoo filters. However, the labels of $\chi(v_1)$ and $\chi(v_2)$ remain unknown. Instead of just updating the average degree for an arbitrary vertex label, we distribute the added degree across each possible vertex label for v_2, scaling according to the proportion of vertices in c_2 which possess that label.

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, $\mathbb{S}_{\theta E}$:

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

8 EVALUATION

In this section, we provide a detailed experimental analysis of our framework.

Datasets & Workload. We consider datasets from [18] and [21] for our analysis. These datasets come from a variety of domains. Broadly, those from [21] are larger and undirected while those from [18] are smaller and directed. We adapt undirected workloads to directed methods by including reverse edges in the data graph but not in the query graphs. Table 2 shows their different characteristics.

Table 2: Experimental Datasets

Dataset	|V|	|E|	$\|\ell_V\|$	$\|\ell_E\|$
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

Comparison Methods. For comparison, we use the methods considered in [18] and additionally apply them to the larger, more complex datasets from [21]. These methods include: 1) Correlated Sampling (CS) [22] 2) Characteristic Sets (CSet)[17] 3) Wander Join (WJ) [15] 4) Alley (alley) and alleyTPI [12] 5) Join Sampling with Upper Bounds (JSUB), an adaptation of [27] 6) Bound Sketch (BSK) [2] which corresponds to a hash-based coloring and using the max degree estimator (when we apply our partial aggregation optimization, we call this BSK++) 7) IMPR [3] and 8) SumRDF [20] 9) 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 [18] and [12] (i.e. .03 for all methods except for Alley which uses .001). We ran into challenges when attempting to run AlleyTPI on the new datasets. It timed out (> 30 minutes) when building its indices for youtube, dblp, eu2005, and patents, and it produced an error when estimating yeast. We are reaching out to the authors of [12] to attempt to remedy this.

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.

