Subgraph Counting: Color Coding Beyond Trees

Venkatesan T. Chakaravarthy[1], Michael Kapralov[2], Prakash Murali[1], Fabrizio Petrini[3], Xinyu Que[3], Yogish Sabharwal[1], and Baruch Schieber[3]

[1,3]IBM Research
[1]{vechakra, prakmura, ysabharwal}@in.ibm.com
[3]{fpetrin, xque, sbar}@us.ibm.com
[2]EPFL
[2]michael.kapralov@epfl.ch

April 2, 2016

Abstract

The problem of counting occurrences of query graphs in a large data graph, known as subgraph counting, is fundamental to several domains such as genomics and social network analysis. Many important special cases (e.g. triangle counting) have received significant attention. Color coding is a very general and powerful algorithmic technique for subgraph counting. Color coding has been shown to be effective in several applications, but scalable implementations are only known for the special case of *tree queries* (i.e. queries of treewidth one).

In this paper we present the first efficient distributed implementation for color coding that goes beyond tree queries: our algorithm applies to any query graph of treewidth 2. Since tree queries can be solved in time linear in the size of the data graph, our contribution is the first step into the realm of colour coding for queries that require superlinear running time in the worst case. This superlinear complexity leads to significant load balancing problems on graphs with heavy tailed degree distributions. Our algorithm structures the computation to work around high degree nodes in the data graph, and achieves very good runtime and scalability on a diverse collection of data and query graph pairs as a result. We also provide theoretical analysis of our algorithmic techniques, showing asymptotic improvements in runtime on random graphs with power law degree distributions, a popular model for real world graphs.

1 Introduction

Graphs serve as common abstractions for real world data, making graph mining primitives a critical tool for analyzing real-world networks. Counting the number of occurrences of a query graph in a large data graph (subgraph counting, often referred to as motif counting) is an important problem with applications in a variety of domains such as bioinformatics, social sciences and spam detection (e.g. [8, 10, 23]).

Subgraph counting and its variants have received a lot of attention in the literature. Substantial progress has been achieved for the case of small queries such as triangles or

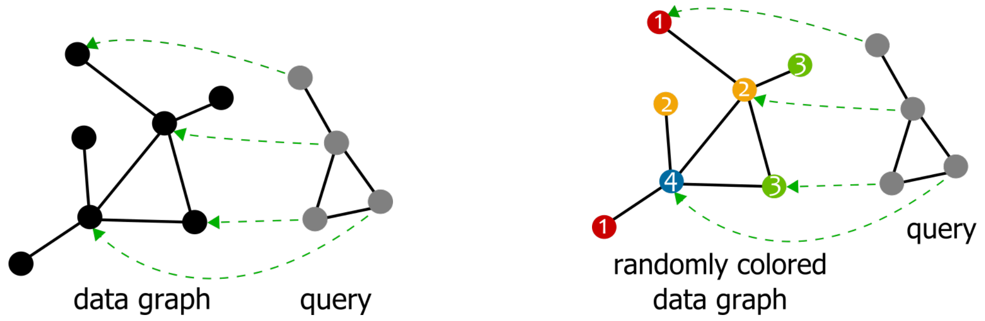


Figure 1: Illustration of a match (left) and a colorful match (right)

4-vertex subgraphs: not only have very efficient algorithms been developed (e.g. [15, 20, 27, 31]), but also theoretical explanation of their performance on popular graph models has been obtained (see [5] and references therein).

Some of the recent work has addressed larger queries [29, 30, 6, 26, 7], but our understanding here is far from complete. Even for reasonably large graphs (a million edges) and small queries (e.g. 5-cycles), the number of solutions tend to be enormous, running into billions. This explosion in the search space makes the subgraph counting problem very hard even for moderately large queries. Theoretically, the fastest known algorithm for counting occurrences of a k-vertex subgraph in an n-vertex data graph runs in time $n^{\omega k/3}$, where $O(n^{\omega})$ is the time complexity of matrix multiplication (currently $\omega \approx 2.38$). This improves upon the trivial algorithm with runtime n^k, but is prohibitively expensive even for moderate size queries.

To address the above issue, Alon et al. [2] proposed the *color coding* technique. Here, given a k-node query, we assign random colors between 1 and k to the vertices of the data graph, and count the number of occurrences of the query that are *colorful*, meaning the vertices matched to the query have distinct colors. See Figure 1. The count is scaled up appropriately to get an estimate on the actual number of occurrences. The accuracy is then improved by repeating the process over multiple random colorings and taking the average. Restricting the search to colorful matches leads to pruning of the search space and improved efficiency. Using this method, Alon et al. obtained faster algorithms for cetain queries such as paths, cycles, trees and bounded treewidth graphs.

The power of color coding as a very general counting technique together with the importance of subgraph counting in various applications (as mentioned above) makes it important to design practically efficient and scalable implementations. In a different work, Alon et al. [1] applied the color coding technique for counting the occurrences of *treelets* (tree queries) in biological networks. Color coding allowed them to handle tree queries up to size 10 in protein interaction networks, extending beyond the reach of previously known approaches [25, 18, 17]. Recently, Slota and Madduri [28, 30] presented FASCIA, an efficient and scalable distributed implementation of subgraph counting (via color coding), again for the case of treelet queries. However, despite considerable interest in non-tree queries from several application domains (see the experimental section for details), the technique has not been explored for more general settings. In this work we present the first efficient distributed implementation of *color coding beyond tree queries*.

As part of their original color coding solution, Alon et al. [2] presented faster algorithms for certain special classes of queries. They showed that if the query is a tree, then colorful subgraph counting can be solved in time $O(2^k m)$, i.e. in time *linear* in the size of the data graph. They extended the algorithm to show that if the query is close to a tree, specifically has *(small) treewidth* t, a running time of $O(2^k n^{t+1})$ can be achieved. Treewidth [9] is a widely adopted measure of the intrinsic complexity of a graph. Intuitively, it measures how close the topology of a given graph is to being a tree: tree queries have treewidth 1, and a cycle is the simplest example of a treewidth 2 query. The above algorithm, restricted to trees, forms the basis for the previously-mentioned treelet counting implementations [28, 30, 1].

While the runtime of the above algorithm is *linear* for the case of trees (i.e. *acyclic queries*), it becomes *at least quadratic* for query graphs of treewidth 2 and beyond. This phenomenon also manifests itself in practice: on real world graphs with even moderately skewed degree distribution load imbalance is observed and the running time tends to have quadratic dependence on the maximum degree of the graph. Thus, even triangles (the smallest cyclic query) are harder to handle, and have received considerable attention from the research community (as mentioned earlier).

The goal of this paper is to study the colorful subgraph counting problem on *queries of treewidth* 2, taking the first step in the realm of color coding with cyclic queries. The class of queries of treewidth 2 is quite rich. In particular, it contains all trees, cycles, series-parallel graphs and beyond. Figure 8 shows treewidth 2 queries (used in our experimental evaluation) drawn from real-world studies on biological, social and collaboration networks [22, 32, 4].

To the best of our knowledge, the previously-mentioned algorithm [1] is the best known algorithm for treewidth 2 queries, and we use it as our baseline. We rephrase this algorithm within our framework and devise a distributed implementation. The rephrased algorithm becomes a recursive procedure that decomposes the query into simpler *path subqueries*, which are then solved to get the overall count. We thus refer to our baseline as the *Path Splitting algorithm (*PS*)*.

Our Contributions

1. Building on the PS algorithm, we develop novel strategies that lead to significant performance gains in terms of runtime, scalability, and the size of graphs and queries handled.

2. Our algorithm works by decomposing the query to cycles and leaves, thereby reducing the problem of colorful subgraph counting on treewidth 2 queries to counting (annotated) cycles.

3. The decomposition in terms of cycles enables us to exploit the so-called degree ordering approach (e.g., MINBUCKET algorithm for triangle enumeration [5]) Specifically, we show how to force the computation process to (mostly) work around high degree vertices, leading to substantial speedups and scalability gains.

4. We present a detailed experimental evaluation of the algorithms on real-world graphs having more than million edges and real-world queries of size up to 10 nodes. The results show that our strategies offer improvements of up to 28x in terms of running time and exhibit improved scalability.

5. Finally, we complement our experimental evalutation by a theoretical analysis of the runtime of our degree ordering approach for cycle queries, on a popular class of random power law graphs (Chung-Lu graphs [14]). Our analysis provides justification for empirically observed performance gains of the approach.

Related Work

Subgraph counting has received significant attention in the fields of computational biology [25, 18, 17] and social network analysis [21, 13, 27, 5, 20]. We give an overview of prior work on the problem (both theoretical and empirical) as well as techniques for making subgraph counting scalable, and explain how our contributions relate to this prior work.

Color Coding and Approximate Subgraph Counting: Color coding was introduced in an influential paper by Alon et al. [2] as a fast algorithm for finding occurrences of a query in a data graph and counting the number of such occurrences. In a different work, Alon et al. [1] explored its applications to approximate subgraph counting (most commonly known as *motif counting*) in computational biology. They were motivated by the fact that subgraph counting is an important primitive for characterizing biological networks [23]. Color coding allowed Alon et al. to count occurrences of treelets (tree queries) up to size 10 in protein interaction networks, extending beyond the reach of previously known approaches [25, 18, 17]. A scalable distributed implementation of color coding for trees has been reported by Slota and Madduri [29, 30], but no principled solutions beyond tree queries are known. ParSE [33] extends beyond tree queries, by considering query graphs that can partitioned into subtemplates via edge cuts of size 1. However, the only class of query graphs that can be perfectly partitioned using this method is trees; ParSE resorts to brute force enumeration for other cases. Our work provides the first principled approach to implementing color coding in a scalable way beyond trees queries. Further, our analysis of the runtime of our cycle counting subroutine on a random graphs with a power law degree distribution provides a theoretical justification of our algorithmic techniques.

While our work and the above-mentioned prior work [1, 29, 30] count *non-induced* sugraphs, some other prior work [25, 18, 17] addressed the case of counting *induced* subgraphs. The search space of non-induced subgraphs is larger and furthermore, these counts are more robust with respect to perturbations of the data graph [1].

Degree Based Approaches: Designing scalable subgraph counting algorithms turns out to be hard even for the simple case of triangle counting. A naive approach lets each vertex enumerate pairs of neighbors and check if they are connected. This leads to wasteful computations and also runs into load balancing issues on graphs with heavy tailed degree distributions [31]. The above issue has been addressed using a simple, but efficient solution (referred to as the MinBucket algorithm [15, 31]): each vertex enumerates pairs of neighbors with degree no smaller than its own (with arbitrary tie breaking) and checks they are connected. It is not hard to see that this gives a correct count, and it has been empirically observed that this algorithm does not run into load balancing issues even on heavy tailed graphs [31]. The MinBucket heuristic has also been shown to give polynomial runtime improvement over the naive method when the input is a random graph with a power law degree distribution [5]. A recent work by Jha et al. [20] applies the degree based technique for counting 4-vertex queires. There are a few prior approaches for arbitrary queries of [7, 3, 26], but algorithms do not use degree information, and are comparable to the baseline algorithm used in our study.

To the best of our knowledge, prior to our work there has not been a systematic study of how MINBUCKET generalizes to larger subgraph counting problems. In this work we generalize the method for counting occurrences of treewidth 2 graphs, perform a thorough experimental evaluation and provide a theoretical runtime analysis of our technique in the random power law graph model. Our paper improves upon prior work along three axes: generality of queries handled, scalability of the proposed solution and theoretical analysis of the main algorithmic primitive on a class of graphs often used to model real world networks.

2 Preliminaries

Subgraph counting problem. The *subgraph counting problem* is defined as follows. The input consists of a *query graph* $\mathcal{Q} = (V_\mathcal{Q}, E_\mathcal{Q})$ over a set of k nodes and a data graph $G = (V_G, E_G)$ over a set of n vertices and m edges. The task is to count the number of (not necessarily induced) subgraphs of G that are isomorphic to $\mathcal{Q}$. Formally, count the number of injective mappings $\pi : V_\mathcal{Q} \to V_G$ such that for any pair of query nodes $a, b \in V_\mathcal{Q}$, if $\langle q_1, q_2 \rangle \in E_\mathcal{Q}$, then $\langle \pi(q_1), \pi(q_2) \rangle \in E_G$. We refer to such mappings π as *matches*.
Color coding and colorful matches. A *coloring* is a function $\chi : V_G \to \{1, 2, \ldots, k\}$, where for every vertex $u \in V_G$, $\chi(u)$ denotes its color. A match π from $V_\mathcal{Q}$ to V_G is *colorful* if the vertices of $\mathcal{Q}$ are mapped to k distinctly colored vertices in G, i.e. $\bigcup_{a \in V_\mathcal{Q}} \chi(\pi(a)) = \{1, 2, 3, \ldots, k\}$. The main idea is that instead of counting all possible matches of the k vertices of the query graph to the vertices of the data graph, one first *colors* the vertices of the data graph uniformly at random using k colors, and then searches for *colorful* matches.
Colorful subgraph counting problem. In the *colorful subgraph counting problem* the task is to count the number of colorful matches of the query $\mathcal{Q}$ in V_G.

Our setting counts the number of colorful matches or mappings from $\mathcal{Q}$ to the data vertices. Alternatively, we may want to count the number of colorful subgraphs that are isomorphic to $\mathcal{Q}$. The latter quantity can be obtained by dividing the former by $\texttt{aut}(\mathcal{Q})$, the number of automorphisms of $\mathcal{Q}$. While it is computationally hard to compute $\texttt{aut}(\mathcal{Q})$ for an arbitrary query graph, the quantity can be computed quickly for queries of relatively small size (say about 10 nodes). Given the above discussion, we focus on counting the number of colorful matches.
Treewidth. Intuitively, if the query graph $\mathcal{Q} = (V_\mathcal{Q}, E_\mathcal{Q})$ has treewidth t then $\mathcal{Q}$ can be decomposed into subgraphs $Q_1, Q_2, \ldots$ such that each subgraph Q_i is also of treewidth t, and each Q_i has no more than t nodes that belong also to other subgraphs. We call such nodes the *boundary nodes* of Q_i. In addition, the total number of distinct boundary nodes in all subgraphs $Q_1, Q_2, \ldots$ is at most $t + 1$. Note that the decomposition can be done recursively as each Q_i has treewidth t, until we are left only with subgraphs that have at most $t + 1$ nodes. This results in a *treewidth decomposition tree* denoted $\mathcal{T}_\mathcal{Q}$. A formal definition is givne below.

A tree decomposition of a query $|\mathcal{Q}|$ is a tree $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$, wherein each node $p \in V_\mathcal{T}$

is associated with a subset of query nodes $S(p) \subseteq V_{\mathcal{Q}}$, called pieces, such that the following properties are true: (i) for every query edge $(a, b) \in E_{\mathcal{Q}}$, there exists a piece $S(p)$ (for some $p \in V_{\mathcal{T}}$) that contains both a and b; (ii) for every query node $a \in V_{\mathcal{Q}}$, the set of nodes whose pieces contain a induce a connected subtree. Alternatively, the second property states that if a belongs to pieces $S(p_1)$ and $S(p_2)$ for some p_1 and p_2, then a must also belong to the piece $S(p)$ for any node p found on the (unique) path connecting p_1 and p_2 in $\mathcal{T}$. The width of the tree decomposition is the maximum cardinality overall pieces minus one, i.e., $\max_p |S(p)| - 1$. The treewidth t of the query is the minimum width over all its tree decompositions.

Approximate subgraph counting via color coding. Counting the number of colorful matches turns out to be easier than counting the actual (not necessarily colorful) matches. The price to pay is that the algorithm is randomized. We color the graph randomly and obtain the number of colorful matches, and repeat the process independently at random a few times. Then, an estimate for the number of matches (occurrences of the query) can be obtained by taking the average.

For a given input graph G and query $\mathcal{Q}$ let $n(G, \mathcal{Q})$ denote the number of matches π from $\mathcal{Q}$ to G. For a (random) coloring χ of vertices of G let $n^{colorful}(G, \mathcal{Q}, \chi)$ denote the number of colorful matches of $\mathcal{Q}$ to G under coloring χ. It was shown [2, 1] that with proper normalization the colorful count $n^{colorful}(G, \mathcal{Q}, \chi)$ is an unbiased estimator of the actual count. Specifically, the right normalization factor is $k^k/k!$, i.e. we have $(k^k/k!) \cdot \mathbf{E}_{\chi}[n^{colorful}(G, \mathcal{Q}, \chi)] = n(G, \mathcal{Q})$. The variance of the estimator can also be bounded (see [1], section 2.1). Thus, taking the average of $n^{colorful}(G, \mathcal{Q}, \chi)$ under a few independently chosen colorings χ converges to the right answer, i.e. $n(G, \mathcal{Q})$. Thus, in order to obtain an approximate subgraph counting algorithm it suffices to solve the colorful subgraph counting problem. The rest of the paper is devoted to designing a scalable solution to colorful subgraph counting.

3 Overview

The work of Alon et al. [2] yields a natural algorithm for the colorful subgraph counting problem on bounded treewidth query graphs. This algorithm is based on the following intuition. Suppose that we have found a colorful match π for a subgraph Q of the input query graph $\mathcal{Q}$, and we wish to extend it into a colorful match π' for $\mathcal{Q}$ by additionally fixing the mapping of the nodes outside Q. For this we do not need to know the mapping of the non-boundary nodes of Q, since they do not share edges with nodes outside Q. Instead, it suffices to know the mapping of the boundary nodes (i.e., the nodes that share edges with nodes outside Q) and the set of colors used by π. The mapping of the boundary nodes is needed to ensure that for any edge from a boundary node to outside, the corresponding data vertices share an edge in the data graph; and the set of colors is needed to avoid repeating a color already used by π. Analogously, in the setting of counting, in order to count the number of colorful matches for $\mathcal{Q}$, we do not need a complete listing of colorful matches of Q. Instead, we can group the colorful matches based on the set of colors used and the mappings for the boundary nodes and it suffices to know the count per group.

