2013 IEEE 13th International Conference on Data Mining

Subgraph Enumeration in Dynamic Graphs

Abhijin Adiga*, Anil Kumar S. Vullikanti*†, Dante Wiggins*
*Virginia Bioinformatics Institute, Virginia Tech
†Department of Computer Science, Virginia Tech
abhijin@vbi.vt.edu, vsakumar@vt.edu, danwte90@vt.edu

***Abstract*—A fundamental problem in many applications involving social and biological networks is to identify and count the number of embeddings of a given small subgraph in a large graph. Often, they involve dynamic graphs, in which the graph changes incrementally (e.g., by edge addition/deletion). We study the *Dynamic Subgraph Enumeration* (DSE) Problem, where the goal is to maintain a dynamic data structure to solve the subgraph enumeration problem efficiently when the graph changes incrementally. We develop a new data structure that combines two techniques: (i) the color-coding technique of Alon et al., 2008, for enumerating trees, and (ii) a dynamic data structure for maintaining the h-index of the graph (developed by Eppstein and Spiro, 2009). We derive worst case bounds for the update time in terms of the h-index of the graph and the maximum degree. We also study the empirical performance of our algorithm in a large set of real networks, and find significant improvement over the static methods.**

***Keywords*-Subgraph enumeration; color coding; h-index;**

I. Introduction

A fundamental problem in diverse data mining applications, ranging from social network analysis and semantic web to bioinformatics, is to count or identify embeddings of specific subgraphs in an underlying graph, e.g., [1], [2], [3], [4]. Existence of subgraphs is sometimes used to find anomalies or hidden relationships in the data, e.g., [2], [1]. In a number of bioinformatics approaches, counts of specific subgraphs are used to characterize the graphs [4], and frequent subgraphs in an ensemble of graphs are used to identify novel associations and functional groups, e.g., [5], [6].

Subgraph isomorphism and enumeration are computationally very challenging problems. Most of the current work is restricted to static networks, e.g., [7], [1], which develop parallel techniques that work on graphs with millions of nodes in several hours for small subgraphs (between 6 and 12). The best theoretical results for exact counting include $O(2^k n^{O(1)})$ time for paths [8] and $O(2^s n^{k-s+3} k^{O(1)})$ time [9] if the subgraph of size k has an independent set of size s. For approximate counting, Alon et al. [10] develop an approach with a significantly improved time of $O(mc^k)$, where m is the number of edges in the network.

Many of the settings that require subgraph enumeration involve dynamic graphs, e.g., social networks involve temporal changes, in which the graph evolves over time, e.g., [11], [12], and such applications require subgraph enumeration in dynamic graphs. Further, a widely used approach in statistics and computational sociology for generating networks with specific properties is the class of *Exponential Random Graph Models* (ERGM) [13], [14]; these are designed to generate random graphs with specific attributes, e.g., constraints on degree and specific subgraph counts. ERGMs are typically implemented by a Markov-chain Monte-Carlo (MCMC) approach: At each step t, a graph $G_t = G'$ is constructed with an incremental change from the earlier graph G_{t-1}, e.g., by the addition or deletion of a random edge. The change is rejected or accepted, depending on whether or not G' has the desired properties; this requires computing the properties of G_t dynamically.

This motivates the *Dynamic Subgraph Enumeration* ($\mathrm{DSE}(\mathcal{G}, H)$) problem: Given a subgraph H and a sequence $\mathcal{G}$ of graphs $G_1, G_2, \ldots$, in which G_{t+1} is obtained by modifying a single edge in G_t, the goal is to maintain a dynamic data structure for each G_t so that the number of subgraphs of G_{t+1} isomorphic to H can be estimated efficiently, significantly faster than recomputing them on G_{t+1} from scratch. Since subgraph enumeration in static graphs itself is very challenging, there is very limited work on the DSE problem [12], [11], [15], [16], [17], [18]. The work by Eppstein and Spiro [15], [16] gives the best known theoretical results for the DSE problem, but only considers subgraphs of size 3 and 4. Manjunath et al. [18] and Kane et al. [17] develop dynamic streaming algorithms for approximating the number of cycles and for any constant size subgraph, respectively, but these have higher space complexity than our results.

Our contributions. In this paper we develop provable algorithms for the $\mathrm{DSE}(\mathcal{G}, H)$ problem that give approximate counts, within any desired accuracy. We derive rigorous bounds on the update time of our algorithms in terms of the properties of G and H, which are validated through experiments on a large number of diverse graphs. Our data structure and algorithm combines two different techniques: (i) The *color-coding* technique of Alon et al. [10] that gives a randomized approximation algorithm for counting the number of trees on k vertices in a graph. (ii) The *h-index* of a graph (which is the largest integer h such that there exist h vertices of degree at least h (Hirsch [19])), and a dynamic data structure of Eppstein and Spiro [15], [16] to efficiently maintain the set of vertices with degree h or more. We let root(T) denote the root of a tree T, and consider the set $\mathcal{T}(d, \ell)$ of trees of depth d (relative to the root), in which the root has ℓ children. Our main contributions are summarized below.

1. The DSE *problem for trees in* $\mathcal{T}(d, 2)$. We first consider trees in the set $\mathcal{T}(2, 2)$, in which the root has at most two children. Let h denote the maximum h-index of any graph G_t in the sequence $\mathcal{G}$. When $T \in \mathcal{T}(2, 2)$ is a tree with

1550-4786/13 $31.00 © 2013 IEEE
DOI 10.1109/ICDM.2013.157

depth 2, and the root has 2 children, we design a dynamic data structure, $\mathcal{D}_t^h$ with update time $O\left(2^k e^k \frac{h^2}{\epsilon^2} \log n\right)$ for approximating the number of embeddings of T in G_t within a factor of $1 \pm \epsilon$ with probability at least $1 - 1/n$, for any given $\epsilon > 0$. For a graph G_t, an integer β and vertex $u \in V(G)$, we use $G_t^\beta(u)$ to denote the graph induced by the vertices in G_t at distance at most β from u. Let $n_\beta = \max_{u,t} |V(G_t^\beta(u))|$ and $m_\beta = \max_{u,t} |E(G_t^\beta(u))|$. When $T \in \mathcal{T}(d, 2)$, and $d > 2$, our data structure $\mathcal{D}_t$ can be updated in time $O\left(2^k e^k \log n \frac{1}{\epsilon^2} h^2 m_{d-1}\right)$, and gives an approximation of the number of embeddings of T in graph G_t within a factor of $1 \pm \epsilon$ with probability at least $1 - 1/n$ for any $\epsilon > 0$. It is important to note that this update time bound depends on the depth d and m_{d-1}, instead of on m_k, which can be much larger than m_d, where k denotes the number of nodes in T; further, note that we have m_{d-1} in the bound, instead of m_d, which can be much smaller, and can lead to a significantly better bound if the h-index is small. For small d, m_d is typically much smaller than $|E(G)|$ in many graphs (which is borne out from our empirical results). Also, the above bound on the update time is a "worst case" bound, which holds for the addition/deletion of any possible edge e_{t+1} to graph $G_t \in \mathcal{G}$; in practice, or if edge e_{t+1} is picked randomly, the update time is likely to be much smaller.