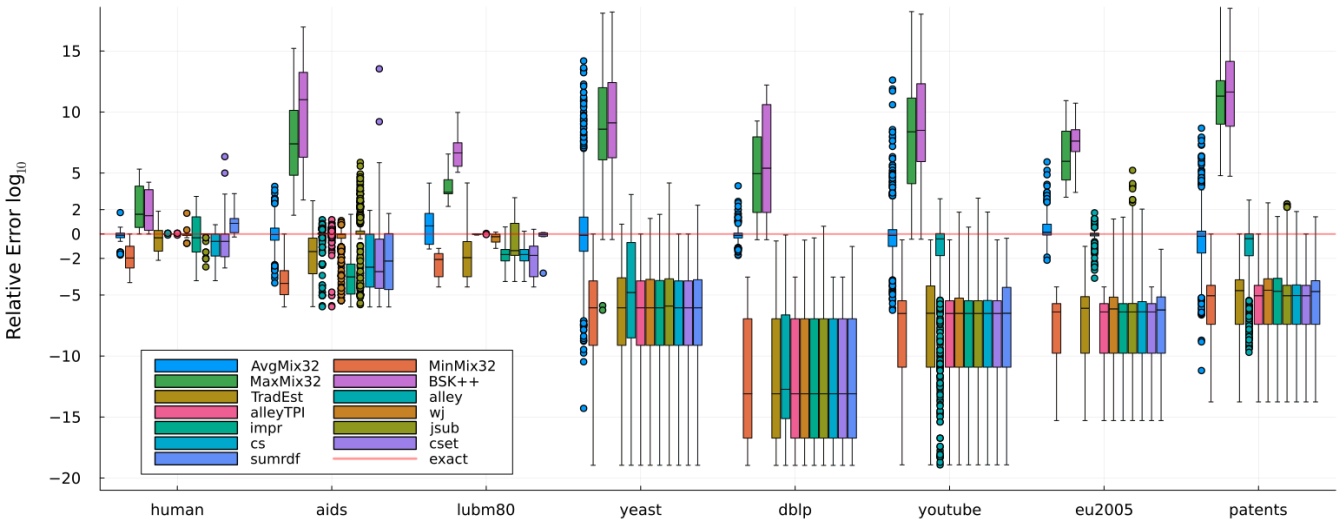

Figure 2: Relative Error by Estimator

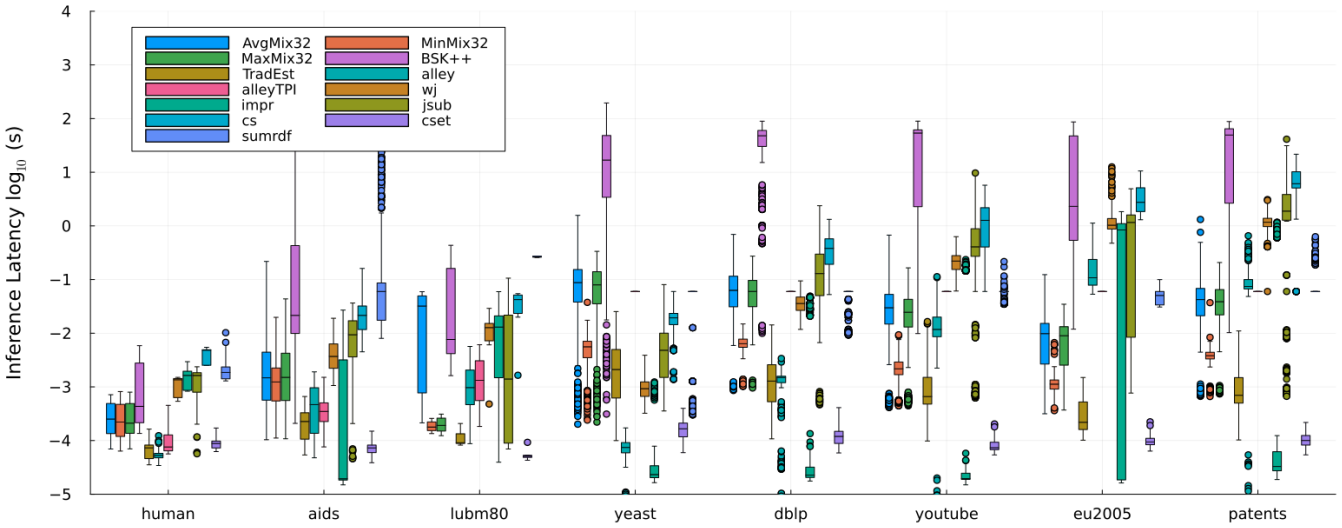

Figure 3: Inference Time by Estimator

Experimental Setup. To reduce noise in our latency results, 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 a Intel(R) Xeon(R) CPU E7-4890 v2 @ 2.80GHz CPU, and all summary building and inference is done using a single thread. The reference implementation is available anonymously at: https://anonymous.4open.science/r/Cardinality-with-Colors-4333

8.1 Estimator Failure

There are two main ways that an estimator can fail to provide meaningful results: 1) it can time out which we define as taking longer than 1 minute to report a result 2) a sampling-based method can fail to find any qualifying samples. In Table 1, we show the proportion of queries that result in estimation failure for each dataset and technique. The simpler sampling-based methods (CS, WJ, JSUB, IMPR) face estimation failure even on the smaller, less dense query workloads (human, aids, lubm80) and fail to find samples for nearly any queries for the larger more complex workloads (yeast, dblp, youtube, eu2005, patents). Alley achieves much higher success rates across