Based on the above intuition, we apply dynamic programming to count the number colorful matches of $\mathcal{Q}$. Let $\mathcal{T}_\mathcal{Q}$ be the tree decomposition of $\mathcal{Q}$ with treewith t. The algorithm processes $\mathcal{T}_\mathcal{Q}$ in a bottom-up manner and a creates a hash table (that we call a projection table) for each tree node. The subgraph graph Q associated with a node has at most t boundary nodes and these nodes can be mapped to the data vertices in at most n^t ways. In addition, we need to record the colors of the data vertices to which the nodes of Q are mapped. Since we focus on colorful matches, the set of colors used (that we call "signature") can be at most $\binom{k}{t} \leq 2^k$ (where k is the size of the query graph). For each combination of mappings to the boundary nodes and the signature, we record the number of colorful matches of Q consistent with the combination. The number of entries in the table is at most $n^t 2^k$. The projection table for a tree node can be computed from those of its children. We get the total number of colorful matches by performing an aggregation on the projection table of the root node.

Working in the realm of motif counting, Slota and Madduri [30] described an efficient distributed implementation of the above algorithm for the case of tree queries and presented an experimental evaluation. Trees have treewidth one hence, the size of projection tables is linear in the number of vertices and the overall computation can be carried out in time linear in the graph size. Our goal is to address a more general class of queries (beyond trees) in a distributed setting and we focus on the case of queries of treewidth 2. Treewidth 2 queries are more challenging since in the worst case, the tables can be of size quadratic in the number vertices and the computation time also gets quadratic.

The construction of our algorithm is motivated by the fact that real life data graphs tend to exhibit variations in the degree distribution. A naive implementation that treats all data vertices in the same manner would result in a lot of entries in the projection tables of the high degree vertices that do not lead to colorful matches for the overall input query. Moreover, in a distributed setting the processors owning such vertices perform more computation leading to load imbalance.

Our algorithm is based on a crucial observation that any treewidth 2 query can be recursively decomposed into (annotated) cycles or leaves. The core component of the algorithm is an efficient procedure for handling cycles that employs a strategy based on degree based ordering of vertices. This leads to reduction in wasteful computation, as well as improved load balancing. The procedure is inspired by a similar strategy used in prior work [5] for handling triangles. The overall algorithm uses the above decomposition and the improved procedure for handling cycles.

4 Overall Algorithm

In this section we describe the overall structure of our subgraph counting algorithm that proceeds in two steps. In the **first step**, we decompose the query into cycles and leaves (called blocks) and construct a *decomposition tree* for the input query $\mathcal{Q}$ which is essentially a carefully chosen treewidth decomposition tree; each node of the tree represents a block and encodes a convenient subquery. This step is independent of the data graph and can be viewed as a preprocessing phase for the query. Then in the **second step** we traverse the tree in a bottom up manner, performing primitive counting operations over the data graph prescribed by the internal nodes and combining the results. The final count is produced by the root of the tree.

4.1 Decomposition Tree

For an input query graph $\mathcal{Q} = (V_{\mathcal{Q}}, E_{\mathcal{Q}})$, construct the decomposition tree $T(\mathcal{Q})$ by iteratively applying one of two primitive operations: contraction of a leaf edge or a cycle. As these operations are applied the number of nodes in the query $\mathcal{Q}$ decreases. At the same time new edges may appear in $\mathcal{Q}$ to represent contracted structures, and edges as well as nodes may get annotated with the identity of the contracted structures that they represent. Before defining the tree construction algorithm we need to introduce two definitions. First, we say that a cycle $\mathcal{C}$ in $\mathcal{Q}$ is *contractible* if **(a)** $\mathcal{C} = (a_0, a_1, \ldots, a_{L-1})$ is induced (i.e. there are no edges between nodes $a_0, a_1, \ldots, a_{L-1}$ except the edges of $\mathcal{C}$) and **(b)** cycle $\mathcal{C}$ has most two boundary nodes (i.e., nodes that share edges with nodes outside of $\mathcal{C}$). Second, a *leaf edge* is an edge $\mathcal{L} = (a, b)$, where b is a leaf node (has degree one); a is called the boundary node of the leaf edge. We use the common term *block* to refer to leaf edges and contractible cycles.

For example, consider the query named `Satellite` in Fig 2. The cycle (i, j, k) is contractible with a single boundary node i, the cycle (a, b, c, d, e) is contractible with two boundary nodes a and c, and (f, h) is a leaf edge. The cycle (i, f, g) is not contractible since it has three boundary nodes.

We construct the decomposition tree $T(\mathcal{Q})$ starting with an empty tree. The tree is built bottom-up starting from the leaf level and hence, the structure may be a forest with multiple roots in the intermediate stages. Each iteration adds a new node and may make some of the existing roots as its children, culminating in a tree.

In the construction process we iteratively perform the following operations until $\mathcal{Q}$ contains a single node: find a block B (a leaf edge or a contractible cycle) in $\mathcal{Q}$ and remove it from $\mathcal{Q}$ (while possibly adding an edge to $\mathcal{Q}$), and add a corresponding node to $T(\mathcal{Q})$. We iterate until $\mathcal{Q}$ contains a single node. We distinguish 3 cases.

Case 1: *B is a contractible cycle $\mathcal{C}$ with exactly one boundary node $a \in V_{\mathcal{Q}}$:* Remove the nodes and edges of $\mathcal{C}$ from $\mathcal{Q}$, except for node a. Erase any annotation found on a in $\mathcal{Q}$ and annotate it with the block name B.

Case 2: *B is a contractible cycle $\mathcal{C}$ with two boundary nodes $a, b \in V_{\mathcal{Q}}$:* Remove the nodes and edges of $\mathcal{C}$ from $\mathcal{Q}$, except for the nodes a and b. Add an edge (a, b) in $\mathcal{Q}$ and annotate it with B. Erase any annotation found on a and b in $\mathcal{Q}$.

Case 3: *B is a leaf edge $\mathcal{L} = (a, b)$:* Remove b and the edge from $\mathcal{Q}$. Erase any annotation found on node $a \in \mathcal{Q}$ and annotate it with the block name B.

The nodes and edges of B inherit the annotations from $\mathcal{Q}$, as they were before $\mathcal{Q}$ was transformed (this ensures that the annotations on the boundary nodes that got erased get captured by the new annotation).

Next we add a new node B to the tree $T(\mathcal{Q})$. If any node or edge in B has an annotation B', make B' a child of B in $T(\mathcal{Q})$. This completes the construction of T. We show below that the process can find a block in each iteration and terminate successfully on every query of treewidth 2. Assuming termination, it is not difficult to see that the process produces a tree. During contraction, every block B' annotates a particular node or an edge of $\mathcal{Q}$, recording the way in which it has been contracted. The annotation gets inherited by

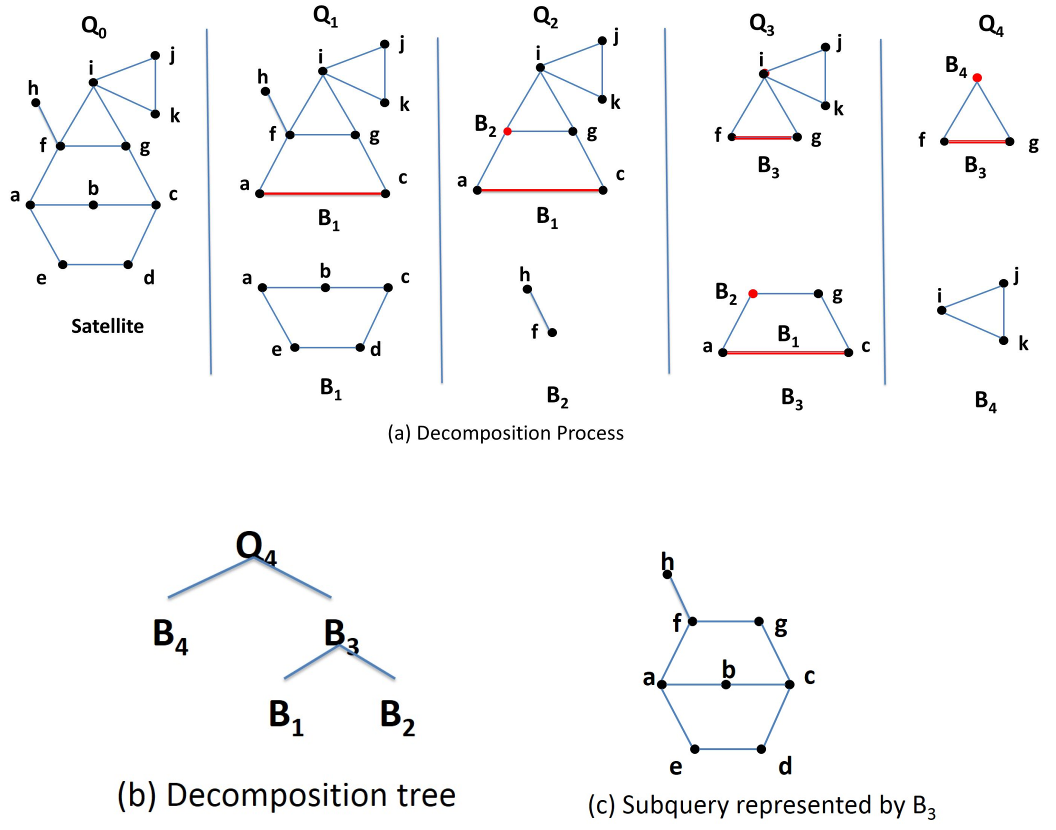


Figure 2: Illustration of the decomposition process. The top row shows the sequence of queries considered in the process (the original query is on the left), the bottom row shows the blocks that were contracted in each step.

some other block B in a subsequent iteration. The block B becomes the parent of B'. The annotation is erased in $\mathcal{Q}$, ensuring that no other block becomes a parent of B'.

Taking `Satellite` as the input query $\mathcal{Q}$, Figure 2 provides an illustration process, along with the output decomposition tree. The bottom row shows the blocks being contracted and the top row shows the transformed $\mathcal{Q}$. The first iteration contracts the cycle $B_1 = (a, b, c, d, e)$. A new edge (a, c) is added to $\mathcal{Q}$, along with the annotation B_1, and B_1 is added to the tree. The second iteration contracts the leaf block $B_2 = (f, h)$. Node f is annotated as B_2 and the B_2 is added to the tree. The third iteration contracts $B_3 = (a, f, g, c)$, by adding an edge (f, g) with the annotation B_3. The block is added to the tree and it is made the parent of B_1 and B_2. In the fourth iteration, the cycle $B_4 = (i, j, k)$ is contracted. Node i gets annotated as B_4 and B_4 is added to the tree. Finally, the query Q_4 is contracted leaving $\mathcal{Q}$ empty. We add Q_4 as the root of the tree, making it the parent of B_3 and B_4.

The following lemma guarantees that for any treewidth 2 query $\mathcal{Q}$, the tree construction procedure will always find a block (a leaf edge or a contractible cycle) in each iteration and terminate successfully. The proof relies on prior work on nested ear decompositions of treewidth 2 queries [16].

Lemma 4.1 *(i) Any treewidth 2 query $\mathcal{Q}$ contains a block; (ii) the transformed query resulting from the contraction process is also a treewidth 2 query.*

Proof: We first prove part (ii) of the lemma. If the contracted block has one boundary node then no new edges are added to $\mathcal{Q}$, in which case the tree $\mathcal{T}_{\mathcal{Q}}$ for the updated $\mathcal{Q}$ is given by deleting all the nodes not in the updated $V_{\mathcal{Q}}$ from the subsets $S_{\mathcal{Q}}(t)$. If the contracted block has two boundary nodes a and b then the edge (a, b) is added to $\mathcal{Q}$. In this case we get the tree for the updated $\mathcal{Q}$ by replacing each occurrence of the nodes not in the updated $V_{\mathcal{Q}}$ by b. Note that the size of each subset is still at most 3, nodes associated with subsets that contain b form a connected component, and for at least one subset $S_{\mathcal{Q}}(t)$, $\{a, b\} \subseteq S_{\mathcal{Q}}(t)$.

We now prove part (i). First, Root the tree $\mathcal{T}_{\mathcal{Q}}$ at an arbitrary non-leaf node. This induces an ancestor-descendant relationship on the nodes in $V_{\mathcal{T}}$. Note that if there are two nodes $\{t, t'\} \subseteq V_{\mathcal{T}}$, such that $S_{\mathcal{Q}}(t') \subseteq S_{\mathcal{Q}}(t)$, node t' can be omitted and all its children connected to t. Thus from now on we assume that no subset $S_{\mathcal{Q}}(t)$ is contained (or identical) to another subset.

We need the following definition and claim.

Definition 4.1 *For a node $t \in V_{\mathcal{T}}$, let $\mathcal{Q}_t$ be the subgraph of $\mathcal{Q}$ induced by the nodes that are in the union of the subsets associated with the nodes of $\mathcal{T}_{\mathcal{Q}}$ in the subtree rooted at t.*

Claim 4.1 *For every node $t \in V_{\mathcal{T}}$, either $\mathcal{Q}_t$ contains a block, or $\mathcal{Q}_t$ is a path whose endpoints are in the subset associated with the parent of t (if such exists).*

Before proving the claim we show how it implies the lemma. Since the claim holds also for the root of $\mathcal{T}_{\mathcal{Q}}$ then either $\mathcal{Q}$ contains a block or it is a path in which case it also contains a leaf block. ■

Proof of Claim 4.1: We prove the claim by induction. The base of the induction is a leaf node. Consider a leaf node $t \in V_{\mathcal{T}}$. There are two possibilities: (i) $S_{\mathcal{Q}}(t) = \{x, y\}$, and (ii) $S_{\mathcal{Q}}(t) = \{x, y, z\}$. If $S_{\mathcal{Q}}(t) = \{x, y\}$, then at least one node, say y, is only connected to x and thus (x, y) is a leaf edge.

If $S_{\mathcal{Q}}(t) = \{x, y, z\}$, then consider the subgraph induced by $\{x, y, z\}$. If this subgraph is a triangle then it must be a contractible cycle. The only remaining case is the subgraph induced by $\{x, y, z\}$ forms a path. Assume that the endpoints of this path are x and z. If one of these endpoints, say z, is not in the subset associated with the parent of t then (y, z) is a leaf edge. Otherwise, let t' be the parent of t, we have $S_{\mathcal{Q}}(t) \cap S_{\mathcal{Q}}(t') = \{x, z\}$.

For the inductive step consider a non-leaf node $t \in V_{\mathcal{T}}$. If $\mathcal{Q}_{t'}$ for any child t' of t contains a block then we are done. Assume that this is not the case. Consider first the case that t has a single child t'. By the inductive hypothesis $\mathcal{Q}_{t'}$ is a path whose endpoints x and y are in $S_{\mathcal{Q}}(t)$. Let $S_{\mathcal{Q}}(t) = \{x, y, z\}$. If z is connected to both x and y then the cycle closed by z is a contractible cycle. If z is connected to only one endpoint, say y, then we get a path with endpoints x and z. If either x or z are not in the subset associated with the parent of t, then the missing endpoint is leaf node. If both x and z are in the subset associated with the parent of t then the inductive claim follows.

Next, Consider the case that t has several children. If two of the children of t, say t' and t'', share endpoints then the cycle formed by $\mathcal{Q}_{t'}$ and $\mathcal{Q}_{t''}$ is contractible. Otherwise, t must have exactly two children, say t' and t'', with endpoint $\{x, y\}$ and $\{y, z\}$, forming a path with endpoints x and z. If z is connected also to x then the cycle closed by the edge (x, z)

is a contractible cycle. If either x or z are not in the subset associated with the parent of t, then the missing endpoint is a leaf node. If both x and z are in the subset associated with the parent of t then the inductive claim follows. ∎

An input query may admit multiple decomposition trees and the choice of the tree influences the performance of our algorithm. In Section 6, we present a heuristic for finding a good decomposition. Each node of the tree represents a block and it will be convenient to view to the node simply as the block represented by it.

At this point, it is interesting to consider tree queries studied by Slota and Madduri [30]. Given a tree query, their algorithm fixes a suitable query node as the root and iteratively processes the tree in a bottom-up manner. The algorithm implicitly uses a decomposition tree. However, since trees do not have cycles, the decomposition tree consists of only leaf edge blocks. In contrast, the decomposition trees of treewidth two queries involve the more challenging case of cycles as well.

4.2 Tree Traversal

Here, we describe the second step of the algorithm that traverses the decomposition tree in a bottom-up manner and computes the number of colorful matches of the blocks in the data graph. For this purpose, we define the notion of subqueries represented by blocks.

A *subquery* Q of the input query $\mathcal{Q}$ refers to any induced subgraph of $\mathcal{Q}$. Consider a block B and let U be the union of nodes found in the block B and its descendant blocks in the tree. The *subquery represented by* B, denoted $\mathsf{SQ}(B)$, refers to the subquery induced by U. For example, Figure 2 shows the subquery represented by the block B_4. The decomposition tree yields a nested hierarchy of subqueries: the root block represents the whole input query and for any block B with the parent B', the subquery $\mathsf{SQ}(B)$ is contained within $\mathsf{SQ}(B')$.