2. The DSE *problem for other trees and constant size subgraphs*. When T is a tree in $\mathcal{T}(d, \ell)$, with $\ell > 2$, we show that the color-coding technique can be adapted to run in a dynamic manner with update time of $O\left(2^k e^k \log n \frac{1}{\epsilon^2} m_d\right)$. When T is not a tree, we extend our algorithm for the DSE problem, and bound the update time in terms of the *treewidth* $\alpha(H)$ of H; the treewidth quantifies how close H is to a tree (see Section V for details). Specifically, for any given $\epsilon > 0$, our algorithm gives a $1 \pm \epsilon$ approximation to the number of embeddings of H in G_t in time $O(2^k e^k |V(G^{3d})|^\alpha \log n)$, with probability at least $1 - 1/n$. The best static algorithms for the counting problem take time $\Omega(n^{2\alpha})$; therefore, this bound is a significant improvement. Our algorithm generalizes our result for trees in $\mathcal{T}(d)$, along with the technique from [10] for tree-width bounded subgraphs.

3. Empirical performance. We study the performance of our algorithms on 14 different real networks from [20]. The update times for paths of length 5 and 9 and a complete binary tree of depth 2 are in general within a factor of 10% of the static time, for most of the graphs. The memory usage of our algorithms is under 6GB for the largest graph, for a subgraph which is a path of size 9. This shows that our approach can be of significant practical utility in biological and social networks, where subgraph enumeration is commonly used.

The novel aspect of our work is the combination of color-coding and h-index based techniques, which improves upon prior bounds for the DSE problem, for some classes of graphs. For trees in $\mathcal{T}(2, 2)$, a direct adaptation of the color-coding technique would result in an update time that grows as m_2, which can be as large as Δ^2, where Δ denotes the maximum node degree in G; this can be much higher than our bound which grows as h^2 (For a comparison between h and Δ for some real-world networks, we refer the reader to Table I). For trees in $\mathcal{T}(d, 2)$, with $d > 2$, our update time bound which depends on $h^2 m_{d-1}$ improves up on the direct bound based on color-coding which depends on m_d. While our main focus is theoretical, our empirical results show that our approach performs quite well on many different real networks.

Organization. Many of the technical details, including proofs, are available in the complete online version [21]. We summarize the related work in Section II and start with the basic concepts and notation in Section III. We present our algorithm for trees in Section IV, and the extension to general subgraphs in Section V in the full version [21]. We discuss some of the empirical results in Section VI and conclude in Section VII.

II. Related Work

There has been a lot of research on subgraph isomorphism and enumeration problems because of their interest in diverse applications. These problems are computationally very challenging; Aravind et al. [22] show that counting cliques and independent sets of size k in a given graph G are $W[1]$-hard— informally this means that for a graph with n vertices and a subgraph of size k, these problems are not expected to be solvable in much better than $O(n^k)$ time. There are results that improve on this bound slightly, and under specific assumptions on the subgraph, e.g., Eisenbrand *et al*. [23] give a bound of roughly $O(n^{\omega k/3})$, where ω denotes the exponent of the best possible matrix multiplication algorithm, and Vassilevska *et al*. [9] and Kowaluk *et al*. [24] improve this to $O(2^s n^{k-s+3} k^{O(1)})$, where the template has an independent set of size s. The best result for counting the number of paths of length k is an $O(2^k n^{O(1)})$ time algorithm by Williams [8], and abbreviated as $O^*(2^k)$. For arbitrary subgraphs of size k, detecting the existence of an embedding can be done in time $O^*(2^k)$ [25]. The above results are for exact enumeration of subgraphs, and because of their hardness, there has also been a lot of work on approximation algorithms. Alon et al. [10] developed the color coding technique for approximating the number of paths, trees and tree-width bounded subgraphs of size k in time $O(k|E|2^k e^k \log(1/\delta)\frac{1}{\varepsilon^2})$, where ε and δ are error and confidence parameters, respectively. Huffner et al. [26] optimize the parameters of the color coding technique, including the choice of the number of colors, and efficient data structures to minimize the running time. Aravind et al. [22] give results for other special classes of subgraphs, including cliques.

A number of practical heuristics have also been developed for various versions of these problems, especially for the frequent subgraph mining problem, e.g., [5], [6]; many of these are based on candidate generation approaches, with heuristics for speeding up the pruning and exploration. Several parallel techniques have been developed for variants of subgraph enumeration problems, including [1], [27], [7]. There has been limited work on subgraph analysis in dynamic graphs. Some of the related work is on finding communities and anomalies in dynamic graph models, e.g., [12], [11]. The work by Eppstein

and Spiro [15], [16] gives the best known theoretical results for the DSE problem, but only considers subgraphs of size 3 and 4. They design the dynamic h-index data structure we use here, and give an $O(h^2)$ time algorithm for subgraphs of size 4. They also do an extensive analysis of the h-index of a wide variety of real world graphs, and show that it tends to be small, e.g., relative to the the maximum degree. Other recent work is by Manjunath et al. [18], who develop algorithms for dynamically estimating the number of cycles, while Kane et al. [17] develop an algorithm for dynamically estimating the number of embeddings of any constant size subgraph. These results consider the problem in a streaming setting, where the edges are streamed, which is an even more challenging problem. However, the space requirement for their algorithm is $\Omega(m^k\Delta^k/\phi(H)^2)$, which can be much larger than our bounds, in general; here $\phi(H)$ denotes the number of embeddings of H in G, as defined in the next section.