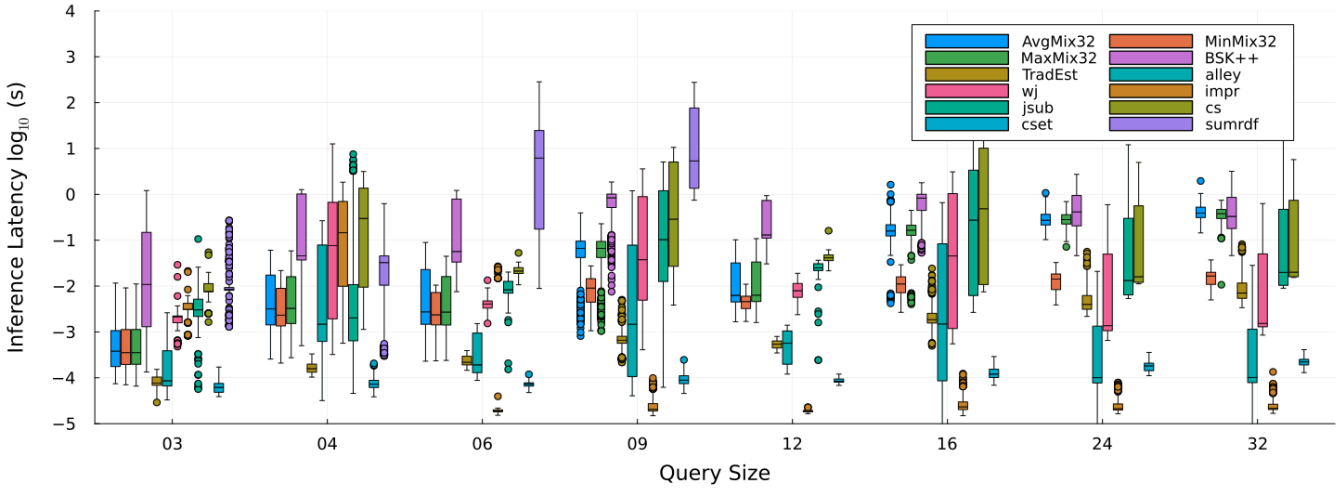

Figure 4: Inference Time by Query Size

datasets due to its sophisticated sampling approach, but it still fails a significant portion of the time on four datasets. The 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.

8.2 Accuracy

In Fig. 2, we show the relative error of various methods and workloads[3]. Across workloads, our method, AvgMix32, is unbiased and scales well to larger, more cyclic query queries (yeast, dblp, youtube, eu2005, patents). It even achieves a median Q-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 [18] fail to scale to the larger more cyclic workloads. In particular, we reproduce the finding in [12] that WanderJoin, IMPR, and JSUB overwhelmingly fail to find a positive sample in a reasonable time on these datasets. Further, SumRDF times out on all larger queries due to its lack of aggregation and sampling.

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. 7, 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 [10] 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.

[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 assumed to be an estimate of 1.

8.3 Inference Latency

Fig. 3 shows the distribution of inference latencies for each method across workloads. We can see that the inference latency of the COLOR methods lies in the middle of the competing methods across workloads. On the smaller, less cyclic queries of human, they achieve a very fast median latency of around 10^{-4} seconds due to partial aggregation, and on the larger more complex queries of patents they have a median latency of $\sim$ .05 seconds via sampling. We can see this more clearly in Fig. 4 which shows the latency increasing linearly in the query size.

Further, when compared to the other graph summarization methods, SumRDF and BSK++, the methods from our framework scale far better to larger queries. The former methods timeout (>1 minute) 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 achieving 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. Fortunately, even a compact summary ($<$ 20MB) can be highly accurate as shown by Fig. 5. Part of this compactness comes from the fact that we store our summary sparsely. This means that if two colors do not share an edge with a particular label combination then we don't explicitly store any degree statistic about this combination. Due to this, a better coloring can actually result in a more compact summary because