Let B be a block. A node $a \in \mathsf{SQ}(B)$ is said to be a *boundary node*, if a shares an edge with a node outside $\mathsf{SQ}(B)$. It is not hard to see that these boundary nodes are the same as the boundary nodes of B (identified during the tree construction process). Thus, $\mathsf{SQ}(B)$ can have at most two boundary nodes.

Before describing the counting algorithm we extend the notion of colorful matches to subqueries: a colorful match for a subquery $Q = (V_Q, E_Q)$ is an injective mapping $\pi : V_Q \to V_G$, such that for any edge $(a, b) \in E_Q$, $(\pi(a), \pi(b)) \in E_G$, and the vertices of Q are mapped to distinctly colored vertices of G.

The algorithm traverses the tree in a bottom-up manner. For each block B, it outputs a succinct synopsis of the set of colorful matches of the subquery $\mathsf{SQ}(B)$, using a projection table and signature (as outlined in Section 3). that we now define precisely.

Signature: Let $K = \{1, 2, \ldots, k\}$ denote the set of colors used in the data graph, where k is the size of the input query $\mathcal{Q}$. The term *signature* refers to any subset $\alpha \subseteq K$. For a subquery Q and a colorful match π of Q, the *signature* of π refers to the set of colors of the data vertices used by π and it is denoted $\mathtt{sig}(\pi)$, i.e., $\mathtt{sig}(\pi) = \cup_{a \in Q}\{\chi(\pi(a))\}$.

Projection Tables: Let Q be subquery with two boundary nodes a and b. For a pair of data vertices u and v and a signature $\alpha \subseteq K$ let $\mathtt{cnt}(u, v, \alpha|Q)$ denote the number of colorful matches of Q wherein the boundary nodes a and b are mapped to u and v and the

```
Overall Algorithm
  1. Compute a decomposition tree $T(Q)$ for the input query $Q$.
  2. Traverse the tree bottom-up.
     For each non-root block $B$:
        Use the projection tables of the children blocks of $B$ and
           compute the projection table for $B$
  3. Output the number of colorful mathes of the subquery
           represented by the root-block.
```

Figure 3: Overall Algorithm

signature of π is α:

$$\texttt{cnt}(u, v, \alpha | Q) = |\{\pi \in \Pi \ : \ \pi(a) = u \text{ and } \pi(b) = v \text{ and } \texttt{sig}(\pi) = \alpha\}|,$$

where Π is the set of all the colorful matches of Q.

These counts can be conveniently represented in the form a hash table with (u, v, α) forming the key and the count forming the value. We refer to any encoding of the above counts (such as the hash table above) as the *projection table* of Q. In the worst case, the table may have size quadratic in the input data graph. However, a significant fraction of the triplets will have a count of zero and we maintain only the non-zero counts.

The projection table for subqueries having a single boundary node a is defined in a similar manner. For a data vertex u and a signature $\alpha \subseteq K$, define

$$\texttt{cnt}(u, s | Q) = |\{\pi \in \Pi \ : \ \pi(q) = u \text{ and } \texttt{sig}(\pi) = \alpha\}|.$$

4.3 Computing the Counts

Given a decomposition tree, the algorithm works based on the fact that the projection table for a block can be computed by joining the projection table of its children blocks.

As an illustration of the idea, consider the block B_3 having boundary nodes f and g, and the subquery represented by it (Figure 2). For a pair of vertices u and v, and a signature α, the projection count $\texttt{cnt}(u, v, \alpha | B_3)$ can be computed as follows. The block consists of the path (a, f, g, c), and any match π for the subquery must map these nodes to vertices (x, u, v, y) that form a path in the data graph. The block is annotated by its children blocks B_1 with boundary nodes a and c, and B_2 with boundary node f. Any pair of matches π_1 and π_2 for $\mathsf{SQ}(B_1)$ and $\mathsf{SQ}(B_2)$ can be extended as matches for $\mathsf{SQ}(B_3)$, as long as their signatures α_1 and α_2 are disjoint (since the blocks do not share any node) and are contained within α. Therefore, we can derive the desired count by performing the following aggregation over all quadruples $(x, y, \alpha_1, \alpha_2)$ satisfying the properties: (x, u, v, y) forms a path in the data graph; $\alpha_1, \alpha_2 \subseteq \alpha$; $(\alpha_1 \cap \alpha_2)$ is empty. The aggregation is:

$$\texttt{cnt}(u, v, \alpha | B_3) = \sum_{x,y} \sum_{\alpha_1, \alpha_2} \texttt{cnt}(x, y, \alpha_1 | B_1) \times \texttt{cnt}(u, \alpha_2 | B_2).$$

We can express the projection counts for any block in the above manner. However, as the number of children increases, the cartesian product involved in the aggregation

```
**Procedure 1: Computing Projection Table for $P^+$**
  For each edge $(u, v)$ in the data graph $G$
    $\texttt{cnt}(u, v, \alpha | P^+_{p,p\oplus 1}) \leftarrow 1$, where $\alpha = \{\chi(u), \chi(v)\}$.
  For $j = p \oplus 2, p \oplus 3, \ldots, q$
    For each triple $(u, v, \alpha)$ with $\texttt{cnt}(u, v, \alpha | P^+_{p,j\ominus 1}) \neq 0$
      For each edge $(v, w)$ in $G$ such that $\chi(w) \notin \alpha$ do:
        Let $\alpha' = \alpha \cup \{\chi(w)\}$.
        Increment $\texttt{cnt}(u, w, \alpha' | P^+_{p,j})$ by $\texttt{cnt}(u, v, \alpha | P^+_{p,j\ominus 1})$.

**Procedure 2: Computing Projection Table for $\mathcal{C}$**
  For each entry $(u, v, \alpha_1)$ with $\texttt{cnt}(u, v, \alpha_1 | P^+) \neq 0$
    For each entry $(u, v, \alpha_2)$ with $\texttt{cnt}(u, v, \alpha_2 | P^-) \neq 0$
      If $\alpha_1 \cap \alpha_2 = \{\chi(u), \chi(v)\}$
        $\alpha' \leftarrow \alpha_1 \cup \alpha_2$
        $\mathsf{val}_1 \leftarrow \texttt{cnt}(u, v, \alpha_1 | P^+)$;    $\mathsf{val}_2 \leftarrow \texttt{cnt}(u, v, \alpha_2 | P^-)$
        Increment $\texttt{cnt}(u, v, \alpha' | \mathcal{C})$ by $\mathsf{val}_1 \times \mathsf{val}_2$.
```

Figure 4: PS Algorithm

would be prohibitively expensive. Our procedures efficiently simulate the aggregation by performing a sequence of join operations involving the projection tables of children blocks.

Given a decomposition tree, the algorithm traverses the decomposition tree in a bottom-up manner, computing the projection tables for all the blocks and culminates in the root-block representing the whole input query. At this step, instead of producing a projection table, the algorithm simply computes the number of colorful matches. The pseudo-code is shown in Figure 3.

5 Solving Blocks

The main step of the algorithm is the construction of the projection tables of a block from its children blocks. In this section we develop efficient procedures for handling cycles. For the sake of highlighting the main ideas, we first focus on the case of cycles found at a leaf level of the decomposition tree (such as the cycle B_1 in Figure 2); these cycles do not have other blocks annotating them. General cycles are handled by extending these ideas as discussed later.

5.1 Solving Cycles at the Leaf Level

Consider a cycle block $\mathcal{C} = (a_0, \ldots, a_{L-1})$ of length L without annotations. The cycle may have at most two boundary nodes. We discuss the more interesting case where the number of boundary nodes is exactly two; the other cases are handled in a similar fashion. Let the two boundary nodes of the cycle be a_p and a_q, for some $0 \leq p, q \leq L-1$. We present two procedures for computing the projection table of $\mathcal{C}$: a baseline procedure that uses a path splitting strategy and an efficient procedure guided by a degree based ordering of vertices.
Path Splitting Algorithm (PS). For two nodes a_s and a_t on the cycle, let $P^+_{s,t}$ and $P^-_{s,t}$ be the paths obtained by traversing the cycle from a_s to a_t in the clockwise and counter-clockwise

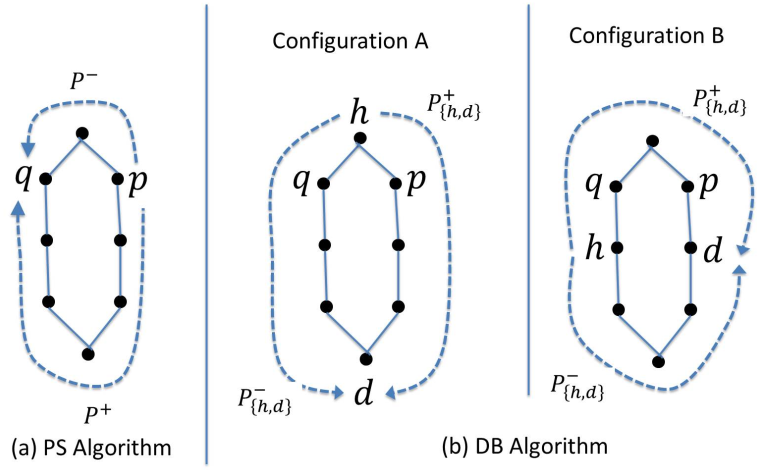


Figure 5: PS and DB Illustrations.

directions, respectively, i.e., $P^+_{s,t} = (a_s, a_{s\oplus 1}, \ldots, a_t)$ and $P^-_{s,t} = (a_s, a_{s\ominus 1}, \ldots, a_t)$, where $\oplus$ and $\ominus$ refer to addition and subtraction modulo L.

Let $\texttt{cnt}(\cdot,\cdot,\cdot|P^+_{s,t})$ denote the projection counts for path $P^+_{s,t}$ taking a_s and a_t as the boundary nodes. Namely, for a triple (u, v, α), let $\texttt{cnt}(u, v, \alpha|P^+_{s,t})$ denote the number of colorful matches for $P^+_{s,t}$ wherein $\pi(a_s) = u$, $\pi(a_t) = v$ and $\texttt{sig}(\pi) = \alpha$. A similar notion is defined for the paths $P^-_{s,t}$.

The procedure splits the cycle into two paths along the boundary nodes, given by $P^+_{p,q}$ and $P^-_{p,q}$; we refer to these special paths as P^+ and P^-. See Fig 5 (a) for an illustration.

The projection table for P^+ is constructed iteratively, by building the tables for the paths $P^+_{p,j}$, for each node a_j found along the path. This is accomplished by extending the projection table for the prior path $P^+_{p,j\ominus 1}$ via a join with the edges of the data graph. The pseudocode is given in Figure 4 (Procedure 1). We assume that all the counts are initialized to zero. The first iteration is handled by directly reading the edges of the data graph. In the subsequent iterations, we extend every triple (u, v, α) with non-zero count $\texttt{cnt}(u, v, s|P^+_{p,j\ominus 1})$, with any edge (v, w), provided the resulting match is colorful. The counts for P^- are constructed analogously. Finally, the projection table for the cycle $\mathcal{C}$ is obtained by joining the counts of P^+ and P^-, as shown in Procedure 2. Here, a pair of triples (u, v, α_1) and (u, v, α_2) are joined, if the resulting match is colorful.

Discussion of baseline. As discussed below (Section 5.2), the PS procedure can be extended to handle general cycles with annotations, and yields an algorithm for handling treewidth 2 queries. The resultant PS algorithm is equivalent to the original color coding algorithm of Alon et al. [2]. Prior work [30, 1] on colorful subgraph counting utilize the algorithm of Alon et al. as the basis for counting tree queries (treelets). We developed a distributed implementation of the PS algorithm, and use it as the baseline in our experimental study. Known techniques for subgraph counting with large queries (e.g. [7, 26]) employ similar graph traversal techniques, making PS consistent with the state of the art for subgraph counting as well as color coding.

We develop an procedure, called Degree Based (DB) algorithm, that outperforms the PS algorithm for practical graphs and queries. It is motivated by the following observations. First, the paths P^+ and P^- may have uneven lengths (for instance, in Figure 5), $|P^+| = 6$ and $|P^-| = 2$) and the processing of the longer path dominates the overall running time. Second, in real-graphs with skewed degree distributions, high degree vertices tend to have more paths passing through them, which populate the projection tables of P^+ and P^-. However, significant fraction of these paths do not find appropriate counterparts in the other table to complete a match, leading to wasteful computations. Third, in a distributed setting, the above phenomenon manifests as higher load on processors owning high degree vertices, leading to load imbalance.

It is not difficult to address the first issue alone. The only intricacy is that when the paths are split evenly, the boundary nodes may appear internally on the the paths (see Figure 5 with a split across nodes denoted h and d). This can be handled by recording the mapping for the boundary nodes as part of the projection counts. We implemented the above algorithm as well and noticed that the issue of wasteful computations and load imbalance still persists. And furthermore, performance of the PS algorithm and the modified implementations does not differ significantly on our benchmark graphs and queries.

Degree Based Algorithm (DB**).** The DB algorithm addresses all the three issues by using the strategy of building the paths from high degree vertices.

Arrange the data vertices in the increasing order of their degree; if two vertices have the same degree, the tie is broken arbitrarily, say by placing the vertex having the least id first. We say that a vertex u is *higher* than a vertex v, if u appears after v in the above ordering and this is denoted "$u \succ v$".

Consider the input cycle $\mathcal{C} = (a_0, a_1, \ldots, a_{L-1})$ with boundary nodes a_p and a_q and let π be a colorful match for $\mathcal{C}$ that maps the above nodes to data vertices $u_0, u_1, \ldots, u_{L-1}$, respectively. Among these data vertices, let u_j be the highest vertex. We refer to the corresponding node a_j as the *highest* node of π.

The idea is to partition the set of colorful matches into L groups based on their highest node a_h and compute the projection table for each group separately. For a pair of data vertices u and v, and a signature α, let $\texttt{cnt}(u, v, \alpha|\mathcal{C}, \mathrm{hi} = h)$ denote the number of colorful matches of π for $\mathcal{C}$, wherein $\pi(a_p) = u$, $\pi(a_q) = v$, $\texttt{sig}(\pi) = \alpha$ and a_h is the highest node of π. The projection table for $\mathcal{C}$ can be obtained by aggregating the above counts: for any triple (u, v, α),

$$\texttt{cnt}(u, v, \alpha|\mathcal{C}) \quad = \quad \sum_{h=0}^{L-1} \texttt{cnt}(u, v, \alpha|\mathcal{C}, \mathrm{hi} = h). \qquad (1)$$

We next describe an efficient procedure for computing the counts $\texttt{cnt}(u, v, \alpha|\mathcal{C}, \mathrm{hi} = h)$. The concept of high starting matches plays a crucial role in the procedure. Let a_d be the node diagonally opposite to a_h on the cycle, i.e., $d = h \oplus \lfloor L/2 \rfloor$. The procedure splits the cycles into two paths $P^+_{h,d}$ and $P^-_{h,d}$. Figure 5 (b) shows the paths for two sample values of h. Let a_j be a node found on the path $P^+_{h,d}$. A colorful match π for $P^+_{h,j}$ is said to be *high-starting*, if the data vertex $\pi(a_h)$ is higher than all the other data vertices used by π, i.e., $\pi(a_h) \succ \pi(a_i)$, for all nodes a_i on the path $P^+_{h,j}$. For a pair of vertices u and v, and a signature α, let $\texttt{cnt}^*(u, v, \alpha|P^+_{h,j})$ denote the number of high-starting colorful matches for the path $P^+_{h,j}$ wherein $\pi(a_h) = u$, $\pi(a_j) = v$ and $\texttt{sig}(\pi) = \alpha$.

```
Procedure 1: Compute cnt*$(u, v, \alpha | P^{+}_{h,d})$
For each edge $(u, v)$ in the data graph $G$ with $u \succ v$
    cnt*$(u, v, \alpha | P^{+}_{h,h\oplus 1}) \leftarrow 1$, where $\alpha = \{\chi(u), \chi(v)\}$.
For $j = h \oplus 2, a \oplus 3, \ldots, d$
    For each triple $(u, v, \alpha)$ with cnt*$(u, v, \alpha | P^{+}_{h,j\ominus 1}) \neq 0$
        For each edge $(v, w)$ in $G$ s.t. $u \succ w$ and $\chi(w) \notin \alpha$:
            Let $\alpha' = \alpha \cup \{\chi(w)\}$.
            Incr. cnt*$(u, w, \alpha' | P^{+}_{h,j})$ by cnt*$(u, v, \alpha | P^{+}_{h,j\ominus 1})$.

Procedure 2: Compute cnt*$(x, y, \alpha | \mathcal{C}, \mathrm{hi} = h)$ for Config. (A)
For each entry $(u, v, x, \alpha_1)$ with cnt*$(u, v, x, \alpha_1 | P^{+}_{h,d}) \neq 0$
    For each entry $(u, v, y, \alpha_2)$ with cnt*$(u, v, y, \alpha_2 | P^{-}_{h,d}) \neq 0$
        If $\alpha_1 \cap \alpha_2 = \{\chi(u), \chi(v)\}$
            $\alpha' \leftarrow \alpha_1 \cup \alpha_2$
            $\mathsf{val}_1 \leftarrow$ cnt*$(u, v, x, \alpha_1 | P^{+}_{h,d})$;
            $\mathsf{val}_2 \leftarrow$ cnt*$(u, v, y, \alpha_2 | P^{-}_{h,d})$
            Incr. cnt*$(x, y, \alpha' | \mathcal{C}, \mathrm{hi} = h)$ by $\mathsf{val}_1 \times \mathsf{val}_2$.
```

Figure 6: DB Algorithm

We then count the high-starting colorful matches for the two paths, which can be accomplished via edge extensions, as in the PS algorithm. However, the current setting offers a crucial advantage: we can dictate that the starting node a_h is the highest node, meaning whenever an entry (u, v, α) gets extended by an edge (v, w), we can impose the condition that u is higher than w in the degree based ordering. Imposing the condition leads to a significant pruning of the tables. The pseudo-code is given in Figure 6 (Procedure 1).