III. Preliminaries

Given graphs $G = (V, E)$ and $H = (V_H, E_H)$, a homomorphism from H to G is a mapping $f : V_H \to V$ such that if $(u, v) \in E_H$ then $(f(u), f(v)) \in E$. H is generally a small graph (constant size) and we refer to it as a template/subgraph. We are interested in the number of homomorphisms of H into G, informally referred to as the number of "embeddings" or "copies" of H in G, and denote it by $\phi(H)$. For vertex $v \in V$, we use $N(v)$ to denote the set of neighbors of v. Let $\Delta = \Delta(G) = \max_{v\in V} |N(v)|$ denote the maximum degree in the graph G. Let $d(v, G)$ denote the degree of v in G. Let $n(u, v) = |N(u) \cap N(v)|$ denote the number of common neighbors of u and v. Let $[\ell]$ denote the set $\{1, \ldots, \ell\}$. For a graph T, we use V_T and E_T to denote the set of vertices and edges of T.

We focus on dynamic graphs in this paper, and assume we are given a sequence $\mathcal{G} = G_1, G_2, \ldots$ of graphs. We let $G_t = (V_t, E_t)$ denote the graph at time t. In most of this paper, we assume that $V_t = V$ for each time t, so that the vertex set remains the same, and only the edges change. Further, in order to keep our algorithms tractable, we assume that E_{t+1} is obtained by adding/deleting a single edge e_t from E_t. Let $\phi(H, G_t)$ denote the number of embeddings of H in G_t; we will also sometimes refer to it as $\phi(H)$ or ϕ_t, if H and/or G_t are clear from the context. For a graph G, an integer β and a set of vertices $A \subset V(G)$, we use $G^\beta(A)$ to denote the graph induced by the vertices in G at distance at most β from any vertex in A. In the first part of this paper, we focus on subgraphs H which are trees. We use $\mathcal{T}(d)$ to denote the set of all (arbitrarily rooted) trees with depth d; we use $\mathcal{T}(d, \ell)$ to denote the set of trees in $\mathcal{T}(d)$ in which the root has ℓ children. Formally, the focus of this paper is the following problem.

Definition 1 (Dynamic Subgraph Enumeration DSE$(\mathcal{G}, H)$ problem**).** Given a sequence of graphs $\mathcal{G} = G_1, G_2, \cdots, G_t, G_{t+1}, \cdots$ and a subgraph H, compute ϕ_t, the number of embeddings of H in G_t, efficiently *for each* t.

The goal is to have an algorithm which computes the count for G_{t+1} by updating the count for G_t and therefore runs significantly faster than the method of recomputing the solution for each G_t. This is possible under the assumption that the graphs G_t and G_{t+1} differ by few links or nodes.

Remark 2. One may be tempted to compute subgraph counts for any arbitrary graph G by viewing it as a sequence $I = G_0, G_1 \cdots, G_t = G$ where I is an empty graph on $V(G)$ vertices and G_i is obtained by adding a few edges to G_{i-1} and subsequently using the methods developed in this paper. This will turn out to be very inefficient. Our algorithms are not designed to substitute subgraph counting algorithms for static graphs.

Our results also extend to other variants of this problem, such as finding a minimum weight embedding of H in a weighted version of G, following [26].

Since our algorithm combines the color-coding and dynamic h-index approaches, we first describe them briefly here.

A. The color-coding technique

The color-coding technique of Alon et al. [10] gives an efficient randomized approximation algorithm for computing ϕ_0, and is based on the following two key ideas: (i) we consider a coloring of G using k colors, where $k \geqslant |V_T|$, and compute the number of "colorful" embeddings of H in G (where a colorful embedding is one in which each node has a distinct color– see Fig. 2)— the main insight is that this problem can be solved by dynamic programming (in contrast to the original problem of counting all embeddings), and (ii) if the coloring of G is done randomly, the expected number of colorful embeddings equals the total number of embeddings times $\frac{m!\binom{k}{m}}{k^m}$. Below, we first discuss step (ii), and how the overall color-coding scheme works (in Section III-A1), and then discuss the dynamic program for step (i).

1) **Color coding (overall approach):** A random k-vertex coloring of a graph corresponds to coloring each vertex of the graph independently and uniformly at random with one of the k colors from the set $[k] = \{1, \ldots, k\}$ of colors. A colorful copy of T in G is a copy of T (non-induced) in which all its vertices are assigned unique colors (see Fig. 2 for an example); we use $\text{col}_i(v)$ to denote the color assigned to v in the ith coloring. Given a random k-vertex coloring of G, the expected number of colorful copies of T, denoted by $\tau_i(T) = \tau_i(T, G)$, is $\tau_i(T, G) = \phi(T)\frac{k!}{k^k}$; we sometimes also use τ_i instead of $\tau_i(T)$, and also drop the subscript i, if it is clear from the context. In [10], an (ϵ, δ)-approximation algorithm to estimate $\phi(T)$ is given, i.e., the estimate of the algorithm lies within $[(1 - \epsilon)\phi(T), (1 + \epsilon)\phi(T)]$ with probability $1 - 2\delta$. The general procedure, ColorCoding is depicted in Fig. 1. It involves considering $O(e^k)$ random k-vertex colorings of G. For each coloring, the counting step described in Section III-A2 is used to efficiently compute the number of colorful embeddings of T. Finally, a median of averages method is used to approximate $\phi(T)$. For the analysis of the algorithm in Fig. 1, see [10].

Input: Graph G, subgraph T of size k, real numbers $\epsilon, \delta > 0$.

(1) Let $a = \log(1/\delta)$ and $b = \frac{4k^k}{\epsilon^2 k!}$. Assume that a and b are integers.

(2) For $i = 1$ to ab,

(a) Color each vertex using a random color from $\{1, \ldots, k\}$.

(b) COLORFULCOUNT(T): This procedure depends on T. The output of this step is $\tau_i(G)$, the number of colorful copies of T in G for the ith random coloring.

(3) For $1 \leqslant j \leqslant a$, let $\gamma_j = \frac{1}{b} \sum_{i=b(j-1)+1}^{bj} \tau_i(G)$.

(4) Let γ be the median of $\gamma_1, \ldots, \gamma_a$.

(5) Return $\hat{\phi}(T) = \frac{1}{\sigma} \gamma k^k / k!$, where σ is the number of automorphisms of T.

Fig. 1. COLORCODING: A general framework for an (ϵ, δ)-approximation algorithm to count the number of non-induced copies of a motif T.

2) **The counting step:** Consider $\text{col}_i(\cdot)$, the ith k-vertex random coloring of G. When $T \in \mathcal{T}(d)$, the number of colorful occurrences of subgraph T can be efficiently computed using a dynamic programming routine as described in [10] in $O(2^k|E(G)|)$ time. This method can be extended to other subgraphs such as cycles and in general, treewidth-bounded motifs as well (as will be discussed in Section V). Here, we limit our discussion to $T \in \mathcal{T}(d)$. Let $r \in V(T)$ be such that every vertex of T is within distance d from it; $T(r)$ denotes T rooted at vertex r and $\text{root}(T) = r$. The ongoing discussion will be supported by an example illustrated in Fig. 2.