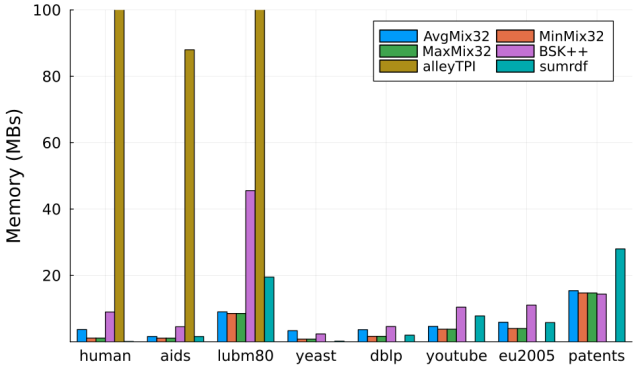

Figure 5: Statistics Size

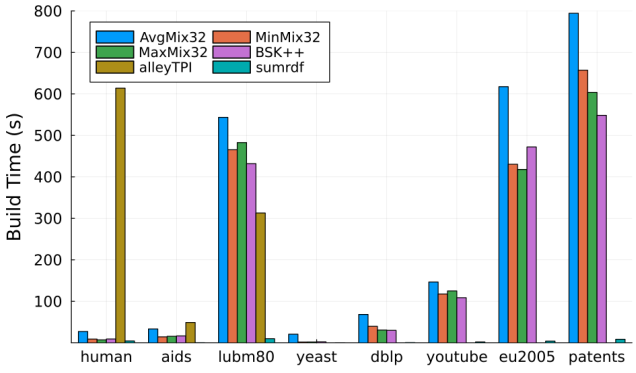

Figure 6: Build Time

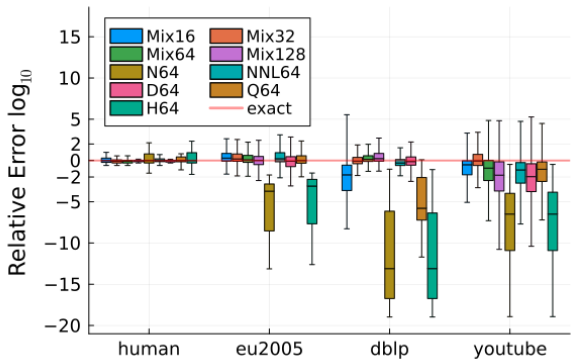

Figure 7: Relative Error by Coloring Method

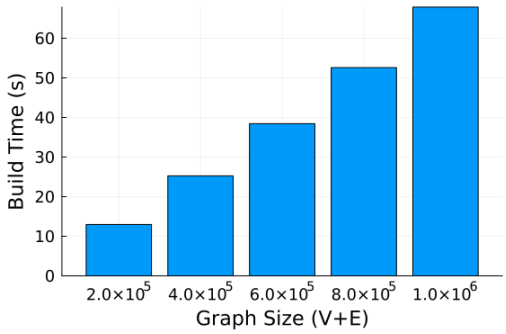

Figure 8: Construction Scaling

many color and label combinations won't occur. On the other hand, a hash coloring can result in a large summary, as shown by BSK on lubm80, because every pair of colors is likely to have at least a single edge between them that must be stored explicitly. On Human and Lubm80, AlleyTPI's pattern index requires 600 and 300 MB, respectively.

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. 8, we confirm this by generating erdos-renyi graphs of varying sizes and recording the average time to build the lifted graph over 20 trials.

8.5 Updates

In Fig. 9, we evaluate the effectiveness of our method for handling updates (Sec. 7.3). To do this, we randomly partition the edges of the AIDS 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. Additionally, due to uncertainty about the labels and colors for added edges and vertices, the proposed update method generalizes and may update statistics where it may be unnecessary, leading to errors at high update proportions.

Surprisingly, we find that, as more of the lifted graph is updated, the accuracy remains very consistent. This only fails when over half of the graph is composed of updates, at which point a reconstruction of the lifted graph is necessary. This implies that a full rebuilding of the lifted graph can occur very infrequently and be amortized over many updates.

Update Time. There was a relatively consistent throughput for edge and vertex updates across configurations. For vertex updates, the throughput was roughly 2,500 OPS while for edge updates the throughput was roughly 12,500 OPS. Vertex updates are slower because in addition to the edge statistics, the largest color must be updated with the new vertices.

8.6 Micro-Benchmarks

Inference Sampling. In Fig. 10, we vary the sample size used during inference on the Youtube workload to demonstrate the efficacy of our sampling method. Using a very small sample size 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 with the

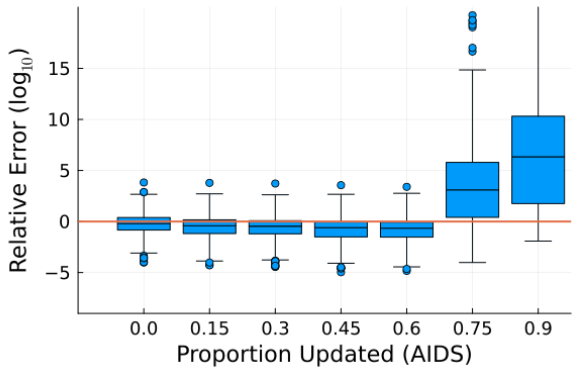

Figure 9: Relative Error vs Proportion Updated (AIDS)

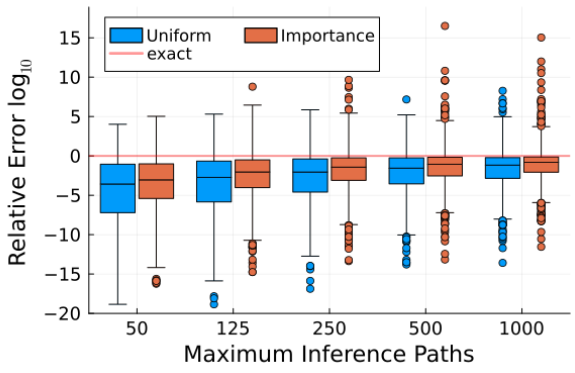

Figure 10: Relative Error vs Samples (Youtube)

performance at 250 samples for importance sampling being roughly equal to the accuracy of uniform sampling at 1000 samples.