While the degree based strategy is more efficient, we need to address an intricacy regarding the projection aspects. In contrast to the PS algorithm, the DB algorithm splits at the highest node and consequently, the boundary nodes p and q may appear inside the paths. Thus, in order to get the projection counts on p and q, we also need to explicitly record the mappings for the boundary nodes.

The two nodes a_p and a_q may occur on either $P^{+}_{h,d}$ or $P^{-}_{h,d}$. Six different configurations are possible, of which two are shown in Figure 5 (b). In Configuration (A), the paths include one boundary each, whereas in the second configuration, the same path includes both the boundary nodes. The other four configurations are symmetric: the boundary nodes may swap the paths in which they occur and in Configuration (B) can also reverse the order in which they occur. We discuss the two configurations shown in the figure; the other configurations are handled in a similar fashion.

Consider configuration (A). In order to record the mappings of the boundary node a_p, we introduce an additional field in the projection counts. For a triple of data vertices u, v and x, and a signature α, let cnt*$(u, v, x, \alpha | P^{+}_{h,d})$ denote the number of high-starting matches π for $P^{+}_{h,d}$ with $\pi(a_h) = u$, $\pi(a_d) = v$, $\pi(a_p) = x$ and $\mathtt{sig}(\pi) = \alpha$. These counts are computed in a manner similar to the base procedure shown in Figure 6 (Procedure 1); however, when the process encounters the boundary node p (namely, the initialization step or $j = p$), the mapped vertex (v or w, respectively) is recorded in the addi-

```
**Compute Projection Table for** $P^+_{h,d}$
Let $B$ be the block annotating the edge $(a_h, a_{h\oplus 1})$
$\mathtt{cnt}^*(\cdot,\cdot,\cdot|P^+_{h,h\oplus 1}) = \mathtt{cnt}^*(\cdot,\cdot,\cdot|B)$
For $j = h\oplus 1, h\oplus 2, \ldots, d$
   Execute **NodeJoin**$(a_j)$
   Execute **EdgeJoin**$(a_j)$
Execute **NodeJoin**$(a_d)$

**NodeJoin**$(a_j)$:
If $a_j$ is annotated by a block $B$
   For each $(u, v, \alpha_1)$ with $\mathtt{cnt}^*(u, v, \alpha_1|P^+_{h,j}) \neq 0$
      For each $(v, \alpha_2)$ with $\mathtt{cnt}(v, \alpha_2|B) \neq 0$
         If $(\alpha_1 \cap \alpha_2 = \{\chi(v)\}$
            $\alpha \leftarrow \alpha_1 \cup \alpha_2$
            $\mathsf{val}_1 \leftarrow \mathtt{cnt}^*(u, v, \alpha_1|P^+_{h,j})$;    $\mathsf{val}_2 \leftarrow \mathtt{cnt}(v, \alpha_2|B)$
            Incr. $\mathtt{cnt}^*(u, v, \alpha|P^+_{h,j})$ by $\mathsf{val}_1 \times \mathsf{val2}$

**EdgeJoin**$(a_j)$
For each entry $\mathtt{cnt}^*(u, v, \alpha_1|P^+_{h,j}) \neq 0$
   For each entry $\mathtt{cnt}(v, w, \alpha_2|B) \neq 0$ and $u \succ w$
      If $(\alpha_1 \cap \alpha_2 = \{\chi(v)\}$
         $\alpha \leftarrow \alpha_1 \cup \alpha_2$
         $\mathsf{val}_1 \leftarrow \mathtt{cnt}^*(u, v, \alpha_1|P^+_{h,j})$;    $\mathsf{val}_2 \leftarrow \mathtt{cnt}(v, w, \alpha_2|B)$
         Incr. $\mathtt{cnt}^*(u, w, \alpha|P^+_{h,j\oplus 1})$ by $\mathsf{val}_1 \times \mathsf{val}_2$
```

Figure 7: DB Procedure for General Cycle Blocks

tional field. The analogous counts for P^- can derived in a similar manner. The value of $\mathtt{cnt}^*(u, v, \alpha|\mathcal{C}, \mathrm{hi} = h)$ is obtained by joining the two; see Procedure (2) in Figure 6. Configuration (B) is handled in a similar fashion, except that we need two additional fields to record the mappings for both the boundary nodes. Namely, we maintain counts having keys of the form (u, v, x, y) representing the mapping of the nodes h, d, q and p to the vertices u, v, x and y. Procedure (2) is also adjusted accordingly. Finally, we can get the projection table $\mathtt{cnt}(u, v, \alpha|\mathcal{C})$ via aggregation, as in Equation 1.

5.2 Solving General Blocks

In this section, we present procedures for handling generic blocks. We first consider the case of cycle blocks with two boundary nodes.

Consider a generic cycle $\mathcal{C} = (a_0, a_1, \ldots, a_{L-1})$ having two boundary nodes a_p and a_q, whose nodes and edges may be annotated with other blocks (children of $\mathcal{C}$ in the decomposition tree). All these blocks have at most two boundary nodes and these are found on $\mathcal{C}$. For such any block B, the subquery represented by B has the same boundary nodes as that of B. Thus, we can get the projection table for $\mathcal{C}$ by joining the projection tables of the subqueries represented by the above blocks, as described below.

As before, we consider each possible choice for the highest node a_h and split the cycle into two paths $P^+_{h,d}$ and $P^-_{h,d}$. The path segment $P^+_{h,d}$ also represents a subquery (induced by the union of the nodes found in the path and the blocks annotating path). Thus, we can extend the notion of projection tables for these segments as well. The procedure for computing the projection table for $P^+_{h,d}$ is similar that the one discussed in previous section (Procedure 1 in Figure 6), and works by extending one edge in each step. However, two aspects need to be addressed. Firstly, in contrast to the prior procedure, the edge being extended may be annotated with a block or un-annoated (and correspond to an original edge found in input query $\mathcal{Q}$). In the former case, we perform a join operation with the edges of the data graph (as before), whereas in the latter case the join operation involves the projection table of the block B. For the sake of uniformity, it will be convenient to view the former edges as blocks as well, denoted B_G, and associate with them a projection table derived from the graph edges, as follows. For each edge $(u, v) \in G$, set $\texttt{cnt}(u, v, \alpha)$ as 1, for $\alpha = \{\chi(u), \chi(v)\}$; all other entries of the table are set to a count of zero. The second aspect is that the nodes of the cycles may also be annotated, and these get included as part of the sequence of joins being performed. The two aspects are addressed by procedures called **NodeJoin** and **EdgeJoin**. The pseudo-code is shown in Figure 7.

The procedure starts with an initial table representing the first edge $(a_h, a_{h\oplus 1})$ and performs a sequence of join operation with the blocks annoatating the nodes and edges of the cycle. At this juncture, two intricacies must be highlited. Firstly, the endpoint a_h and/or a_d may be annotated by a block B, which must be joined by either $P^+_{h,d}$ or $P^+_{h,d}$, but not by both (to avoid double counting). For this purpose, we adopt the convention that $P^+_{h,d}$ and $P^-_{h,d}$ include only the block annotating a_d and a_h (if found), respectively. Secondly, for a block with two boundary nodes p and q, the projection table views one of them as the first boundary node and the other as the second (corresponding to the two components of the keys of the form (u, v, α)). Thus, the boundary nodes are ordered and the projection tables need not be symmetric: taking q as the first boundary node and p as the second boundary node would produce a different boundary tables. However, the boundary tables are transpose of each other ($\texttt{cnt}(u, v, \alpha) = \texttt{cnt}(v, u, \alpha)$). Our algorithm maintains both the tables and uses the appropriate one as dictated by the nodes of the cycle. The pseudo-code reflects the first aspect, but, for the sake of clarity, ignores the second.

The projection counts obtained by the above process are joined using a procedure similar to Figure 6, taking into account the configuration in which the boundary nodes occur. These are aggregated over all possible choices of the high node a_h.

Cycles with a single boundary node are handled in a similar manner by considering each possible choice for the highest node a_h and splitting the cycle into two paths $P^+_{h,d}$ and $P^-_{h,d}$. The setting is simpler with only two configurations possible on how the boundary nodes may appear on the paths: the (single) boundary node may appear in P^+ or P^-. Thus, the prior procedures can be applied here as well.

The case of leaf blocks are also handled via join operations. Any leaf block (a, b) is processed by joining the projection table for the blocks annotating the nodes a, the edge (a, b) and the node b (if found).

At the end of the traversal process, the root block is solved, which is either a cycle or a singleton node. In the former case, the block is treated as a cycle without boundary nodes. Instead of computing its projection table, we simply count the number of colorful matches, via a procedure similar to that of two-boundary cycles. In the latter case, we consider the projection table of the block annotating the singleton node and output the sum of counts across all entries of the table. The process yields the number of colorful matches of the input query $\mathcal{Q}$.

6 Finding Good Decomposition Trees

In each step of the decomposition process, multiple blocks may be available for contraction. Each sequence of choices leads to a unique decomposition tree, and hence, multiple trees are possible for a given query. For example, the query `brain1` (Figure 8) admits two decomposition trees: (i) contract the 4-cycle first and then the 6-cycle, and (ii) vice versa. We conducted an experimental study involving a number of real-world data graphs and queries. For each query, we enumerated all the possible decomposition trees and evaluated the execution time on each graph. We observed a maximum difference of 13x in the execution times of two decomposition trees for the same graph-query combination. However, we noted that in most cases the optimal tree is independent of the data graph and is mainly determined by the structure of the query. These observations show that we need a procedure for selecting a good tree, but in this process, we need not analyze the large data graph; rather, it suffices to focus on the structural properties of the small query graph.

Our study also showed that the following factors, in the decreasing order of importance, determine the execution time: (i) length of the longest cycle block; (ii) number of boundary nodes; (iii) number of node/edge annotations. Armed with the above observations, we designed a simple heuristic procedure. Enumerate all possible trees for the given query and pick the best using the above factors for comparison. In our experimental setting, barring a few exceptions, the heuristic picked the optimal tree in majority of the cases and a near-optimal tree for the rest. Since the queries are of small size (about 10 nodes), even a sequential implementation of the heuristic takes insignificant amount of running time.

7 Distributed Implementation

In this section, we present a brief sketch of the distributed implementation of the two algorithms, highlighting their main aspects. The distributed implementation consists of three layers. The first layer, called the *planner*, finds a good decomposition tree for the given query a fast sequential implementation the heuristic discussed in Section 6. The second layer, called the *plan solver*, takes the data graph and the decomposition tree and implements the PS and DB algorithms presented in Section 5. It accomplishes the above task by using efficient join routines supported by the third layer, called *engine*. The engine has three functionalities. The first is to store the data graph in a distributed manner. This is achieved via a $1D$ decomposition, wherein the vertices are equally distributed among the processors using block distribution, and each vertex is owned by some processor. The second is to maintain projection tables. These tables are of two types: unary projection tables having single-vertex keys of the form (u, α) associated with blocks having single boundary nodes; binary projection tables having two-vertex keys of the form (u, v, α). The binary tables also have variants involving additional fields for storing the mappings for

Table 1: Real Data Graphs

Graph	Domain	Nodes	Edges	Avg Deg	Max Deg
`brightkite`	Geo loc.	58K	214K	4	1135
`condMat`	Collab.	23K	93K	4	281
`astroph`	Collab.	18K	198K	11	504
`enron`	Commn.	36K	180K	5	1385
`hepph`	Citation	34K	421K	12	848
`slashdot`	Soc. net.	82K	900K	11	2554
`epinions`	Soc. net.	131K	841K	6	3558
`orkut`	Soc. net.	524K	1.3M	3	1634
`roadNetCA`	Road net.	2M	2.7M	1.3	14
`brain`	Biology	400K	1.1M	3	286

the boundary vertices. The engine provides a convenient abstraction to the plan solver for all these types of tables. All the tables are maintained as distributed hash tables which use open addressing to resolve collisions. Every entry (u, v, α) is stored on the processor owning v; the degree of v is packed as part of the entry for enforcing the degree constraint in the join operations (of the form $u \succ w$ in Procedure 1 of Figure 6). Signatures are maintained as bitmaps. The third functionality is to support two types of join operations on the projection tables. The first type of join is used for extending a path segment an edge; this involves a join with either the graph edges or the projection table of the block annotating the edge. In the former case, the extension of an entry with a key (u, v, α) with an edge (v, w) will be performed at the owner of v. The result is an entry with a key (u, w, α'); this entry is communicated to the owner of w, where it gets stored. The latter case involves join of two entries with keys (u, v, α_1) and (v, w, α_2). Since the first entry is stored at the owner of v and the second, at the owner of w, a communication is performed to bring the two entries to a common processor. The second type of join is used for merging the projection tables of two path segments (for example, Procedure 2 in Figure 6) and it is implemented in a similar way. The two operations are implemented using a standard sort-merge join procedure with signature compatibility checks performed via fast bitwise operations.

8 Experimental Study

We present an extensive experimental evaluation of the algorithms presented in the paper. Our experiments include a comparison of the algorithms on execution time, strong and weak scaling studies for our algorithm, and studies to evaluate the quality of our query plan generation heuristic and the efficacy of color coding for treewidth two queries.

8.1 Experimental Setup

System. The experiments were conducted on an IBM Blue Gene/Q system [12]. Each BG/Q node has 16 cores and 16 GB memory; multiple nodes are connected using a 5D

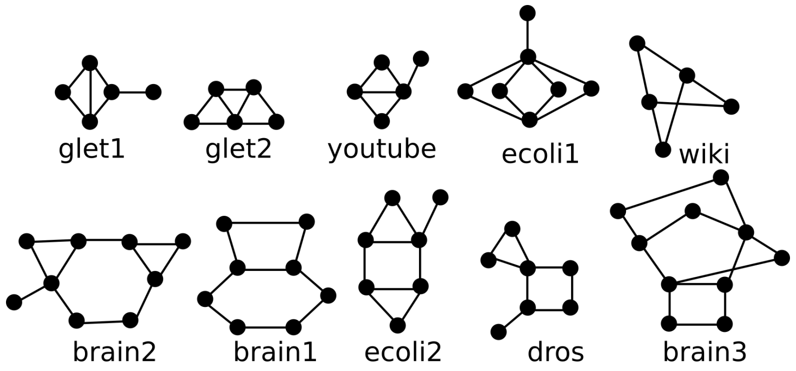


Figure 8: Real world queries used in our study.

Graph	Time(s)		
brightkite	19.1	orkut	3.8
condMat	0.9	roadNetCA	0.9
astroph	23.5	brain	2.3
enron	44.9	slashdot2	25.8
hepph	29.4	epinions	80.2

Query	Time(s)		
dros	2.2	wiki	1.6
ecoli1	15.1	youtube	0.4
ecoli2	2.2	brain1	34.5
glet1	0.4	brain2	68.6
glet2	0.4	brain3	118.5

Figure 9: Average execution time (seconds).

torus interconnect. Our implementation is based on MPI2 with `gcc 4.4.6` with the number of ranks varying from 32 to 512. Each MPI rank was mapped to a single core. The number of MPI ranks mapped to a node was adjusted based on the memory requirements of individual experiments.

Graphs. The experiments involved nine real world graphs obtained from the SNAP dataset collection and the human brain network from the Open Connectome Project (`http://snap.stanford.edu,http://www.openconnectomeproject.org/`). Our benchmark includes representative graphs from different domains in SNAP. The graphs and their characteristics are presented in Table 1. We also used synthetic R-MAT graphs [11], for the purpose of studying the weak scaling behavior of our algorithms.

Queries. Our query benchmark consists of the ten real world queries shown in Figure 8. The queries were derived from prior network analysis work spanning diverse domains: `dros, ecoli1, ecoli2, brain1, brain2, brain3` - biological networks [22, 19]; `glet1, glet2` - graphlets [7]; `wiki` - collaboration networks [32]; `youtube` - spam networks [24].

Algorithms. We study two algorithms: PS, which serves as the baseline, and our degree based DB algorithm. Recall that PS is equivalent to the dyamic programming based algorithm of Alon et al. [2].

8.2 Graph-Query Characteristics

The characteristics of the input graph and query strongly influence the running time of query counting algorithms. To obtain an overall characterization of the phenomenon, we measured the execution time of the DB algorithm on each of the 100 real graph and query combinations using 512 MPI ranks. Figure 9 shows the average running time for each graph across the ten queries and the average running time of each query across the ten graphs. The wide variations in execution time across graphs and queries is indicative of their relative difficulty in practice. For example, although roadNetCA is a larger graph than epinions, the average running time of the former is smaller than the latter by an order of magnitude. We can understand this behaviour by studying the skew in underlying degree distribution. In general, counting colorful occurrences of a query on a graph with high skew (indicated by high maximum degree in Table 1) tends to be computationally expensive. Similarly, the queries also exhibit large variations in running time, ranging from sub-second for youtube, glet1 and glet2 to more than a minute for brain2 and brain3. These variations can be accounted for by studying the differences in the size and the sub-structures of the queries. We observed that queries with longer cycles are more challenging. As an extreme case, a 12-vertex complete binary tree query requires 2 seconds on average, in contrast to the 10-vertex brain3 query which requires nearly 2 minutes on average, exemplifies our observation.