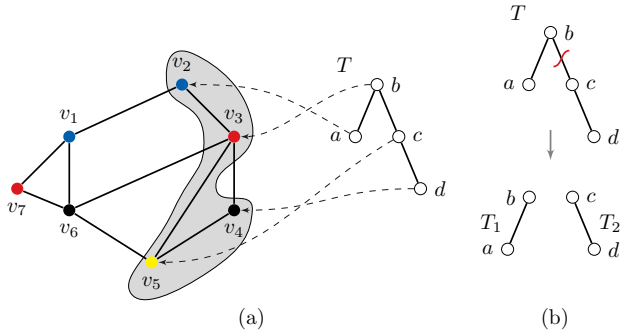

Fig. 2. (a) Example showing a tree T in $\mathcal{T}(2,2)$ and one of its embeddings in a graph G. The embedding shown here is also a colorful embedding, which maps the root b of T to node v_3. (b) Partitioning of the tree T to obtain the set of rooted subtrees $\mathcal{S}(T)$.

Definition 3. $\mathcal{S}(T(r))$ denotes all the distinct rooted subtrees of $T(r)$ obtained inductively as follows: First of all $T(r) \in \mathcal{S}(T(r))$. For each $T'(r') \in \mathcal{S}(T(r))$, there are exactly two subtrees $T'_1(r')$ and $T'_2(r'')$ in $\mathcal{S}(T(r))$ where r'' is a neighbor of r' in T. These two subtrees are obtained as a result of removing the edge (r', r'') from T'.

This is illustrated in Fig. 2(b), where, T_1 and T_2 are obtained by removing the edge (b, c). For this tree, $\mathcal{S}(T(b))$ consists of $T(b), T_1(b), T_2(c)$ and one-vertex trees with roots a, b, c and d.

Definition 4. Colorful subtree counts $C_i(\cdot)$ and data structure $\mathcal{C}$: For every tree $T'(r') \in \mathcal{S}(T(r))$, $x \in V(G)$ and the ith coloring of G, let $C_i(x, T'(r'), S)$ denote the number of colorful trees isomorphic to T' rooted at x, where S is a color subset of size $|T'|$. $\mathcal{C}$ is the collection of all these counts.

We drop the subscript i whenever there is no ambiguity. In Fig. 2, for example

$$C(v_3, T_1(b), \{\text{red, blue}\}) = C(v_4, T_2(c), \{\text{black, yellow}\}) = C(v_6, T_2(c), \{\text{black, yellow}\}) = 1, \quad (1)$$
$$C(v_5, T_2(c), \{\text{black, yellow}\}) = 2.$$

The key idea is that the number of colorful copies of T in G can be computed efficiently by using an inductive routine COLORFULCOUNT(T) described in Fig. 3. The quantity $\hat{\phi}(T) = \frac{1}{\sigma} \sum_{x \in V} C(x, T(r), [k])$ is the number of colorful copies of T where σ is the number of automorphisms of T.

In Fig. 2, $C(v_3, T(b), \{\text{red, blue, black, yellow}\})$ can be written as the sum $\sum_{S_1, S_2} C(v_3, T_1(b), S_1)\big(C(v_5, T_2(c), S_2) + C(v_6, T_2(c), S_2) + C(v_4, T_2(c), S_2)\big)$, over all partitions $S_1 \cup S_2 = \{\text{red, blue, black, yellow}\}$, since v_3 has three incident edges (v_3, v_4), (v_3, v_5) and (v_3, v_6). From (1), it follows that $C(v_3, T(b), \{\text{red, blue, black, yellow}\}) = 3$.

B. Maintaining the h-index set dynamically

The h-index of a graph $G(V, E)$ is the largest h such that the graph contains h vertices of degree at least h, e.g., the h-index of the graph in Fig. 2 is 3. Let $H \subseteq V$ be a set of vertices that form the h-index. $(H, V \setminus H)$ is referred to as an h-partition of $V(G)$.

Property 5. *([16]) There exists an h-partition $(H, V \setminus H)$ of G such that all vertices x with $d(x, G) > h$ belong to H.*

Eppstein and Spiro [16] developed an algorithm for updating the h-index and h-partition of a graph in constant time when a vertex (or edge) is inserted or deleted. For completeness, we now describe their update method when a single edge is added to the graph. A similar method applies to deleting an edge. First of all we assume without loss of generality that H satisfies Property 5 for G. We maintain a data structure

Consider the ith random k-vertex coloring, $\text{col}_i(\cdot)$, of G. Let $T(r)$ denote the tree T rooted at vertex r; we assume that every other vertex is within distance d from r. Recall Definitions 3 and 4 of $\mathcal{S}(T(r))$ and $\mathcal{C}$ respectively.

(1) If $|T'(r')| = 1$, then, for all $l \in [k]$

$$C_i(x, T'(r'), \{l\}) = \begin{cases} 1, & \text{if col}(x) = l, \\ 0, & \text{otherwise.} \end{cases}$$

(2) If $|T'(r')| > 1$, then, let $T'_1(r')$ and $T'_2(r'')$ be its two child subtrees in $\mathcal{S}(T(r))$.

$$C_i(x, T'(r'), S) = \sum_{y \in N(x)} \sum_{S_1, S_2} C_i(x, T'_1(r'), S_1) C_i(y, T'_2(r''), S_2),$$

where, $S_1 \cup S_2 = S$, $S_1 \cap S_2 = \varnothing$.

These values are stored in $\mathcal{C}$ for every random coloring of G.

Fig. 3. COLORFULCOUNT($T \in \mathcal{T}(d)$): Procedure to count colorful motifs $T \in \mathcal{T}(d)$ in a graph.

$\mathcal{H} = (H, V - H)$ as described in Fig. 4; its performance is summarized in the following lemma.

Data structure $\mathcal{H} = (H, V - H)$ consists of the following:
(1) A dictionary F mapping each vertex to its degree,
(2) the set H, the set $B = \{x \in H | d(x, G) = |H|\}$ and
(3) a dictionary C mapping each non-negative integer i to the set $\{x \in V \setminus B \mid d(x, G) = i\}$.