Cycle 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. 11. 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. 12 demonstrates the effects of partial aggregation and sampling. In this figure, using the Youtube data graph, we observe how inference latency changes across different query pathwidths if partial aggregation or sampling is used. Recall that pathwidth is a measure of cyclicity where a query with pathwidth 1 is acyclic and higher pathwidth queries are increasingly interconnected. Without partial aggregation, the estimation begins to time out (>1 min) when query pathwidths exceed 2. This is generally due to the higher pathwidth queries being larger as well. When using partial aggregation without sampling, the estimation times out after query 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 the importance of including partial aggregation for achieving speedups without affecting accuracy, and that the use of sampling can achieve consistent, fast inference.

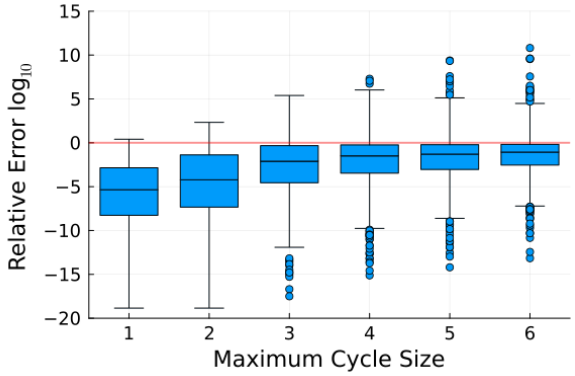

Figure 11: Relative Error vs Max Cycle Probabilities Stored (Youtube)

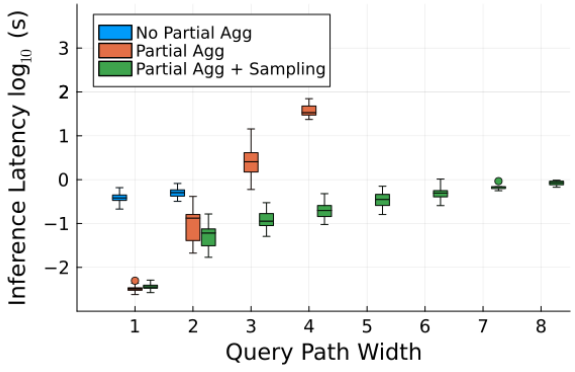

Figure 12: Build Time vs Query Path Width (Youtube)

REFERENCES

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

[2] 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

[3] 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

[4] Radu Curticapean, Holger Dell, and Dániel Marx. 2017. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 210–223.

[5] 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.

[6] 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

[7] 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

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

[9] 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

[10] 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

[11] 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

[12] 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

[13] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf

[14] 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.

[15] 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

[16] 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

[17] 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

[18] 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

[19] 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

[20] 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

[21] 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.

[22] 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

[23] 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).

[24] Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2023. FactorJoin: A New Cardinality Estimation Framework for Join Queries. *Proc. ACM Manag. Data* 1, 1 (2023), 41:1–41:27. https://doi.org/10.1145/3588721

[25] Kangfei Zhao, Zongyan He, Jeffrey Xu Yu, and Yu Rong. 2023. Learning with Small Data: Subgraph Counting Queries. *Data Sci. Eng.* 8, 3 (2023), 292–305. https://doi.org/10.1007/S41019-023-00223-W

[26] 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

[27] 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

A PROOF OF THM. 1

In this section, we prove the following theorem for unlabeled graphs, but the extension to labeled graphs is straightforward.

Theorem 3. *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:*

$$|\hom(Q, G)| = \Phi(Q, \mathcal{G}) \tag{7}$$

Proof. We start by noting that each match in the data graph, $\pi \in Hom(Q, G)$, is associated with precisely one coloring, $\pi' \in Hom(Q, F)$, based on the matched vertices' colors, i.e. $\pi' = \sigma_S \circ \pi$. We denote the set of matches with this coloring as $Hom(Q, G|\pi')$. If we calculate $|Hom(Q, G|\pi')|$ for each coloring, π', then we can compute the total as $\sum_{\pi' \in Hom(Q,F)} |Hom(Q, G|\pi')|$. By this logic and equation 4, we simply need to show that $W(\pi') = |Hom(Q, G|\pi')|$.