8.3 Performance Comparison of PS and DB Algorithms

We study the performance of the PS and DB algorithms on 100 graph-query combinations obtained by selecting a graph from Table 1 and a query from Figure 8. For our DB algorithm, we used plans supplied by the heuristic described in Section 6. In contrast, for the PS algorithm, we enumerated all the possible plans and obtained the optimal plan. Thus, we compare our algorithm to the best possible scenario for the baseline algorithm.

We compute the improvement factor (IF) of DB over PS as the ratio of the execution time of PS to DB. Figure 10 shows IF at 32 and 512 ranks. The combinations where DB outperforms PS ($IF > 1$) are highlighted in green. The blank entries represent cases where PS (or DB) did not complete execution, due to lack of available memory. At 32 ranks, we can see that DB outperforms PS on 84% of the graph-query combinations with IF being as high as 9.1x (average 2.4x). At 512 ranks, DB outperforms the baseline on 89% of the cases, with IF becoming as high as 28.7x (average 5.0x).

We can see that the relative performance of the two algorithms is dependent on the graph-query pair. For instance, the average IF on `enron` and `condmat` graphs are 8.4 and 3.1 on 512 ranks, respectively, correlating well with their skew in the degree distribution (see Table 1). Similarly, the improvement factors is higher on complex queries such as `brain1` where the average improvement is 13.1x, compared to `youtube` where the average improvement is only 4.1x. The phenomenon becomes extreme in the case of road networks that have very low skew and exhibit sub-second average running time across queries.

Our DB algorithm scales better than PS, as demonstrated by the increase in IF at higher ranks. For different graph-query combinations, we computed the ratio of IF at 512 ranks to that of 32 ranks and found that IF increases by a factor of up to 4.7x (average 1.7x).

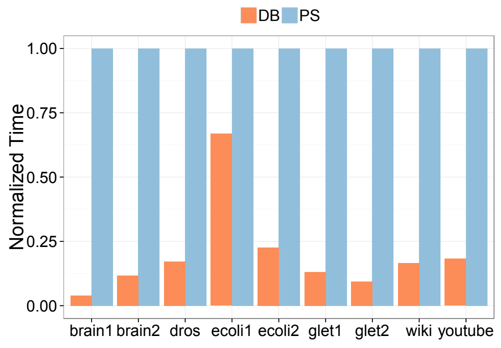
	dros	ecoli1	ecoli2	glet1	glet2	wiki	ytube	brain1	brain2	brain3
brightkite	2.3	0.9	2.1	2.8	4.3	2.8	2.5	-	2.9	-
condMat	1.2	1.0	1.4	1.6	2.1	1.7	1.4	5.2	1.6	2.5
astroph	1.1	0.5	1.1	1.6	2.1	1.5	1.1	-	-	-
enron	3.3	0.9	2.8	4.2	5.7	3.6	3.1	-	-	-
hepph	1.3	0.6	1.4	1.9	2.7	2.0	1.5	-	-	-
orkut	2.9	2.1	3.1	2.7	3.2	2.9	3.5	9.1	8.6	4.1
roadNetCA	0.2	0.4	0.8	0.9	8.2	1.5	4.8	0.4	0.3	0.2
brain	1.7	1.3	1.5	1.3	1.8	1.6	2.0	3.9	3.2	3.1
slashdot2	2.7	0.8	2.9	3.8	5.4	3.4	3.6	-	-	-
epinions	2.6	-	2.1	3.3	4.2	2.4	2.4	-	-	-

(a) 32 Ranks

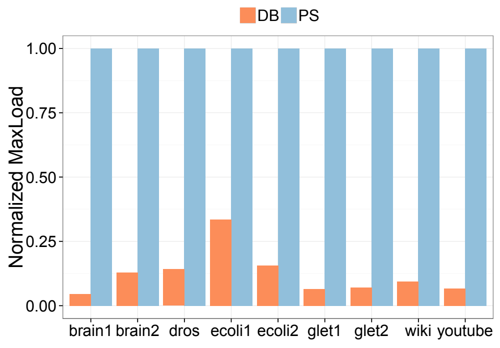
	dros	ecoli1	ecoli2	glet1	glet2	wiki	ytube	brain1	brain2	brain3
brightkite	4.9	1.3	3.3	5.6	10.2	5.5	4.4	28.8	5.4	-
condMat	1.9	1.2	2.1	2.0	2.8	2.1	1.6	9.8	2.3	4.7
astroph	1.5	0.6	1.4	2.1	3.2	1.9	1.4	7.1	2.7	2.4
enron	5.8	1.5	4.4	7.7	10.6	6.0	5.5	25.3	8.5	-
hepph	1.8	0.9	2.0	2.8	4.5	2.6	2.1	6.3	2.3	-
orkut	7.2	5.2	8.6	7.9	9.0	8.0	9.7	21.7	19.2	11.3
roadNetCA	0.7	0.7	0.6	0.7	0.9	0.7	0.8	1.3	0.4	1.0
brain	1.8	1.4	1.6	1.5	1.9	1.8	2.2	4.3	3.5	3.7
slashdot2	6.2	1.6	6.3	9.4	14.5	7.3	7.9	-	-	-
epinions	5.7	1.1	5.0	8.4	9.2	3.7	5.6	-	-	-

(b) 512 Ranks

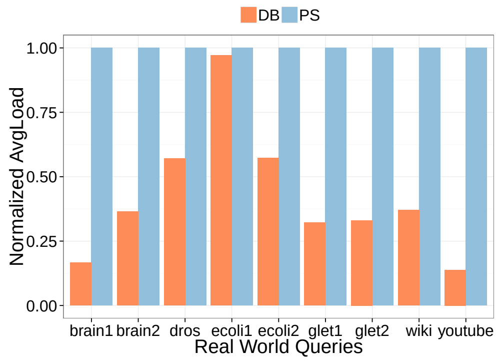
Figure 10: Improvement factor of the DB algorithm over the PS algorithm.



(a) Time



(b) Max. Load



(c) Avg. Load

Figure 11: Normalized execution time, average load and maximum load on `enron` graph.

To understand this trend further, we compute the load (number of projection function operations) for both algorithms for processing different queries on the `enron` graph at 512 ranks. For different queries, Figure 11 shows the execution time and the average and maximum load. We can see that DB has lesser average load than PS, since DB avoids wasteful computations. Furthermore, the improvement obtained by DB over PS on execution time correlates well with improvement obtained on the maximum load. For example, on `ecoli1` query, even though PS outperforms DB at 32 ranks, the performance is reversed at 512 ranks (see Fig 10), because of superior load balancing characteristic of DB.

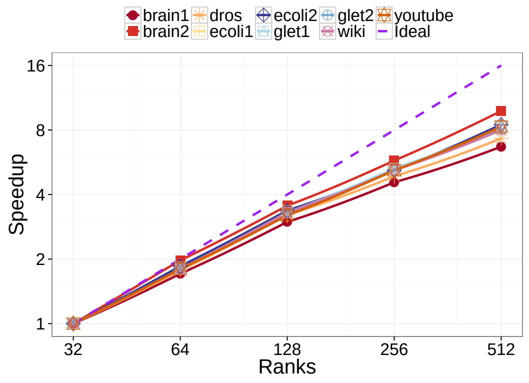
8.4 Scalability Characteristics of DB Algorithm

We studied the scaling of DB across the 100 graph-query combinations. For each combination, we computed the ratio of the execution time at 512 ranks to that of 32 ranks. Figure 12 summarizes the above information by providing the averge of the above speedup for each query across graphs and the same for each graph across queries. As against an

orkut	roadNetCA	brain	slashdot2	epinions	brightkite	condMat	astroph	enron	hepph
10.6	15.8	12.8	9.9	10.2	7.6	7.4	9.8	8.7	10.9

dros	ecoli1	ecoli2	glet1	glet2	wiki	ytube	brain1	brain2	brain3
11	9.7	10.1	10.3	10.2	9.9	10.3	11.7	11.1	11.2

Figure 12: Avg. speedup of DB at 512 ranks compared to 32 ranks.



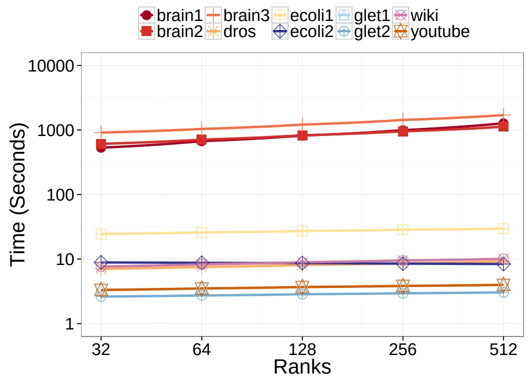


Figure 13: Strong and weak scaling

ideal speedup of 16x, we see that the algorithms obtains speedups in the range of 7.4x to 15.8x.

We studied the strong scaling behavior of our algorithm, using `enron` as a representative graph. Taking 32 ranks as the baseline, Figure 13 shows the speedup up to 512 ranks for different queries. The algorithm scales well across queries, with an average speedup of 8.2x and maximum speedup of 9.9x at 512 ranks (as against an ideal speedup of 16x).

To study weak scaling, we use R-MAT synthetic graphs with parameters $A = 0.5$, $B = 0.1$, $C = 0.1$ and $D = 0.3$ and edge factor 16, suggested in a Graph 500 benchmark specification (`http://www.cc.gatech.edu/~jriedy/tmp/graph500/`). The number of vertices was fixed at $1K$ per rank and the number of ranks was varied from 32 to 512. We report the execution times each query-rank combination in Figure 13. We see excellent weak scaling behavior with the execution times at 512 ranks remaining close to that of the baseline 32 ranks.

8.5 Evaluation of Plan Generation Heuristic

We studied the quality of our plan generation heuristic for the DB algorithm at 512 ranks. For each graph-query combination, we determined the optimal plan via an exhaustive enumeration. We compared the execution time of the heuristic plan to the optimal plan and measured the percentage difference. These results are reported in Figure 14. We can see that in 90% of the case, the heuristic generated the optimal plan, whereas in the remaining cases, the difference was at most 15%.

	dros	ecoli1	ecoli2	glet1	glet2	wiki	ytube	brain1	brain2	brain3
brightkite	0.0	5.3	4.1	0.0	0.0	0.0	10.5	0.0	0.0	0.0
condMat	0.0	0.4	0.0	0.0	0.0	0.0	0.0	2.8	0.0	0.0
astroph	0.0	6.8	0.0	1.2	0.0	0.0	15.0	0.0	0.0	0.0
enron	0.0	7.1	0.5	0.0	0.0	0.0	9.8	0.0	0.0	0.0
hepph	0.0	2.3	0.0	0.0	0.0	0.0	3.0	10.7	0.0	0.0
orkut	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	2.7	0.0
roadNetCA	4.6	0.0	0.0	0.0	0.0	0.0	0.0	12.4	17.8	0.0
brain	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.5	12.1	0.0
slashdot2	0.0	6.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-
epinions	0.0	13.4	10.8	0.5	0.0	0.0	14.7	0.0	0.0	-

Figure 14: Error % of the execution time of the plan proposed by the plan heuristic with reference to the optimal plan for each graph-query combination.

	dros	ecoli1	ecoli2	glet1	glet2	wiki	ytube	brain1	brain2	brain3
brightkite	0.04	0.06	0.05	0.04	0.04	0.03	0.02	0.08	0.09	0.10
condMat	0.06	0.15	0.08	0.03	0.05	0.04	0.02	0.16	0.18	0.26
astroph	0.02	0.04	0.03	0.02	0.02	0.02	0.01	0.04	0.06	0.08
enron	0.03	0.05	0.04	0.02	0.02	0.02	0.02	0.05	0.06	0.08
hepph	0.03	0.09	0.05	0.01	0.02	0.02	0.02	0.05	0.07	0.13
orkut	0.05	0.09	0.08	0.04	0.04	0.04	0.05	0.05	0.10	0.12
roadNetCA	0.01	0.08	0.01	0.00	0.01	0.01	0.01	0.01	0.02	0.05
brain	0.01	0.06	0.02	0.01	0.03	0.01	0.02	0.01	0.01	0.03
slashdot2	0.10	0.04	0.02	0.03	0.09	0.05	0.04	0.05	0.03	-
epinions	0.12	0.02	0.01	0.02	0.02	0.01	0.02	0.02	0.01	-

Figure 15: Coefficient of variation with 50 trials of color coding for each graph-query combination.

8.6 Precision of Color Coding

We evaluated the precision of color coding on our benchmark by performing independent trials and computing the empirical variance of the sample (see Section 2). Specifically, for a given graph-query combination we performed a sequence of trials, where in each trial the colorful count $n^{colorful}(G, \mathcal{Q}, \chi)$ was computed for a fresh random coloring. We performed 10 random trials for each of the 100 graph-query combinations in our test set and evaluated the empirical mean and variance of the number of colorful matches. For each graph-query combination, we computed the coefficient of variation, which is the ratio of the empirical variance to the mean. The results are shown in Figure 15. A value close to 0 indicates the convergence of our estimate to the true mean $n(G, \mathcal{Q})$. We observed that with only three trials, 82% of the graph-query combinations had coefficient of variation at most 0.1; when the number of trials was increased to 10, it increases to 91%. Hence, using 512 ranks, for a majority of the input graph-query combinations in our benchmark, we require less than a minute to count the actual number of matches of the query, with $\approx 10\%$ accuracy. We conclude that our DB algorithm enables fast approximate counting of treewidth 2 queries for data graphs spanning various real domains.

9 Cycle queries on random power law graphs

In this section we concentrate on cycle queries of constant size and analyze the expected runtime of a variant of the PS and DB procedures on a certain class of random data graphs. We prove a lower bound on the expected runtime of the PS procedure and an upper bound on the expected runtime of the DB procedure. Both bounds are functions of the (expected) degree sequence of the graph. We show that our upper bound on the runtime of the DB procedure is never worse (up to constant factors) than the lower bound on the PS procedure. Moreover, if the random graphs satisfy a natural power law property then we prove that the expected runtime of the DB procedure is polynomially better. Recall that the most complicated blocks in our general decomposition of the query graph are (annotated) cycles. Thus, we postulate that the better performance of our variant of the DB procedure on cycle queries explains the better performance of the DB procedure studied in the main body of the paper on queries of treewidth 2.

The class of random graphs considered is a certain variant of the Chung-Lu graphs [14], a popular model for random graphs that captures several properties of real world social networks, as defined precisely below.

9.1 The procedures analyzed

Consider a cycle query $\mathcal{C}$ of constant size k. Since the query graph is a cycle, the decomposition tree consists of only a root node which represents this cycle with a single boundary node.

Recall that procedure PS computes $\texttt{cnt}(u, \{1, \dots, k\}|\mathcal{C})$, for each node u. For this it first computes $\texttt{cnt}(u, v, \alpha_1|P^+)$ and $\texttt{cnt}(u, v, \alpha_2|P^-)$ for all of nodes v and all signatures. Since k is constant the number of signatures is also constant and thus computing $\texttt{cnt}(u, \{1, \dots, k\}|\mathcal{C})$ given $\texttt{cnt}(u, \cdot, \cdot|P^+)$ and $\texttt{cnt}(u, \cdot, \cdot|P^-)$ can be done in linear time. In the PS procedure the computation of $\texttt{cnt}(u, \cdot, \cdot|P^+)$ and $\texttt{cnt}(u, \cdot, \cdot|P^-)$ is done by iteratively recomputing $\texttt{cnt}(u, v, \cdot|P^+)$ for all u and v. This recomputation can be viewed as a optimized version of enumerating all paths starting at u, where instead of storing paths explicitly, we only store the two endpoints, the signature, and a count. Our simplified variant of the PS procedure, which is more amenable to analysis, computes $\texttt{cnt}(u, \cdot, \cdot|P^+)$ and $\texttt{cnt}(u, \cdot, \cdot|P^-)$ by enumerating all possible paths, instead of performing this optimized enumeration. Thus, its complexity is linear in the number of possible paths of lengths $1, \dots, \lceil \frac{1}{2}k \rceil$. We will refer to this version of PS procedure as the PS procedure throughout this section to simplify notation.

To increase the efficiency of our PS procedure when applied to cycle queries we can break symmetry using the id of the nodes and count only colorful matches where node u has the largest id among all data nodes in the image of π. Consequently, $\texttt{cnt}(u, \cdot, \cdot|P^+)$ and $\texttt{cnt}(u, \cdot, \cdot|P^-)$ need only to count paths with the property that node u has the largest id among nodes on the path.

For an integer $q \geq 3$ let

$$Y(q) = |\{(u_1, \dots, u_q) \text{ is a simple path and } \mathrm{id}(u_1) > \mathrm{id}(u_j), j \in [2..q]\}| . \tag{2}$$

It follows that the expected complexity of procedure PS is linear in $\sum_{q=1}^{\lceil k/2 \rceil} \mathbf{E}[Y(q)]$. Later, we derive lower bounds on $\mathbf{E}[Y(q)]$ for any constant q, and our bounds are monotone increasing in q. Thus, the dominant term in the complexity of procedure PS is provided by our bound on $Y(\lceil \frac{1}{2}k \rceil)$.