Updating procedure after edge addition/deletion:
Suppose we add edge (u, v) in G. Let $G' = G + (u, v)$. Let h_G and $h_{G'}$ denote the h-indices of G and G' respectively. We have the following cases (similar steps follow for edge deletions):
(1) $d(u, G), d(v, G) \neq h$: We have $h_{G'} = h_G$, $H' = H$ and B is unchanged. If $d(u, G) = j$, we move u from $C[j]$ to $C[j + 1]$; similarly for v.
(2) $d(u, G) = h, d(v, G) \neq h$ (analogous if $d(u, G) \neq h, d(v, G) = h$): If $B = \varnothing$ or $\{u\}$, we set $h_{G'} = h_G + 1$, $H' = H \cup \{u\}$, $B = C[h_G + 1]$ and $C[h_G + 1] = \varnothing$. Otherwise, $|B| \geqslant 2$ and we set $h_{G'} = h_G$. If $u \in B$, we remove u from B. If $u \notin B$, we choose an $x \in B$ and set $H' = (H \setminus \{x\}) \cup \{u\}$. C is updated with regard to changes in the status of u, v and x respectively.
(3) $d(u, G), d(v, G) = h$: If $u, v \in B$, then, $h_{G'} = h_G$ and we set $H' = H$ and remove u, v from B. If $u, v \notin B$, then we have two cases. If $|B| \geq 2$, then $h_{G'} = h_G$ and we choose two vertices $x, y \in B$ and set $H' = (H \setminus \{x, y\}) \cup \{u, v\}$, and update C w.r.t. u, v, x, y. If $|B| < 2$, then, $h_{G'} = h_G + 1$. We set $H' = (H \setminus B) \cup \{u, v\}$. If $u \notin B$ and $v \in B$, then, we again have two cases. If $B = \{v\}$, then, $h_{G'} = h_G + 1$ and we set $H' = H \cup \{u\}$. Else, we choose an $x \in B$ other than v, set $H' = (H \setminus \{x\}) \cup \{u\}$. In all these cases where $h_{G'} = h_G + 1$, we set $B = C[h_G + 1]$ and $C[h_G + 1] = \varnothing$.

Fig. 4. UPDATEHINDEX: The data structure $\mathcal{H}$ and the algorithm to update it [16].

Lemma 6 ([16]). *Let $G' = G \pm (u, v)$ and suppose H satisfies Property 5 for G. The procedure* UPDATEHINDEX *of Fig. 4 updates H to H' such that: (1) H' satisfies Property 5 for G'. (2) $|H \oplus H'| \leqslant 4$ where, $\oplus$ denotes symmetric difference. (3) For any $x \in H \oplus H'$, $d(x) \in \{h, h+1\}$ or $d(x) \in \{h-1, h\}$ depending on whether the edge (u, v) was inserted or deleted respectively.*

IV. THE GENERAL FRAMEWORK FOR $T \in \mathcal{T}(d, 2)$

We now describe our general framework for the $\mathrm{DSE}(\mathcal{G}, H)$ problem when H is a tree T of depth d with the root having 2 children, i.e. $T \in \mathcal{T}(d, 2)$; we discuss extension to other subgraphs later in Section V. Our approach combines the dynamic h-index data structure with the color-coding technique.

Data structure. Suppose root(T) has two sub-trees T_1 and T_2, of sizes k_1 and k_2, respectively (this can be extended easily to the more general setting). The main data structure $\mathcal{D}_t^h$ used in our algorithm is described in Fig. 5 and its update scheme is described in Fig. 6. As in the case of color-coding, our algorithm involves keeping $R = O\big(\log(1/\delta)\frac{1}{\varepsilon^2}\big)$ colorings of the graphs (recall Fig. 1); we maintain $\mathcal{D}_t^h$, for each coloring, but for exposition, we only describe the data structure and algorithm for a specific coloring. Among other counts, the algorithm maintains the counts $C(x, T', S')$ of the kind used in color-coding, for all nodes $x \in V$, sub-trees T' and color sets S'. When $T \in \mathcal{T}(2, 2)$, these can be maintained very easily, but when $d > 2$, we adapt the color-coding technique for updating them.

Let T_1 and T_2 denote the subtrees of the root of T. Let $k_i = |T_i|$. Recall the notation $\mathcal{S}(T_i)$ as described in Fig. 2(b). $\mathcal{D}_t^h$ contains the following:
(1) Data structure $\mathcal{C}$ of Fig. 3 to compute and update the quantities $C(v, T', S')$ for all $T' \in \mathcal{S}(T_1) \cup \mathcal{S}(T_2)$.
(2) The partition $(H, \overline{H})$ using UPDATEHINDEX of Fig. 4.
(3) For each $x \in V$, $i = 1, 2$ and S_i of size k_i, we maintain $A_x^i(S_i) = \sum_{y \in N(x) \cap \overline{H}} C(y, T_i, S_i)$.
(4) For each partition S_1, S_2 such that $S_1 \cup S_2 \cup \{\mathrm{col}(x)\} = S$, the sum $\sum_{x \in V} A_x^1(S_1) \cdot A_x^2(S_2)$.
(5) For each $y \in H$, $i = 1, 2$ and S_i of size k_i, the quantity $F_y^i(S_i) = \sum_{x \in N(y)} A_x^i(S_i)$.
(6) We maintain $\sum_{x \in V} C(x, T, S)$.

Fig. 5. Description of data structure $\mathcal{D}_t^h$.

Suppose edge $e_t = (u, v)$ is added. $\mathcal{D}_t^h$ is updated in the following manner; an analogous sequence of steps follow if e_t is deleted.
(1) For each $T' \in \mathcal{S}(T_1) \cup \mathcal{S}(T_2)$, update $C(x, T', S')$, for $x \in V$ which are affected, using COLORFULCOUNT(T) of Fig. 3.
(2) Let $\mathcal{A}^i(S_i)$ be the set of vertices x such that $C(x, T_i, S_i)$ is updated in the previous step for $i = 1, 2$.
(3) Update the h-index partition $(H, \overline{H})$ using UPDATEHINDEX of Fig. 4.
(4) For each $y \in \mathcal{A}^i(S_i) \cap \overline{H}$, for each $x \in N(y)$: (a) increase $A_x^i(S_i)$ by $\alpha = C(y, T_i, S_i) - C^{\text{old}}(y, T_i, S_i)$, and (b) if $x \in \overline{H}$, then for each $y' \in N(x) \cap H$: increase $F_{y'}^i(S_i)$ by α.
(5) For each $x \in H \cap \mathcal{A}^i(S_i)$, for each $y' \in N(x) \cap H$: increase $F_{y'}^i(S_i)$ by $A_x^i(S_i) - A_x^i(S_i)^{\text{old}}$.
(6) For each $y \in \mathcal{A}^i(S') \cap \overline{H}$: Consider each $x \in N(y)$ only if x has not been considered before in this step. Increase $\sum_{x'} A_{x'}^1(S') A_{x'}^2(S'')$ by $A_x^1(S') A_x^2(S'') - A_x^1(S')^{\text{old}} A_x^2(S'')^{\text{old}}$, where $S'' = S - S' - \{col(x')\}$.
(7) Finally, for S_1, S_2 such that $|S_i| = k_i$, compute $\psi(S_1, S_2) = \sum_{x \in V} A_x^1(S_1)\, A_x^2(S_2) + \sum_{y \in H} C(y, T_1, S_1)\, F_y^2(S_2) + \sum_{y \in H} C(y, T_2, S_2)\, F_y^1(S_1) + \sum_{y,z \in H} C(y, T_1, S_1) C(z, T_2, S_2) n(y, z)$, and update $\sum_{x \in V} C(x, T_0, S) = \sum_{S_1, S_2} \psi(S_1, S_2)$.