We note that Q is a acyclic and proceed inductively on the topological ordering $v_1, \ldots, v_{|Q|}$. Let Q_i be the sub-tree restricted to the vertices $v_1, \ldots, v_i$, and let π'_i be the coloring restricted to these vertices. We begin with the base case,

$$W(\pi'_1) = \psi(\pi'_1(v_1)) = |Hom(Q_1, G|\pi'_1)|$$

The number of matches to a vertex of a color is equal to the number of vertices in that color, so this is immediately true. Now, suppose that $W(\pi'_i) = |Hom(Q_i, G|\pi'_i)|$, and we will show that $W(\pi'_{i+1}) = |Hom(Q_{i+1}, G|\pi'_{i+1})|$. We restate Eq. 4 as follows,

$$W(\pi'_{i+1}) = \psi(\pi'_{i+1}(v_1)) \prod_{(v_j, v_k) \in E_{Q_{i+1}}} \tau((\pi'_{i+1}(v_j), \pi'_{i+1}(v_k)))$$

Let v_j be the parent of v_{i+1}, and we can express $W_{\psi,\tau}(\pi'_{i+1})$ inductively,

$$W(\pi'_{i+1}) = W(\pi'_i) \cdot \tau((\pi_{i+1}(v_j), \pi_{i+1}(v_{i+1})))$$

By our inductive assumption, this means,

$$W(\pi'_{i+1}) = |Hom(Q_i, G|\pi'_i)| \cdot \tau((\pi_{i+1}(v_j), \pi_{i+1}(v_{i+1})))$$

Denote the number of edges that a vertex $v \in V_G$ has to vertices with color C as $deg_G(v|C)$. Because σ_S is a stable coloring, we know,

$$deg_G(v|C) = deg_G(v'|C) \;\; \forall v, v' \in V_G \; s.t. \; \sigma_S(v) = \sigma_S(v')$$

By the definition of τ, this degree is precisely,

$$deg_G(v|C) = \tau(\sigma_S(v_j), C)$$

We denote this degree with its color as $deg_G(C|C')$. Returning to our expression for $W_{\psi,\tau}(\pi'_{i+1})$, we can now plug in our degree expression,

$$W(\pi'_{i+1}) = |Hom(Q_i, G|\pi'_i)| \cdot deg_G(\pi'_{i+1}(v_j)|\pi'_{i+1}(v_{i+1}))$$

By simply expanding $|Hom(Q_i, G|\pi'_i)|$ into a sum, we get,

$$W^{std}(\pi'_{i+1}) = \sum_{\pi_i \in Hom(Q_i, G|\pi'_i)} deg_G(\pi'_{i+1}(v_j)|\pi'_{i+1}(v_{i+1}))$$

At this point, we use the fact that $deg(\pi'_i(v_j)|\pi'_{i+1}(v_{i+1}))$ $= deg(\pi_i(v_j)|\pi'_{i+1}(v_{i+1}))$ when $\sigma_S(\pi_i(v_j)) = \pi'_i(v_j)$, which is assured by $\pi \in Hom(Q_i, G|\pi'_i)$.

$$W^{std}(\pi'_{i+1}) = \sum_{\pi_i \in Hom(Q_i, G|\pi'_i)} deg_G(\pi_i(v_j)|\pi'_{i+1}(v_{i+1}))$$

Because Q is a tree, the RHS is the target of the induction,

$$W^{std}(\pi'_{i+1}) = |Hom(Q_{i+1}, G|\pi'_{i+1})|$$

□

B PROOF OF THEOREM 2

We begin by restating the theorem,

Theorem 4. *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 $O(|C|^{tw(Q)})$ where $tw(Q)$ is the treewidth of Q where $\Phi(Q, F, W_{\psi,\tau})$ is defined,*

$$\Phi(Q, F, W_{\psi,\tau}) = \sum_{c_{v_1}, ..., c_{v_{|Q|}} \in C} \psi(c_0) \prod_{i=1}^{|E_Q|} \omega(\pi_{v_k \to c_{v_k}}, e_i | e_1, ..., e_{i-1})$$

We now define tree decompositions,

Definition 12. *Given a graph H, a tree decomposition $T(E_T, V_T, \chi, \gamma)$ is composed of four pieces,*

1. *E_T, V_T are the edges and vertices of a tree*
2. *$\chi : V_T \to 2^{V_H}$ is a function which maps vertices in the tree to sets of vertices in H*
3. *$\gamma : V_T \to 2^{E_H}$ is a function which maps vertices in the tree to sets of edges in H*

Lastly, it has two requirements,

1. *For all $v \in V_H$, the set of vertices in V_T that contain v form a connected sub-tree*
2. *Each edge in E_H is mapped to exactly one vertex of V_T by γ and the vertex of V_T which it is mapped to includes both the endpoints*

The treewidth is then defined as follows,

Definition 13. *Given a graph H, let the set of valid tree decompositions be $\mathcal{T}_H$. The treewidth is then defined as,*

$$\min_{T \in \mathcal{T}_H} \max_{v \in V_T} |\chi(v)| - 1$$

Intuitively, graphs that are "more acyclic" will have a lower treewidth and graphs that are "more cyclic" have a higher treewidth.