Procedure DB also computes $\texttt{cnt}(u, \{1, \ldots, k\}|\mathcal{C})$, but does it by computing high-starting colorful matches. Again, to increase the efficiency of procedure DB when applied to cycle queries we can break symmetry and only compute $\texttt{cnt}(u, \{1, \ldots, k\}|\mathcal{C}, \text{hi} = u)$, namely, colorful matches where node u is the highest in the degree based ordering among all data nodes in the image of π. It follows that for this we need only to compute $\texttt{cnt}^*(u, \cdot, \cdot|P^+_{h,d})$ and $\texttt{cnt}^*(u, \cdot, \cdot|P^-_{h,d})$, namely, the number of high-starting colorful matches for the paths $P^+_{h,d}$ and $P^-_{h,d}$, wherein $\pi(a_h) = u$, $\pi(a_d) = v$ and $\texttt{sig}(\pi) = \alpha$, for all possible v and α. Note that similarly to the PS procedure, the recomputation in the DB procedure considered in the main body of the paper can be viewed as a optimized enumeration, where instead of storing paths explicitly, we only store the two endpoints, the signature, and a count. Our simplified variant of the DB procedure, which is more amenable to analysis, computes the counts by enumerating all these paths, instead of performing this optimized enumeration.

For an integer $q \geq 3$ let

$$X(q) = |\{(u_1, \ldots, u_q) \text{ is a simple path and } u_1 \succ u_j, j \in [2..q]\}| \,. \tag{3}$$

It follows that the expected complexity of procedure DB is linear in $\sum_{q=1}^{\lceil k/2 \rceil} \mathbf{E}[X(q)]$. Later, we will derive bounds on $\mathbf{E}[X(q)]$, and since our bounds will be monotone increasing in q, the dominant term in complexity of procedure DB is provided by our bound on $\mathbf{E}[X(\lceil \frac{1}{2}k \rceil)]$.

9.2 The random data graphs

We analyze our algorithm on Chung-Lu graphs whose expected degree sequence has some additional property.

Chung-Lu distribution The Chung-Lu distribution on random graphs is defined as follows. First choose a degree sequence $\mathbf{d} = (d_1, \ldots, d_n)$, where $V = [n]$. We assume that $d_u \geq 1$ for all $u \in V$. Let $m = \frac{1}{2}\sum_{u \in V} d_u$. We assume that $m \geq n$ and $\max_{u \in V} d_u \leq \sqrt{n}$. To generate the graph $G = (V, E)$ we include each edge (u, v) independently with probability $d_u d_v/(2m)$. Note that the expected degree of every node $u \in V$ is $\sum_{v \in [n]} d_u d_v/(2m) = d_u$ as required. We write $\mathsf{deg}(u)$ to denote the actual degree of u in G.

We require the degree sequence $\mathbf{d} = (d_1, \ldots, d_n)$ to be λ-*balanced*.

Balanced degree sequence A degree sequence $\mathbf{d} = (d_1, \ldots, d_n)$ is λ-*balanced* for some $\lambda \in (0, 1)$ if for any integers $a, b \geq 1$, $\sum_u d_u^{a+b} \leq \lambda \cdot (\sum_u d_u^a)(\sum_u d_u^b)$. Intuitively, a degree sequence is balanced if it is not too concentrated on the high degree nodes.

For some of the claims we require the sequence to satisfy the stronger truncated power law.

Truncated power law distribution A degree sequence $\mathbf{d} = (d_1, \ldots, d_n)$ satisfies the truncated power law for a constant $\alpha \in (1, 2)$ if for each $0 \leq j \leq \frac{1}{2}\log n$, the number of nodes with degree between 2^j and 2^{j+1} is $\Theta(n/2^{\alpha j})$. We show later that if a sequence satisfies the power law for α then it is λ-balanced for $\lambda = O(n^{\alpha/2-1})$.

9.3 Main theorem

Theorem 9.1 *Let G be sampled according to the Chung-Lu distribution on n vertices with an $n^{-\delta}$-balanced degree sequence $\mathbf{d}$, for some constant $\delta > 0$, and let $q \geq 3$ be a constant. Then,*

(1) *The expected number of paths $(u_1, \ldots, u_q)$ of length q for which id(u_i) is with the highest id is lower bounded by*

$$\mathbf{E}[Y(q)] \geq (1 - o(1)) \cdot \frac{1}{q}(2m)^{-q+3}\left(\sum_u d_u^2\right)^{q-2}.$$

(2) *The expected number of high-starting paths of length q is upper bounded by*

$$\mathbf{E}[X(q)] \leq C(2m)^{-q+2}\left(\sum_{u \in V} d_u^{2-1/(q-1)}\right)^{q-1}$$

for some constant $C > 1$.

(3) *Based on the above inequalities*

1. $\mathbf{E}[X(q)] = O(\mathbf{E}[Y(q)])$.
2. *If the degree sequence satisfies the truncated power law with parameter α, for any constant $\alpha \in (1, 2)$, then $\mathbf{E}[X(q)]$ is polynomially smaller than $\mathbf{E}[Y(q)]$.*

Proof: **(1)** follows by Lemma 9.5. **(2)** follows by Lemma 9.6. **(3)** follows by putting together Lemma 9.7 and Lemma 9.8. ■

Remark 9.2 *Note that if $\sum_u d_u^2 \geq \sum_u d_u = 2m$ and $\sum_u d_u^{2-1/(q-1)} \geq \sum_u d_u = 2m$, since $d_u \geq 1$ by our assumption on the degree sequence. Thus both $\mathbf{E}[Y(q)]$ and $\mathbf{E}[X(q)]$ are monotone in q. It follows that for any constant k, the dominant terms in the complexity of our* PS *and* DB *procedures for query cycles of length k is indeed determined by $\mathbf{E}[Y(\lceil\frac{1}{2}k\rceil])]$ and $\mathbf{E}[X(\lceil\frac{1}{2}k\rceil])]$, respectively, as claimed in section 9.1.*

The rest of this section is devoted to proving Lemmas 9.5, 9.6, 9.7 and 9.8. The analysis uses the approach of [5]. It turns out, however, that new ingredients are necessary for handling cycles due to the presence of multiple intermediate nodes.

9.4 Useful facts

In this section we state some simple claims and known results that will be useful in the rest of the proof. The following claim specifies the probability that a fixed path exists in a random graph drawn from the Chung-Lu distribution:

Claim 9.1 *Let G be drawn from the Chung-Lu distribution with degree sequence* $\mathbf{d}$. *Then for any $q \geq 2$ and a vector $(u_1, \ldots, u_q) \in V^q$ of distinct nodes*

$$\mathbf{Pr}[(u_1, \ldots, u_q) \textit{ is a path in } G] = \frac{d_{u_1} d_{u_q}}{2m} \cdot \prod_{j=2}^{q-1} \frac{d_{u_j}^2}{2m}.$$

Proof: Since the input graph G is drawn from the Chung-Lu distribution, for each $j = 1, \ldots, q-1$ we have

$$\mathbf{Pr}[(u_j, u_{j+1}) \in E] = \frac{d_{u_j} d_{u_{j+1}}}{2m}.$$

Since edges are included in E independently, we have

$$\mathbf{Pr}[(u_1, \ldots, u_q) \text{ is a path in } G] = \prod_{j=1}^{q-1} \frac{d_{u_j} d_{u_{j+1}}}{2m} = \frac{d_{u_1} d_{u_q}}{2m} \cdot \prod_{j=2}^{q-1} \frac{d_{u_j}^2}{2m}$$

■

We will also use

Theorem 9.3 (Chernoff bound) *Let $X_1, \ldots, X_n$ be independent $0/1$ Bernoulli random variables. Let $X = \sum_{i=1}^n X_i$, and let $\mu := \mathbf{E}[X]$. Then for any $\epsilon \in (0,1)$*

$$\mathbf{Pr}[X \leq (1-\epsilon)\mu] < e^{-\Omega(\epsilon^2 \mu)}$$

and for every $\lambda > 0$

$$\mathbf{Pr}[X \geq (1+\lambda)\mu] < (e/(1+\lambda))^{(1+\lambda)\mu}$$

A simple corollary of the bound is

Corollary 9.4 *Let $X_1, \ldots, X_n$ be independent $0/1$ Bernoulli random variables. Let $X = \sum_{i=1}^n X_i$, and let $\mu_X := \mathbf{E}[X]$. Let $Y_1, \ldots, Y_n$ be independent $0/1$ Bernoulli random variables. Let $Y = \sum_{i=1}^n Y_i$, and let $\mu_Y := \mathbf{E}[Y]$. If $\mu_Y \leq \mu_X/20$, then $\mathbf{Pr}[Y \geq X] < 2e^{-c\mu_X}$, for some constant $c > 0$.*

Proof: We have

$$\mathbf{Pr}[Y \geq X] \leq \mathbf{Pr}[X \leq (\mu_X + \mu_Y)/2] + \mathbf{Pr}[Y \geq (\mu_X + \mu_Y)/2].$$

Since $(\mu_X + \mu_Y)/2 \leq (3/4)\mu_X$, we have $\mathbf{Pr}[X \leq (\mu_X + \mu_Y)/2] < e^{-c\mu_X}$, for some constant $c > 0$ by the first bound from Theorem 9.3 invoked with $\epsilon = 1/4$.

We also have $(\mu_X + \mu_Y)/2 \geq 10\mu_Y$, so by the second bound from Theorem 9.3 invoked with $\lambda = 9$ we get

$$\mathbf{Pr}[Y \geq (\mu_X + \mu_Y)/2] < (e/10)^{10\mu_Y} \leq (e/10)^{\mu_X/2}.$$

Clearly, $(e/10)^{\mu_X/2} = e^{-c\mu_X}$ for some constant $c > 0$, as required. ■

Finally, we need the following inequality on the degree sequence $\mathbf{d}$.

Claim 9.2 *For all $t \geq 1$ one has*

$$\sum_u d_u^{2-1/t} \leq \left(\sum_u d_u\right)^{1/t} \cdot \left(\sum_u d_u^2\right)^{(t-1)/t}.$$

Proof: By Hölder's inequality with conjugates t and $\frac{1}{1-1/t}$ we have

$$\begin{aligned}
\sum_u d_u^{2-1/t} &= \sum_u d_u^{1/t} \cdot d_u^{2-2/t} \\
&\le \left(\sum_u (d_u^{1/t})^t\right)^{1/t} \cdot \left(\sum_u d_u^{(2-2/t)\cdot\frac{1}{1-1/t}}\right)^{1-1/t} \\
&\le \left(\sum_u d_u\right)^{1/t} \cdot \left(\sum_u d_u^2\right)^{(t-1)/t}
\end{aligned}$$

■

9.5 Proofs of the lower and upper bounds

We start by lower bounding $\mathbf{E}[Y(q)]$.

Lemma 9.5 *Let $G = (V, E), V = [n]$ be drawn from the Chung-Lu distribution with degree sequence $\mathbf{d}$. Suppose that node id's are chosen uniformly at random. For any integer $q \ge 3$ let $Y(q)$ be defined by* (2). *Then if $\mathbf{d}$ is $n^{-\delta}$-balanced for a constant $\delta > 0$, the following holds for any constant q*

$$\mathbf{E}[Y(q)] \ge (1-o(1)) \cdot \frac{1}{q}(2m)^{-q+3}\left(\sum_u d_u^2\right)^{q-2}.$$

Proof: We have

$$\mathbf{E}[Y(q)] = \sum_{\substack{(u_1,\ldots,u_q)\in V^q:\\ u_i \text{ distinct}}} \mathbf{Pr}[(u_1,\ldots,u_q) \text{ is a path in } G] \cdot \mathbf{Pr}\left[\text{id}(u_1) > \text{id}(u_j), j \in [2..q]\right] \tag{4}$$

By Claim 9.1 one has for any vector $(u_1,\ldots,u_q)$ of distinct nodes

$$\mathbf{Pr}[(u_1,\ldots,u_q) \text{ is a path in } G] = \prod_{j=1}^{q-1} \frac{d_{u_j} d_{u_{j+1}}}{2m} = \frac{d_{u_1} d_{u_q}}{2m} \cdot \prod_{j=2}^{q-1} \frac{d_{u_j}^2}{2m}.$$

Furthermore, for any fixed q-tuple of distinct nodes $(u_1,\ldots,u_q)$ we have

$$\mathbf{Pr}\left[\text{id}(u_1) > \text{id}(u_j), j \in [2..q]\right] = \frac{1}{q}$$

since id's are uniformly random by assumption of the lemma. (Note that this implies that $Y(q)$ is lower bounded by a constant times the total number of paths of length q, since q is constant.)

Plugging this bound into (4), we get

$$\begin{aligned}
\mathbf{E}[Y(q)] &= \sum_{\substack{(u_1,\ldots,u_q)\in V^q:\\ u_i \text{ distinct}}} \mathbf{Pr}[(u_1,\ldots,u_q) \text{ is a path in } G] \cdot \mathbf{Pr}\left[\mathrm{id}(u_1) > \mathrm{id}(u_j), j \in [2..q]\right] \\
&= \frac{1}{q} \sum_{\substack{(u_1,\ldots,u_q)\in V^q:\\ u_i \text{ distinct}}} \frac{d_{u_1}\cdot d_{u_q}}{2m} \cdot \prod_{j=2}^{q-1} \frac{d_{u_j}^2}{2m} \\
&= \frac{1}{q}\left(\sum_{u_1\in V} d_{u_1}\right)\cdot\left(\sum_{u_q\in V}\frac{d_{u_q}}{2m}\right)\cdot\prod_{j=2}^{q-1}\sum_{u_j\in V}\frac{d_{u_j}^2}{2m} \\
&\quad - \frac{1}{q} \sum_{\substack{(u_1,\ldots,u_q)\in V^q:\\ u_i \textbf{ not} \text{ distinct}}} \frac{d_{u_1}\cdot d_{u_q}}{2m} \cdot \prod_{j=2}^{q-1} \frac{d_{u_j}^2}{2m}.
\end{aligned} \tag{5}$$

Clearly,

$$\frac{1}{q}\left(\sum_{u_1\in V} d_{u_1}\right)\cdot\left(\sum_{u_q\in V}\frac{d_{u_q}}{2m}\right)\cdot\prod_{j=2}^{q-1}\sum_{u_j\in V}\frac{d_{u_j}^2}{2m} = \frac{1}{q}(2m)^{3-q}\cdot\left(\sum_{u\in V} d_u^2\right)^{q-2} \tag{6}$$

We now show that

$$\frac{1}{q} \sum_{\substack{(u_1,\ldots,u_q)\in V^q:\\ u_i \textbf{ not} \text{ distinct}}} \frac{d_{u_1}\cdot d_{u_q}}{2m} \cdot \prod_{j=2}^{q-1} \frac{d_{u_j}^2}{2m} = o\left(\frac{1}{q}(2m)^{3-q}\cdot\left(\sum_{u} d_u^2\right)^{q-2}\right). \tag{7}$$

Together with (5) this gives the result.

We show that (7) follows from the assumption that the degree sequence $\mathbf{d}$ is balanced. Let

$$\begin{aligned}
\mathcal{U} &= \{(u_1,\ldots,u_q)\in V^q : u_i \textbf{ not} \text{ distinct}\} \\
\mathcal{W}_{k,\ell} &= \{(u_1,\ldots,u_q)\in V^q : u_k = u_\ell\}, \text{ for } 1\le k<\ell\le q.
\end{aligned}$$

Note that $\mathcal{U} \subseteq \cup_{1\le k<\ell\le q}\mathcal{W}_{k,\ell}$. Let $\vec{v}$ denote a vector $(u_1,\ldots,u_q)\in V^q$. To simplify the exposition denote

$$S(\vec{v}) = \frac{d_{u_1}\cdot d_{u_q}}{2m}\cdot\prod_{j=2}^{q-1}\frac{d_{u_j}^2}{2m}.$$

We show that for every $1\le k<\ell\le q$,

$$\sum_{\vec{v}\in\mathcal{W}_{k,\ell}} S(\vec{v}) \le \frac{1}{q} n^{-\delta}(2m)^{3-q}\cdot\left(\sum_u d_u^2\right)^{q-2}. \tag{8}$$

Since there are constant number of such sets this implies (7). To prove (8) we consider four different cases.