Fig. 6. UPDATEDYNAMIC: Algorithm to update data structure $\mathcal{D}_t^h$.

Lemma 7. *For each t,* UPDATEDYNAMIC *of Fig. 6 correctly computes the number of colorful embeddings for graph G_t.*

Proof: We show that $\sum_x C(x, T, S) = \sum_{S_1, S_2} \psi(S_1, S_2)$, and that $\psi(S_1, S_2)$ is computed correctly from the expression in Step 7 of UPDATEDYNAMIC; the lemma then follows. First, observe that $C(x, T, S) = \sum_{S_1, S_2} \sum_{y_1, y_2 \in N(x)} C(y_1, T_1, S_1) C(y_2, T_2, S_2)$, where the first sum is over all partitions S_1, S_2 such that $|S_i| = k_i$. This can be rewritten as $\sum_{S_1, S_2} \big(\sum_{y \in N(x)} C(y, T_1, S_1)\big)\big(\sum_{y \in N(x)} C(y, T_2, S_2)\big)$, because $S_1 \cap S_2 = \phi$, so that the terms of the form $C(y, T_1, S_1) \cdot C(y, T_2, S_2) = 0$. Let $B_x^i(S_i) =$

$\sum_{y\in N(x)\cap H} C(y,T_i,S_i)$ for $i=1,2$. Recall the definition of $A_x^i(S_i) = \sum_{y\in N(x)\cap \overline{H}} C(y,T_i,S_i)$ in the data structure $\mathcal{D}_t^h$ in Fig. 5. It follows that we can rewrite $C(x,T,S)$ as $\sum_{S_1,S_2}\left(A_x^1(S_1)+B_x^1(S_1)\right)\left(A_x^2(S_2)+B_x^2(S_2)\right) = \sum_{S_1,S_2} A_x^1(S_1)A_x^2(S_2)+A_x^1(S_1)B_x^2(S_2)+A_x^2(S_2)B_x^1(S_1)+B_x^1(S_1)B_x^2(S_2)$. We now prove that

$$\sum_x A_x^2(S_2)B_x^1(S_1) = \sum_{y\in H} C(y,T_1,S_1)F_y^2(S_2), \tag{2}$$

$$\sum_x A_x^1(S_1)B_x^2(S_2) = \sum_{y\in H} C(y,T_2,S_2)F_y^1(S_1), \tag{3}$$

$$\sum_x B_x^1(S_1)B_x^2(S_2) = \sum_{y,z\in H} C(y,T_1,S_1)C(z,T_2,S_2)n(y,z), \tag{4}$$

where $F_y^1(S_1)$ and $F_y^2(S_2)$ are the quantities maintained in $\mathcal{D}_t^h$ (see Fig. 5). First, consider (2):

$$\begin{aligned}\sum_x A_x^2(S_2)B_x^1(S_1) &= \sum_x A_x^2(S_2) \sum_{y\in N(x)\cap H} C(y,T_1,S_1)\\ &= \sum_{y\in H} C(y,T_1,S_1) \sum_{x\in N(y)} A_x^2(S_2)\\ &= \sum_{y\in H} C(y,T_1,S_1)F_y^1(S_1).\end{aligned}$$

Expression (3) follows in a similar way. For (4), $\sum_x B_x^1(S_1)B_x^2(S_2) =$

$$\begin{aligned}&\sum_x \sum_{y\in N(x)\cap H} C(y,T_1,S_1) \sum_{z\in N(x)\cap H} C(z,T_2,S_2)\\ &= \sum_{y,z\in H} \sum_{x\in N(y)\cap N(z)} C(y,T_1,S_1)C(z,T_2,S_2)\\ &= \sum_{y,z\in H} n(y,z)C(y,T_1,S_1)C(z,T_2,S_2).\end{aligned}$$

Next, observe that $A_x^i(S_i)$ needs to be updated for some x,i,S_i only if there exists $y\in N(x)\cap\overline{H}$ such that $C(y,T_i,S_i)$ is revised; all such y belong to $\mathcal{A}^i(S_i)$, by the definition in Step 2 of Fig. 6. This implies that all affected $A_x^i(S_i)$ are updated correctly in Step 4(a). Next, note that for some $y'\in H$, $F_{y'}^i(S_i) = \sum_{x\in N(y')} A_x^i(S_i) = \sum_{x\in N(y')\cap H} A_x^i(S_i) + \sum_{x\in N(y')\cap\overline{H}} A_x^i(S_i)$. If $x\in N(y')\cap\overline{H}$ and $A_x^i(S_i)$ is updated, then there exists $y\in N(x)\cap\overline{H}\cap\mathcal{A}^i(S_i)$; therefore, $\sum_{x\in N(y')\cap\overline{H}} A_x^i(S_i)$ is updated correctly in Step 4(b). If $x\in N(y')\cap H$ and $A_x^i(S_i)$ is updated, $\sum_{x\in N(y')\cap H} A_x^i(S_i)$ is updated correctly in Step 5, so that $F_{y'}^i(S_i)$ is updated correctly in Fig. 6. Finally, the sum $\sum_{x'} A_{x'}^1(S')A_{x'}^2(S'')$ is updated correctly in Step 6, which implies that $\psi(S_1,S_2)$ is correctly computed in Step 7. ■

A. The update time when $d=2$

We first consider the case when $T\in\mathcal{T}(2,2)$. In this case, the subtrees T_i correspond to star graphs K_{1,k_i}, $i=1,2$ and the data structure $\mathcal{C}$, consisting of the colored counts $C(v,T_i,S_i)$ can be maintained easily, as described in UPDATESTAR of Fig. 7.