Proof. For this problem, we can take advantage of tree decompositions by using them to structure the summation in (6). Suppose that the treewidth of Q is k and let $T(E_T, V_T, \chi)$ be a tree decomposition which matches this width. To avoid confusion, we will denote vertices of T as v'_i and vertices of Q as v_i. Further, let $v'_1, \cdots, v'_{|V_T|}$ be a topological ordering of T where v'_1 is the root of the tree, and let $v_1, \cdots, v_{|V_Q|}$ be an ordering of Q such that the sub-tree corresponding to v_i is not a sub-tree of $v_{>i}$. Let $Par(v')$ denote the parent of v' in T and let $Ch(v')$ be the set of child vertices of v' in T. Lastly, we denote the set of query vertices which associated with $v'_{|V_T|}$ and not its parent as $\mathbf{X}_i = \chi(v'_i) \setminus \chi(Par(v'_i))$. We denote the set of query vertices which are associated with $v'_{|V_T|}$ AND its parent as $\mathbf{Y}_{|V_T|} = \chi(v'_{|V_T|}) \cap \chi(Par(v'_{|V_T|}))$.

We will proceed iteratively on these vertices starting with $v'_{|V_T|}$ and proceeding backwards towards the root.

Base Case: Our base case is $v'_{|V_T|}$ which is necessarily a leaf of the true due to the topological ordering. We denote the output of the base case as the function $S_{|V_T|}(c_{y_1}, \ldots, c_{y_m})$ and define it as follows,

$$S_{|V_T|}(c_{y_1}, \ldots, c_{y_m}) = \sum_{c_{x_1}, \ldots, c_{x_n} \in C} \prod_{e_i \in \gamma(v'_{|V_T|})} \omega(\pi_{x_j \to c_{x_j}}, e_i | e_1, \ldots, e_{i-1})$$

At this point of the computation, we will fully materialize the values of the function $S_{|V_T|}$ which has a domain of size $|C|^{|Y_{|V_T|}|}$ and each point requires computing a summation of $|C|^{|X_{|V_T|}|}$ terms. Therefore, the entire computation requires $|C|^{|\chi(v'_{|V_T|})|} \leq |C|^k$ time.

Inductive Case: Suppose that all vertices v'_j where $j > i$ have already been processed, we now consider handling v'_i. The output of this step is defined as,

$$S_i(c_{Y_{v'_i,1}}, \ldots, c_{Y_{v'_i,m}}) = \sum_{c_{X_{v'_i,1}}, \ldots, c_{X_{v'_i,n}} \in C} \prod_{e_i \in \gamma(v'_i)} \omega(\pi_{X_{v'_i,j} \to c_{X_{v'_i,j}}}, e_i | e_1, \ldots, e_{i-1}) \cdot \prod_{v'_j \in Ch(v'_i)} S_{v'_j}(c_{Y_{v'_j,1}}, \ldots, c_{Y_{v'_j,m}})$$

As in the base case, the computation required to fully materialize the values of S_i is bounded by $|C|^{|\chi(v'_{|V_T|})|} \leq |C|^k$.

At this point, we just need to confirm that S_1 is equal to $\Phi(Q, F, W_{\psi,\tau})$ by showing that the sequence of computations represents a valid rearrangement of the sums of the original formula. To this end, we note that every query vertex appears in the summation of precisely one inductive step. This is due to the fact that each query vertex forms a sub-tree in the tree decomposition, so the root of that sub-tree is the only tree vertex which contains it and whose parent does not. Further, each instance of $\omega(\pi_{X_{v'_i,j} \to c_{X_{v'_i,j}}}, e_i | e_1, \ldots, e_{i-1})$ occurs precisely once because of the requirement that each edge of the query graph occurs in one vertex of the tree decomposition.

□