Case 1: $2 \le k < \ell \le q-1$. In this case

$$\begin{aligned}
\sum_{\vec{v}\in\mathcal{W}_{k,\ell}} S(\vec{v}) &= \frac{1}{q} \sum_{(u_1,\dots,u_q)\in\mathcal{W}_{k,\ell}} \frac{d_{u_1}\cdot d_{u_q}}{2m}\cdot\frac{d_{u_k}^4}{4m^2}\cdot \prod_{j\in[2..q-1]\setminus\{k,\ell\}} \frac{d_{u_j}^2}{2m} \\
&= \frac{1}{q}\left(\sum_{u_1\in V} d_{u_1}\right)\cdot\left(\sum_{u_q\in V}\frac{d_{u_q}}{2m}\right)\cdot\left(\sum_{u_k\in V}\frac{d_{u_k}^4}{4m^2}\right)\cdot \prod_{j\in[2..q-1]\setminus\{k,\ell\}}\sum_{u_j\in V}\frac{d_{u_j}^2}{2m} \\
&\le \frac{1}{q}n^{-\delta}\left(\sum_{u_1\in V} d_{u_1}\right)\cdot\left(\sum_{u_q\in V}\frac{d_{u_q}}{2m}\right)\cdot\left(\sum_{u_k\in V}\frac{d_{u_k}^2}{2m}\right)\cdot\left(\sum_{u_k\in V}\frac{d_{u_k}^2}{2m}\right)\cdot \\
&\quad \prod_{j\in[2..q-1]\setminus\{k,\ell\}}\sum_{u_j\in V}\frac{d_{u_j}^2}{2m} \\
&= \frac{1}{q}n^{-\delta}(2m)^{3-q}\cdot\left(\sum_{u\in V} d_u^2\right)^{q-2}
\end{aligned}$$

Note that the inequality follows since the degree sequence $\mathbf{d}$ is $n^{-\delta}$-bounded.
Case 2: $k=1$ and $\ell=q$. In this case

$$\begin{aligned}
\sum_{\vec{v}\in\mathcal{W}_{1,q}} S(\vec{v}) &= \frac{1}{q}\sum_{(u_1,\dots,u_q)\in\mathcal{W}_{1,q}}\frac{d_{u_1}^2}{2m}\cdot\prod_{j\in[2..q-1]}\frac{d_{u_j}^2}{2m} \\
&= \frac{1}{q}\left(\sum_{u_1\in V}\frac{d_{u_1}^2}{2m}\right)\cdot\prod_{j\in[2..q-1]}\sum_{u_j\in V}\frac{d_{u_j}^2}{2m} \\
&\le \frac{1}{q}n^{-\delta}\left(\sum_{u_1\in V} d_{u_1}\right)\cdot\left(\sum_{u_1\in V}\frac{d_{u_1}}{2m}\right)\cdot\prod_{j\in[2..q-1]}\sum_{u_j\in V}\frac{d_{u_j}^2}{2m} \\
&= \frac{1}{q}n^{-\delta}(2m)^{3-q}\cdot\left(\sum_{u\in V} d_u^2\right)^{q-2}
\end{aligned}$$

Case 3: $k=1$ and $\ell\in[2..q-1]$. In this case

$$\begin{aligned}
\sum_{\vec{v}\in\mathcal{W}_{1,\ell}} S(\vec{v}) &= \frac{1}{q}\sum_{(u_1,\dots,u_q)\in\mathcal{W}_{1,\ell}}\frac{d_{u_1}^3}{2m}\cdot\frac{d_{u_q}}{2m}\cdot\prod_{j\in[2..q-1]\setminus\{\ell\}}\frac{d_{u_j}^2}{2m} \\
&= \frac{1}{q}\left(\sum_{u_1\in V}\frac{d_{u_1}^3}{2m}\right)\cdot\left(\sum_{u_q\in V}\frac{d_{u_q}}{2m}\right)\cdot\prod_{j\in[2..q-1]\setminus\{\ell\}}\sum_{u_j\in V}\frac{d_{u_j}^2}{2m} \\
&\le \frac{1}{q}n^{-\delta}\left(\sum_{u_1\in V} d_{u_1}\right)\cdot\left(\sum_{u_q\in V}\frac{d_{u_q}}{2m}\right)\cdot\left(\sum_{u_1\in V}\frac{d_{u_1}^2}{2m}\right)\cdot\prod_{j\in[2..q-1]\setminus\{\ell\}}\sum_{u_j\in V}\frac{d_{u_j}^2}{2m} \\
&= \frac{1}{q}n^{-\delta}(2m)^{3-q}\cdot\left(\sum_{u\in V} d_u^2\right)^{q-2}
\end{aligned}$$

Case 4: $k \in [2..q-1]$ and $\ell = q$. This case is symmetric to Case 3. ∎

The following lemma provides an upper bound on the expected runtime of the degree-based algorithm for enumerating cycles of length $q \geq 3$:

Lemma 9.6 *Let $G = (V, E), V = [n]$ be drawn from the Chung-Lu distribution with degree sequence* $\mathbf{d}$. *For any integer $q \geq 3$ let $X(q)$ be defined by* (3). *Then there exists an absolute constant $C > 1$ such that*

$$\mathbf{E}[X(q)] \leq C(2m)^{-q+2}\left(\sum_{u\in V} d_u^{2-1/(q-1)}\right)^{q-1}.$$

Proof: We have by (3)

$$\mathbf{E}[X(q)] \leq \sum_{\substack{(u_1,\dots,u_q)\in V^q:\\ u_i \text{distinct}}} \mathbf{Pr}[(u_1,\dots,u_q) \text{ is a path in } G \text{ and } \forall j \in [2..q]\ \mathsf{deg}(u_1) \geq \mathsf{deg}(u_j)]$$

where $\mathsf{deg}(u)$ stands for the *actual* degree of the node $u \in V$ in the graph G. Note that while $\mathbf{E}[\mathsf{deg}(u)] = d_u$, there may be deviations due to the sampling process. This fact introduces some complications in the analysis.

Similarly to [5], we start by splitting the summation above into two. For a constant ϕ (to be fixed later to $1/80$) let

$$\mathbf{E}[X(q)] = S_1 + S_2, \tag{9}$$

where

$$S_1 := \sum_{\substack{(u_1,\dots,u_q)\in V^q: u_i \text{ distinct},\\ \forall j\in[2..q]\ d_{u_1} > \phi d_{u_j}}} \mathbf{Pr}[(u_1,\dots,u_q) \text{ is a path in } G \text{ and } \forall j \in [2..q]\ \mathsf{deg}(u_1) \geq \mathsf{deg}(u_j)]$$

$$S_2 := \sum_{\substack{(u_1,\dots,u_q)\in V^q: u_i \text{ distinct}\\ \exists j\in[2..q] \text{ s.t. } d_{u_1} \leq \phi d_{u_j}}} \mathbf{Pr}[(u_1,\dots,u_q) \text{ is a path in } G \text{ and } \forall j \in [2..q]\ \mathsf{deg}(u_1) \geq \mathsf{deg}(u_j)].$$

We now bound the two summations separately.

Bounding S_1 To bound S_1 we use the fact that for any q-tuple of distinct $u_1,\dots,u_q$ by Claim 9.1

$$\begin{aligned}&\mathbf{Pr}[(u_1,\dots,u_q) \text{ is a path in } G \text{ and } \forall j \in [2..q]\ \mathsf{deg}(u_1) \geq \mathsf{deg}(u_j)\\ &\leq \mathbf{Pr}[(u_1,\dots,u_q) \text{ is a path in } G] = \frac{d_{u_1}\cdot d_{u_q}}{2m}\cdot\prod_{j=2}^{q-1}\frac{d_{u_j}^2}{2m}.\end{aligned}$$

We thus get

$$S_1 \leq \sum_{\substack{(u_1,\dots,u_q)\in V^q\\ \forall j\in[2..q]\ d_{u_1} > \phi d_{u_j}}} \frac{d_{u_1}\cdot d_{u_q}}{2m}\cdot\prod_{j=2}^{q-1}\frac{d_{u_j}^2}{2m} \tag{10}$$

Since $\phi d_{u_j} < d_{u_1}$ for all potential paths $(u_1, \dots, u_q)$ in the summation above, we have

$$d_{u_1}^{\left(1-\frac{1}{q-1}\right)} = d_{u_1}^{\frac{q-2}{q-1}} > \prod_{j=2}^{q-1} (\phi d_{u_j})^{\frac{1}{q-1}} = \phi^{\frac{q-2}{q-1}} \prod_{j=2}^{q-1} d_{u_j}^{\frac{1}{q-1}} \geq \phi \prod_{j=2}^{q-1} d_{u_j}^{\frac{1}{q-1}}.$$

It follows that

$$d_{u_1} \cdot \prod_{j=2}^{q-1} d_{u_j}^2 = d_{u_1} \cdot \prod_{j=2}^{q-1} d_{u_j}^{\left(2-\frac{1}{q-1}\right)} \cdot \prod_{j=2}^{q-1} d_{u_j}^{\frac{1}{q-1}} \leq \frac{1}{\phi} \prod_{j=1}^{q-1} d_{u_j}^{\left(2-\frac{1}{q-1}\right)}.$$

Substituting this bound in (10), we get

$$\begin{aligned}
S_1 &\leq \sum_{\substack{(u_1,\dots,u_q)\in V^q \\ \forall j\in[2..q]\ d_{u_1} > \phi d_{u_j}}} \frac{d_{u_1} \cdot d_{u_q}}{2m} \cdot \prod_{j=2}^{q-1} \frac{d_{u_j}^2}{2m} \qquad \leq \frac{1}{\phi} \sum_{\substack{(u_1,\dots,u_q)\in V^q \\ \forall j\in[2..q]\ d_{u_1} > \phi d_{u_j}}} d_{u_q} \cdot \prod_{j=1}^{q-1} \frac{d_{u_j}^{2-1/(q-1)}}{2m} \\
&\leq \frac{1}{\phi} \sum_{(u_1,\dots,u_q)\in V^q} d_{u_q} \cdot \prod_{j=1}^{q-1} \frac{d_{u_j}^{2-1/(q-1)}}{2m} \qquad \leq \frac{1}{\phi} \left(\sum_{u_q \in V} d_{u_q} \right) \cdot \prod_{j=1}^{q-1} \sum_{u_j \in V} \frac{d_{u_j}^{2-1/(q-1)}}{2m} \\
&\leq \frac{1}{\phi} (2m)^{-q+2} \left(\sum_u d_u^{\left(2-\frac{1}{q-1}\right)} \right)^{q-1}.
\end{aligned}$$

Bounding S_2 Recall that

$$\begin{aligned}
S_2 &= \sum_{\substack{(u_1,\dots,u_q)\in V^q: \\ \exists j\in[2..q] \text{ s.t. } d_{u_1} \leq \phi d_{u_j}}} \mathbf{Pr}\left[(u_1, \dots, u_q) \text{ is a path in } G \text{ and } \forall j \in [2..q]\ \mathsf{deg}(u_1) \geq \mathsf{deg}(u_j)\right] \\
&= \sum_{\substack{(u_1,\dots,u_q)\in V^q: \\ \exists j\in[2..q] \text{ s.t. } d_{u_1} \leq \phi d_{u_j}}} \mathbf{Pr}[\forall j \in [2..q]\ \mathsf{deg}(u_1) \geq \mathsf{deg}(u_j) \mid \mathcal{E}(u_1, \dots, u_q)] \cdot \mathbf{Pr}[\mathcal{E}(u_1, \dots, u_q)],
\end{aligned}$$

where we let $\mathcal{E}(u_1, \dots, u_q) := \{(u_1, \dots, u_q) \text{ is a path in } G\}$. We omit the argument of $\mathcal{E}$ below to simplify notation. Note that we have

$$\begin{aligned}
d_{u_1} \leq& \mathbf{E}[\mathsf{deg}(u_1) \mid \mathcal{E}] \leq d_{u_1} + 1 \\
d_{u_j} \leq& \mathbf{E}[\mathsf{deg}(u_j) \mid \mathcal{E}] \leq d_{u_j} + 2 \text{ for all } j \in [2..q-1] \\
d_{u_q} \leq& \mathbf{E}[\mathsf{deg}(u_q) \mid \mathcal{E}] \leq d_{u_q} + 1.
\end{aligned}$$

Furthermore, $\mathsf{deg}(u_j)$ is a sum of independent $0/1$ Bernoulli random variables even conditional on the event $\mathcal{E}$. Now we would like to apply Corollary 9.4 to random variables $\mathsf{deg}(u_1)$ and $\mathsf{deg}(u_j)$ conditional on $\mathcal{E}$, but there is one more issue: these random variables are dependent through the potential edge (u_1, u_j). To avoid this issue, we omit the $0/1$ Bernoulli random variable corresponding to this potential edge from both random variables $\mathsf{deg}(u_1)$ and $\mathsf{deg}(u_j)$. Let $\overline{\mathsf{deg}}(u_1)$ and $\overline{\mathsf{deg}}(u_j)$ be the modified random variables. Namely, $\overline{\mathsf{deg}}(u_1) := \mathsf{deg}(u_1) - \mathbf{1}_{(u_1,u_j)\in E}$ and $\overline{\mathsf{deg}}(u_j) := \mathsf{deg}(u_q) - \mathbf{1}_{(u_1,u_j)\in E}$, where $\mathbf{1}_{(u_1,u_j)\in E}$ is the random variable corresponding to the sampling of the potential edge (u_1, u_j).

Note that conditional on $\mathcal{E}$ the random variables $\overline{\mathsf{deg}}(u_1)$ and $\overline{\mathsf{deg}}(u_j)$ are independent, and are both sums of independent Bernoulli $0/1$ random variables. Note that $d_u \geq 1$ for every node $u \in V$, and since $d_{u_1} \leq \phi d_{u_j}$, then $d_{u_j} \geq 1/\phi$. It follows that

$$\mathbf{E}[\overline{\mathsf{deg}}(u_1) \mid \mathcal{E}] = \mathbf{E}[\mathsf{deg}(u_1) - \mathbf{1}_{(u_1,u_j)\in E} \mid \mathcal{E}] \leq d_{u_1} + 1 \leq \phi(d_{u_j} + \frac{1}{\phi}) \leq 2\phi d_{u_j} \tag{11}$$

On the other hand since $d_{u_1} < m$,

$$\mathbf{E}[\overline{\mathsf{deg}}(u_j) \mid \mathcal{E}] = \mathbf{E}[\mathsf{deg}(u_j) - \mathbf{1}_{(u_1,u_j)\in E} \mid \mathcal{E}] \geq d_{u_j}\left(1 - \frac{d_{u_1}}{2m}\right) \geq \frac{1}{2} d_{u_j}. \tag{12}$$

Putting these bounds together with the assumption that $\phi < 1/80$, we get that the preconditions of Corollary 9.4 are satisfied, and hence

$$\begin{aligned}\mathbf{Pr}[\mathsf{deg}(u_1) \geq \mathsf{deg}(u_j) \mid \mathcal{E}] &= \mathbf{Pr}[\mathsf{deg}(u_1) - \mathbf{1}_{(u_1,u_j)\in E} \geq \mathsf{deg}(u_j) - \mathbf{1}_{(u_1,u_j)\in E} \mid \mathcal{E}] \\ &= \mathbf{Pr}[\overline{\mathsf{deg}}(u_1) \geq \overline{\mathsf{deg}}(u_j) \mid \mathcal{E}] \leq 2e^{-\Omega(d_{u_j})}.\end{aligned} \tag{13}$$

This allows us to bound $\mathbf{Pr}[\forall j \in [2..q]\ \mathsf{deg}(u_1) \geq \mathsf{deg}(u_j) \mid \mathcal{E}]$ as follows. Let j^* be the index of the highest expected degree node in $\{u_2, \ldots, u_q\}$, that is, $j^* := \text{argmax}_{j\in[2..q]} d_{u_j}$. Clearly,

$$\mathbf{Pr}[\forall j \in [2..q]\ \mathsf{deg}(u_1) \geq \mathsf{deg}(u_j) \mid \mathcal{E}] \leq \mathbf{Pr}[\mathsf{deg}(u_1) \geq \mathsf{deg}(u_{j^*}) \mid \mathcal{E}].$$

Note that if there exists $j \in [2..q]$ such that $d_{u_1} \leq \phi d_{u_j}$, then also $d_{u_1} \leq \phi d_{u_{j^*}}$. Thus, applying (13) with $j = j^*$ we get

$$\begin{aligned}\mathbf{Pr}[\forall j \in [2..q]\ \mathsf{deg}(u_1) \geq \mathsf{deg}(u_j) \mid \mathcal{E}] &\leq \mathbf{Pr}[\overline{\mathsf{deg}}(u_1) \geq \overline{\mathsf{deg}}(u_{j^*}) \mid \mathcal{E}] \\ &\leq 2e^{-\Omega(d_{u_{j^*}})}.\end{aligned}$$

Substituting this bound in the expression for S_2 and using the fact that $2e^{-\Omega(d_{u_{j^*}})} \leq 2\prod_{j\in[1..q]} e^{-\Omega\left(\frac{1}{q}d_{u_j}\right)}$ (since $d_{u_{j^*}} \geq d_{u_j}$ for all $j \in [1..q]$), we get

$$\begin{aligned}S_2 &\leq \sum_{\substack{(u_1,\ldots,u_q)\in V^q: \\ \exists j\in[2..q] \text{ s.t. } d_{u_1} \leq \phi d_{u_j}}} (2m)^{-q+1} \cdot d_{u_1} e^{-\Omega(\frac{1}{q}d_{u_1})} \cdot d_{u_q} e^{-\Omega(\frac{1}{q}d_{u_q})} \cdot \prod_{j\in[2..q-1]} d_{u_j}^2 e^{-\Omega(\frac{1}{q}d_{u_j})} \\ &\leq (2m)^{-q+1} \cdot \left(\sum_{u\in V} d_u e^{-\Omega(\frac{1}{q}d_u)}\right)^2 \cdot \left(\sum_{u\in V} d_u^2 e^{-\Omega(\frac{1}{q}d_u)}\right)^{q-2}.\end{aligned}$$

Recall that the first two moments of the exponential distribution are constants and thus since q is assumed to be constant we get that both

$$\sum_{u\in V} d_u e^{-\Omega(\frac{1}{q}d_u)} \text{ and } \sum_{u\in V} d_u^2 e^{-\Omega(\frac{1}{q}d_u)}$$

are constants and thus $S_2 = O\left((2m)^{-q+1}\right) = o(S_1)$. ∎

9.6 Comparing the lower and upper bounds

We show that the bound on $\mathbf{E}[X(q)]$ is not much worse than our bound on $\mathbf{E}[Y(q)]$ *for any degree sequence* $\mathbf{d}$ that is $n^{-\delta}$-balanced. We later show that our bound gives polynomially smaller runtime if the degree sequence $\mathbf{d}$ satisfies the truncated power law.