Data structure: Let T' be the depth 1 tree. For each vertex v, we maintain the following information in a dictionary, denoted by $\mathcal{C}$, for a specific coloring of G.

- The number of neighbors of v having color j, denoted by $n_j(v)$.
- The quantity $C(v,T',S')$.

Algorithm for updating: If an edge $e_t=(u,v)$ is added in step t, the data structure $\mathcal{C}$ can be updated in the following manner (similar steps follow for edge deletion):

1) Let $j=\text{col}(u)$ and $j'=\text{col}(v)$. We increase $n_j(v)$ and $n_{j'}(u)$ by 1.
2) For all possible color sets S' with $|S'|=|T'|$ and $\text{col}(u)=j\in S'$:
   a) If $n_j(v)=1$, $C(v,T',S') = \Pi_{l\in S',\text{col}(v)\neq l} n_l(v)$
   b) Else $C(v,T',S') = C(v,T',S')n_j(v)/(n_j(v)-1)$.

Fig. 7. UPDATESTAR: The data structure $\mathcal{C}$ for maintaining colored counts for depth 1 trees and the algorithm to update it.

Lemma 8. *For $T\in\mathcal{T}(2,2)$,* UPDATEDYNAMIC *of Fig. 6 coupled with* UPDATESTAR *of Fig. 7 updates the data structure $\mathcal{D}_t^h$ in $O(h^2)$ time.*

Proof: We analyze the running time of each step of UPDATEDYNAMIC of Fig. 6. Let c_s denote the maximum of the number of possible color sets S_i of size k_i, $i=1,2$. Since we work with constant size subgraphs, c_s is a constant. Since the two subtrees of T, T_1 and T_2 are depth 1 trees, for Step 1 of Fig. 6, we can apply UPDATESTAR of Fig. 7 and it is easy to see that this algorithm takes $O(c_s)$ steps to update $\mathcal{C}$. Also, note that $\mathcal{A}^i(S_i)$ can only consist of u,v.

From UPDATEHINDEX of Fig. 4 and Lemma 6, it follows that Step 2 takes $O(1)$ time.

Step 4(a) takes time $O(h)$, since $|\mathcal{A}^i(S_i)|\leq 2$, since nodes in $\overline{H}$ have degree at most h. Similarly, Step 4(b) takes time $O(h^2)$, since it is only done if $x\in\overline{H}$. Step 5 takes $O(h^2)$ time, since $|H|=h$. It also follows that Step 6 takes $O(h)$ time, for the same reason.

In Step 7, note that $n(y,z)$ changes in G_t only if one of y,z is u or v, and y,z are within distance 2. Therefore, $n(y,z)$ can be updated in $O(h)$ time. Finally, computing each $\psi(S_1,S_2)$ in Step 6 takes $O(h^2)$ time because it involves examining all pairs of vertices $y,z\in H$, so that Step 6 takes $O(h^2)$ time overall. ■

Now, we summarize the performance of the dynamic enumeration algorithm for any $T\in\mathcal{T}(2,2)$.

Theorem 9. *For any $T\in\mathcal{T}(2,2)$, and any $\epsilon>0$, the number of embeddings ϕ_t of T in G_t can be approximated within a factor of $1\pm\epsilon$ with probability at least $1-1/n$ in time $O\left(\frac{h^2}{\epsilon^2}\log n\right)$ for $t=O(n)$.*

Proof: First, we give a brief outline of the process. For $t=0$, the number of colorful embeddings $\hat{\phi_0}$ is computed using COLORCODING of Fig. 1 with $\delta=2n^2$. The data structure $\mathcal{D}_0^h(j)$ for each coloring j and the h-index partition $(H,\overline{H})$ are initialized. Subsequently, for every $t>0$, from Lemma 7, it follows that the algorithm in Fig 6 correctly computes $\hat{\phi}_t$.

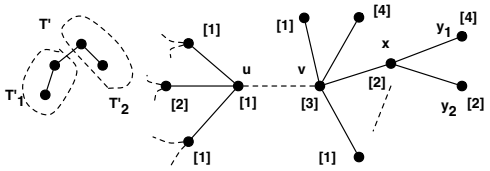

Fig. 8. Edge $e_t = (u, v)$ added in step t. The tree T' is partitioned into T'_1 and T'_2 and part of the graph G is shown. A specific coloring of the graph G is shown, with the colors for individual vertices shown in $[\cdot]$.

Note that the total number of different colored copies of G_t is $R = \log(1/\delta)\frac{4k^k}{\varepsilon^2 k!} = O(\frac{1}{\epsilon^2}\log n)$, since k is assumed to be a constant. From Lemma 8, it follows that for a coloring j, $\mathcal{D}^h_t(j)$ can be updated in time $O(h^2)$ and thus the number of colorful embeddings $\tau_j(G_t)$ can be computed in time $O(h^2)$. Therefore, the overall time to compute $\hat{\phi}_t$ given G_{t-1} and $\mathcal{D}^h_{t-1}(j)$, $j = 1, \ldots, R$ is $O\left(\frac{h^2}{\epsilon^2}\log n\right)$. From Section III-A, we have $\Pr[|\hat{\phi}_t - \phi_t| \geqslant \epsilon] \leqslant 2\delta = \frac{1}{n^2}$ for a given t. Therefore, for each $t = 1, \ldots, O(n)$, $\hat{\phi}_t$ are within a factor of $1 \pm \epsilon$ with probability at least $1 - 1/n$. ■

B. The update time when $d > 2$

We now discuss how to maintain the data structure $\mathcal{C}$ dynamically when $d > 2$. We illustrate some of the counts in $\mathcal{C}$ through the example in Fig. 8, restricted to a specific coloring. In this example, edge (u, v) is added; $C(v, T'_1, \{1, 3\})$ changes from 2 to 3 (after the addition of e_t), whereas $C(v, T'_1, \{2, 3\}) = 1$ and $C(v, T'_1, \{3, 4\}) = 1$ remains unchanged. The steps for updating data structure $\mathcal{C}$ are described in Fig. 9.