Lemma 9.7 *For any $q \geq 3$ and any $n^{-\delta}$-balanced degree sequence $\mathbf{d}$, for a constant $\delta > 0$, $\mathbf{E}[X(q)] = O(\mathbf{E}[Y(q)])$.*

Proof: We start with the bound from Lemma 9.6:

$$\mathbf{E}[X(q)] \leq C(2m)^{2-q} \left(\sum_{u \in V} d_u^{\left(2-\frac{1}{q-1}\right)} \right)^{q-1}.$$

By Claim 9.2 with $t = q-1$ we have

$$\sum_u d_u^{\left(2-\frac{1}{q-1}\right)} \leq \left(\sum_u d_u \right)^{\frac{1}{q-1}} \cdot \left(\sum_u d_u^2 \right)^{\frac{q-1-1}{q-1}}.$$

Substituting this into the bound above, we get that

$$\begin{aligned}
\mathbf{E}[X(q)] &\leq C(2m)^{2-q} \left(\sum_{u \in V} d_u^{\left(2-\frac{1}{q-1}\right)} \right)^{q-1} \\
&\leq C(2m)^{2-q} \left(\left(\sum_u d_u \right)^{\frac{1}{q-1}} \cdot \left(\sum_u d_u^2 \right)^{\frac{q-2}{q-1}} \right)^{q-1} \\
&= C(2m)^{2-q} \left(\sum_u d_u \right) \cdot \left(\sum_u d_u^2 \right)^{q-2} = C(2m)^{3-q} \left(\sum_u d_u^2 \right)^{q-2}.
\end{aligned}$$

The lemma follows by comparing the above bound to the bound on $\mathbf{E}[Y(q)]$ provided by Lemma 9.5 (note that the preconditions are satisfied, as $\mathbf{d}$ is $n^{-\delta}$-balanced for a constant $\delta > 0$ by assumption of the lemma). ■

We now compare the bounds from Lemma 9.6 and Lemma 9.5 when the graph G is drawn from the Chung-Lu distribution with a degree sequence that satisfies the truncated power law.

Lemma 9.8 *Let G be a random graph drawn from the Chung-Lu distribution with degree sequence $\mathbf{d}$ that satisfies the truncated power law with exponent α, for a constant $\alpha \in (1, 2)$. The following bounds hold for any constant $q \geq 3$.*

(1) $\mathbf{E}[Y(q)] = \Omega\left(n^{\alpha-1+\frac{1}{2}(2-\alpha)q}\right)$

(2a) $\mathbf{E}[X(q)] = O\left(n^{\frac{1}{2}+\frac{1}{2}(2-\alpha)(q-1)}\right)$, *for* $\alpha \in (1, 2-\frac{1}{q-1})$.

(2b) $\mathbf{E}[X(q)] = O(n \log n)$, *for* $\alpha \in [2-\frac{1}{q-1}, 2)$.

Proof: We first prove the bound on $\mathbf{E}[Y(q)]$. As shown later in Claim 10.1 the condition that the degree sequence $\mathbf{d}$ satisfies the truncated power law implies that the degree sequence $\mathbf{d}$ is $n^{-\delta}$-balanced for a constant $\delta > 0$, and hence preconditions of Lemma 9.5 are satisfied. To apply the bound of Lemma 9.5 we bound $\sum_u d_u^2$.

$$\sum_u d_u^2 = \sum_{j=0}^{\frac{1}{2}\log_2 n} \left(\frac{n}{2^{\alpha j}}\right) 2^{2j} = n \cdot \sum_{j=0}^{\frac{1}{2}\log_2 n} 2^{(2-\alpha)j} = \Theta\left(n \cdot n^{1-\frac{1}{2}\alpha}\right).$$

The number of edges in a graph with degree sequence that satisfies the power law with exponent $\alpha \in (1,2)$ satisfies

$$m = \frac{1}{2}\sum_{u \in V} d_u = \Theta\left(\sum_{j \geq 0} 2^j \cdot \frac{n}{2^{\alpha j}}\right) = \Theta\left(\sum_{j \geq 0} 2^{(1-\alpha)j} \cdot n\right) = \Theta(n).$$

Plugging both bounds in the bound of Lemma 9.5 we get

$$\mathbf{E}[Y(q)] = \Omega\left(n^{-q+3} n^{(2-\frac{1}{2}\alpha)(q-2)}\right) = \Omega\left(n^{\alpha-1+\frac{1}{2}(2-\alpha)q}\right).$$

To bound $\mathbf{E}[X(q)]$ using Lemma 9.6 we need first to bound the following summation.

$$\sum_{u \in V} d_u^{\left(2-\frac{1}{q-1}\right)} = \sum_{j=0}^{\frac{1}{2}\log_2 n} \frac{n}{2^{\alpha j}} \cdot 2^{\left(2-\frac{1}{q-1}\right)j} = n \cdot \sum_{j=0}^{\frac{1}{2}\log_2 n} 2^{\left(2-\alpha-\frac{1}{q-1}\right)j}. \tag{14}$$

The sum (14) above is dominated either by the first term or by the last term, depending on whether α is less than or greater than $2 - \frac{1}{q-1}$.

Case 1: $\alpha \in \left(1, 2 - \frac{1}{q-1}\right)$. In this case the sum (14) is dominated by the last term and thus bounded by

$$n \cdot \sum_{j=0}^{\frac{1}{2}\log_2 n} 2^{\left(2-\alpha-\frac{1}{q-1}\right)j} = O\left(n^{1+\frac{1}{2}\left(2-\alpha-\frac{1}{q-1}\right)}\right) = O\left(n^{2-\frac{1}{2}\left(\alpha+\frac{1}{q-1}\right)}\right).$$

Substituting this into the bound provided by Lemma 9.6, using the fact that $m = \Theta(n)$ we get

$$\mathbf{E}[X(q)] = O\left(n^{-q+2} n^{\left(2-\frac{1}{2}\left(\alpha+\frac{1}{q-1}\right)(q-1)\right)}\right) = O\left(n^{\frac{1}{2}+\frac{1}{2}(2-\alpha)(q-1)}\right)$$

Case 2: $\alpha \in \left[2 - \frac{1}{q-1}, 2\right)$. In this case the sum (14) is dominated by the first term which is constant and since we have $\frac{1}{2}\log n$ summands the sum (14) is $O(n \log n)$.

Substituting this into the bound provided by Lemma 9.6, using the fact that $m = \Theta(n)$ we get

$$\mathbf{E}[X(q)] = O\left(n^{-q+2} (n \log n)^{q-1}\right) = O\left(n (\log n)^{q-1}\right).$$

■

Applying these bounds we conclude the following

Corollary 9.9 *If G is a random graph drawn from the Chung-Lu distribution with degree sequence* $\mathbf{d}$ *that satisfies the truncated power law with exponent α, for a constant $\alpha \in (1,2)$, then $\mathbf{E}[X(q)] = o\left(\mathbf{E}[Y(q)]\right)$.*

Proof: Using the bounds of Lemma 9.8 above. If $\alpha \in \left(1, 2-\frac{1}{q-1}\right)$ the improvement of the degree based algorithm over the id based (naive) algorithm is

$$\frac{\mathbf{E}[Y(q)]}{\mathbf{E}[X(q)]} \geq \frac{n^{\alpha-1+\frac{1}{2}(2-\alpha)q}}{n^{\frac{1}{2}+\frac{1}{2}(2-\alpha)(q-1)}} = n^{\frac{1}{2}(\alpha-1)}$$

If $\alpha \in \left[2-\frac{1}{q-1}, 1\right)$ the improvement of the degree based algorithm over the id based (naive) algorithm is

$$\frac{\mathbf{E}[Y(q)]}{\mathbf{E}[X(q)]} \geq \frac{n^{\alpha-1+\frac{1}{2}(2-\alpha)q}}{n\left(\log n\right)^{q-1}} = n^{\alpha-2+\frac{1}{2}(2-\alpha)q} \cdot \left(\log n\right)^{1-q}$$

■

Note that if If $\alpha \in \left(2-\frac{1}{q-1}, 1\right)$ then the second term vanishes.

10 Power law and balanced sequences

In this section we show that a degree sequence that satisfies the truncated power law is also balanced.

Claim 10.1 *Any degree sequence* $\mathbf{d}$ *that satisfies the truncated power law with exponent α, for any $\alpha \in (1,2)$ that is bounded away from 1 and 2 by constants, is λ-balanced for $\lambda = O\left(n^{\frac{1}{2}\alpha-1}\right)$.*

Proof: The intuition behind the proof is simple: the bound that needs to be satisfied by a balanced sequence holds because a degree sequence that satisfies the truncated power law contains about $n^{\frac{1}{2}(1-\alpha)}$ nodes of degree $\sqrt{n}$ (the largest), which means that the edge mass is somewhat spread among these nodes, leading to the result of the lemma. We now give the details.

Recall that by definition of the truncated power law, for each $j = 0, \ldots, \frac{1}{2}\log_2 n$, the number of nodes with degree $\Theta(2^j)$ is $\Theta(n/2^{\alpha j})$. For any integer $s \geq 2$, we have

$$\sum_u d_u^s = \Theta\left(\sum_{j=0}^{\frac{1}{2}\log_2 n} 2^{s\cdot j} \cdot \frac{n}{2^{\alpha j}}\right) = \Theta\left(\sum_{j=0}^{\frac{1}{2}\log_2 n} 2^{(s-\alpha)\cdot j} \cdot n\right).$$

Since $s \geq 2$ and α is bounded away from 2 by assumption, the summation is dominated by the last term. Namely,

$$\sum_u d_u^s = \Theta\left(\sum_{j=0}^{\frac{1}{2}\log_2 n} 2^{(s-\alpha)\cdot j} \cdot n\right) = \Theta\left(n^{s/2} \cdot n^{1-\alpha/2}\right)$$

We now show that for for any integers $a, b \geq 1$, $\sum_u d_u^{a+b} \leq \lambda \cdot (\sum_u d_u^a)(\sum_u d_u^b)$.

We distinguish the following three cases:

Case 1: $a \geq 2$ **and** $b \geq 2$**.** In this case the derivation above with $q = a$ and $q = b$ gives

$$\sum_u d_u^a = \Theta(n^{a/2} \cdot n^{1-\alpha/2}) \quad \sum_u d_u^b = \Theta(n^{b/2} \cdot n^{1-\alpha/2}).$$

It follows that

$$\begin{aligned}\frac{\sum_u d_u^{a+b}}{(\sum_u d_u^a)(\sum_u d_u^b)} &= \Theta\left(\frac{n^{(a+b)/2} \cdot n^{1-\alpha/2}}{n^{a/2} \cdot n^{1-\alpha/2} \cdot n^{b/2} \cdot n^{1-\alpha/2}}\right) \\ &= \Theta\left(n^{\alpha/2-1}\right),\end{aligned}$$

which gives the result.
Case 2: $a = 1$**,** $b \geq 2$**.** Since α is bounded away from 1 by a constant we have

$$\sum_u d_u = \sum_{j=0}^{\frac{1}{2}\log_2 n} 2^j \cdot \frac{n}{2^{\alpha j}} = \Theta(n) \tag{15}$$

On the other hand, since $b \geq 2$ and $a + b \geq 2$,

$$\sum_u d_u^{a+b} = \Theta(n^{(a+b)/2} \cdot n^{1-\alpha/2}) \quad \sum_u d_u^b = \Theta(n^{b/2} \cdot n^{1-\alpha/2})$$

Putting the estimates above together, we get

$$\frac{\sum_u d_u^{a+b}}{(\sum_u d_u^a)(\sum_u d_u^b)} = \Theta\left(\frac{n^{(a+b)/2} \cdot n^{1-\alpha/2}}{n \cdot n^{b/2} \cdot n^{1-\alpha/2}}\right) = \Theta\left(n^{-1/2}\right).$$

The result follows since $n^{\alpha/2-1} \geq n^{-1/2}$ by the assumption that $\alpha \in (1, 2)$. Note that the case $a \geq 2$, $b = 1$ is symmetric.
Case 3: $a = 1$**,** $b = 1$**.** Then we have

$$\sum_u d_u^{a+b} = \Theta(n^{(a+b)/2} \cdot n^{1-\alpha/2}) \quad \sum_u d_u^a = \Theta(n) \quad \sum_u d_u^b = \Theta(n)$$

by the estimates above, and hence

$$\begin{aligned}\frac{\sum_u d_u^{a+b}}{(\sum_u d_u^a)(\sum_u d_u^b)} &= \Theta\left(\frac{n^{(a+b)/2} \cdot n^{1-\alpha/2}}{n \cdot n}\right) \\ &= \Theta\left(n^{-\alpha/2}\right)\end{aligned}$$

The result follows since $n^{\alpha/2-1} \geq n^{-\alpha/2}$ by the assumption that $\alpha \in (1, 2)$. ■

References

[1] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. Sahinalp. Biomolecular network motif counting and discovery by color coding. In *ISMB*, 2008.

[2] N. Alon, R. Yuster, and U. Zwick. Color-coding. *JACM*, 42(4):844–856, 1995.

[3] D. Aparicio, P. Ribeiro, and F. da Silva. Parallel subgraph counting for multicore architectures. In *ISPA*, 2014.

[4] K. Baskerville and M. Paczuski. Subgraph ensembles and motif discovery using an alternative heuristic for graph isomorphism. *Phys. Rev. E*, 74:051903, 2006.

[5] J. Berry, L. Fostvedt, D. Nordman, C. Phillips, C. Seshadhri, and A. Wilson. Why do simple algorithms for triangle enumeration work in the real world? In *ITCS*, 2014.

[6] M. Bhuiyan and M. Al Hassan. An iterative MapReduce based frequent subgraph mining algorithm. *IEEE Trans. Knowledge Data Eng.*, 27(3):608–620, 2015.

[7] M. Bhuiyan, M. Rahman, M. Rahman, and M. Al Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *ICDM*, 2012.

[8] E. Bloedorn, N. Rothleder, D. DeBarr, and L. Rosen. Relational Graph Analysis with Real-World Constraints: An Application in IRS Tax Fraud Detection. In *AAAI*, 2005.

[9] H. Bodlaender. Treewidth of graphs. *Encyclopedia of Algorithms*, 2015.

[10] S. Burt. Structural Holes and Good Ideas. *The American Journal of Sociology*, 110(2):349–399, 2004.

[11] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A Recursive Model for Graph Mining. In *SDM*, 2004.

[12] D. Chen, N. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, and J. Parker. The IBM Blue Gene/Q Interconnection Fabric. *IEEE Micro*, 32(1):32–43, March-April 2012.

[13] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Computing*, 14(1):210–223, 1985.

[14] F. Chung and L. Lu. The average distances in random graphs with given expected degrees. *PNAS*, 99(25):15879–15882, 2002.

[15] J. Cohen. Graph twiddling in a MapReduce world. *Computing in Science and Engineering*, 11(4), 2009.

[16] D. Eppstein. Parallel recognition of series-parallel graphs. *Inf. Comput.*, 98(1):41–55, 1992.

[17] J. Grochow and M. Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In *RECOMB*, 2007.

[18] F. Hormozdiari, P. Berenbrink, N. Przulj, and S. Sahinalp. Not all scale-free networks are born equal: The role of the seed graph in ppi network evolution. *PLoS Computational Biology*, 3(7), 2007.

[19] N. Iakovidou, S. Dimitriadis, N. Laskaris, K. Tsichlas, and Y. Manolopoulos. On the discovery of group-consistent graph substructure patterns from brain networks. *Journal of Neuroscience Methods*, 213(2):2014–213, 2013.

[20] M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *WWW*, 2015.

[21] T. Kolda, A. Pinar, T. Plantenga, C. Seshadhri, and C. Task. Counting triangles in massive graphs with MapReduce. *SIAM J. Scientific Computing*, 36(5), 2014.

[22] M. Middendorf, E. Ziv, and C. Wiggins. Inferring network mechanisms: The Drosophila melanogaster protein interaction network. *PNAS*, 102(9):3192–3197, 2005.

[23] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, October 2002.

[24] D. O'Callaghan, M. Harrigan, J. Carthy, and P. Cunningham. Network analysis of recurring youtube spam campaigns. *CoRR*, abs/1201.3783, 2012.

[25] N. Przulj, D. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 2004.

[26] M. Rahman, M. Bhuiyan, and M. Al Hasan. GRAFT: an approximate graphlet counting algorithm for large graph analysis. In *CIKM*, 2012.

[27] T. Schank and D. Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *WEA*, 2005.

[28] G. Slota and K. Madduri. Fast approximate subgraph counting and enumeration. In *ICPP*, pages 210–219, 2013.

[29] G. Slota and K. Madduri. Characterizing biological networks using subgraph counting and enumeration. In *PPSC*, 2014.

[30] G. Slota and K. Madduri. Complex network analysis using parallel approximate motif counting. In *IPDPS*, pages 405–414, 2014.

[31] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, 2011.

[32] G. Wu, M. Harrigan, and P. Cunningham. Classifying wikipedia articles using network motif counts and ratios. In *WikiSym*, 2012.

[33] Z. Zhao, M. Khan, V. Kumar, and M. Marathe. Subgraph enumeration in large social contact networks using parallel color coding and streaming. In *ICPP*, 2010.