Updating $\mathcal{C}$ when edge $e_t = (u, v)$ is added/deleted. Let T' be a depth $d' \geq 1$ tree. Recall the notion of $G^\beta(u, v)$ from Section III. The procedure used for updating $\mathcal{C}$ is based on the algorithm in Fig. 3. Also, recall that $\mathcal{S}(T')$ is the collection of subtrees of T' obtained by inductively decomposing it using the method in Fig. 2(b). For each random coloring, we perform the following recursive step:

1) Consider each rooted subtree $T'' \in \mathcal{S}(T')$ and let d'' denote its depth. If $|T''(r'')| > 1$, then, let $T''_1(r'')$ and $T''_2(\hat{r}'')$ be its two child subtrees in $\mathcal{S}(T')$. For each $x \in V(G^{d''-1})$, we update $C(x, T''(r''), S'')$ as follows: $C(x, T''(r''), S'') = \sum_{y \in N(x)} \sum_{S''_1, S''_2} C(x, T''_1(r''), S''_1) C(y, T''_2(\hat{r}''), S''_2)$, where, $S''_1 \cup S''_2 = S''$ and $S''_1 \cap S''_2 = \varnothing$.

Fig. 9. The data structure $\mathcal{C}$ for maintaining colored counts for trees with depth greater than 1.

Lemma 10. *For a tree T' of depth d', the update process in Fig. 9 takes $O\left(k2^{k-2}|E\left(G^{d'}(u, v)\right)|\right)$ time per random coloring.*

Proof: We focus on one coloring. Let $T''(r'') \in \mathcal{S}(T)$ be a tree of depth d''. We first observe that the addition/deletion of (u, v) affects only those $C_i(x, T''(r''), S'')$ which satisfy the following properties: (a) x should be at distance at most $d'' - 1$ from u or v and (b) S'' should contain both $\text{col}(u)$ and $\text{col}(v)$.

Now we bound the time taken to compute $C(x, T''(r''), S'')$ for all possible x and color subsets S''. The number of such tuples which need to be updated is at most $O(k2^{k-2}|V(G^{d''-1}(u, v))|)$. The recurrence for $C_i(x, T'(r'), S)$ in the counting step in Fig. 9 involves $O(N(x))$ steps, which gives a total contribution of $O(k2^{k-2}|E(G^{d'}(u, v))|)$. ■

We use the implementation of the data structure $\mathcal{C}$ from Fig. 9 within the update algorithm for $\mathcal{D}^h_t$ in Fig. 6. Lemma 7 implies that this correctly computes $\hat{\phi}_t(T)$ in each graph G_t. It also follows from the proof of Lemma 10 that $|\mathcal{A}^i(S_i)| \leq |E(G^{d-1}_{s-1})|$. Following the same analysis as in the proof of Lemma 8, we get a bound on the update time for $\mathcal{D}^h_t$ for $T \in \mathcal{T}(d, \ell)$, which is summarized below.

Theorem 11. *Let $\mathcal{G} = G_0, \ldots, G_t, \ldots$ be a dynamic subgraph sequence defined on a vertex set V such that G_{s-1} differs from G_s by exactly one edge e_s. Let $T \in \mathcal{T}(d, 2)$ and $|T| = k$. For any $s > 0, s = O(n)$, and for any $\epsilon > 0$, the number of embeddings $\phi_s(T)$, can be approximated within a factor of $1 \pm \epsilon$ with probability at least $1 - 1/n$ in time $O\left(2^k e^k \log n \frac{1}{\epsilon^2} h^2 |E(G^{d-1}_{s-1})|\right)$, using the data structure $\mathcal{D}_{s-1}$ from Fig. 9.*

Note that the bound in Theorem 11 only reflects the update time. The time to initialize the data structure $\mathcal{D}_0$ is $O\left(2^k e^k \log n \frac{1}{\epsilon^2} m\right)$, following [10].

V. Extension to general trees and constant size arbitrary subgraphs

The approach in Section IV-B actually extends to general trees $T \in \mathcal{T}(d, \ell)$, with $\ell > 2$. From Lemma 10, it follows that all the counts in the data structure $\mathcal{C}$ can be updated in time $O\left(k2^{k-2}|E\left(G^d(u, v)\right)|\right)$, if $T \in \mathcal{T}(d, \ell)$. Therefore, $\phi_s(T)$ can be computed in time $O\left(2^k e^k \log n \frac{1}{\epsilon^2}|E(G^d_{s-1})|\right)$, following the same idea as in Theorem 11.

We now consider the case of an arbitrary subgraph H, which is not a tree. We derive bounds in terms of its treewidth (denoted by $\alpha = \alpha(H)$), which is defined below. We note that $\alpha(H)$ is always bounded by $|V(H)|$, but is typically much smaller, e.g., if H is a cycle, $\alpha(H) = 2$.

Tree decomposition and treewidth [28]: A tree decomposition of a graph $H = (V(H), E(H))$ is defined by a tree $T_H = (X_H, I_H)$, where: (i) X_H is a collection of subsets of $V(H)$ such that $\cup_{V' \in X_H} V' = V(H)$, (ii) the set E_H defines a tree on V_H, (iii) for each edge $(u, v) \in E(H)$, there exists $V' \in X_H$ such that $u, v \in V'$, and (iv) for all $u \in V(H)$, the set of nodes in X_H containing u forms a connected component in T_H. The treewidth, $\alpha(H)$ of H is defined as $\max_{V' \in X_H} |V'| - 1$, so that if H is a tree $\alpha(H) = 1$.

Our algorithm follows the general structure from Section IV-B for trees, with changes in the data structures since the nodes of T_H are sets of vertices. Following Section III-A2, we will decompose the tree T_H into rooted subtrees $\mathcal{S}$. For $T' \in \mathcal{S}$, we use $\text{root}(T') \in X_H$ to be the assigned root for

T'; note that root(T') is a subset of nodes (instead of a single node, as in the case of trees). Further, we assume the nodes in each $r \in X_H$ are ordered. For $T' \in \mathcal{S}$, we use $V_H(T')$ to be the union of the nodes (which are subsets of nodes of H) in T', and let $E(T')$ denote the set of edges of H in the graph $H[V_H(T')]$ induced by $V(T')$. We use d to denote the radius of T_H.

We now need to generalize the information maintained in the dynamic program. As in Section IV, we keep colored instances G_i of the graph G. For a tuple $A = (v_1, \ldots, v_\alpha)$, and tree $T' \in \mathcal{S}$, let $C_i(A, T', S)$ denote number of colorful embeddings of the graph $H[V_H(T')]$ in G_i, in which the vertices in A are mapped to the corresponding vertices in root(T'), using the set S of colors. For tuple $A = (v_1, \ldots, v_\alpha)$, let $N(A)$ denote the set of all tuples $(v'_1, \ldots, v'_\alpha)$ such that $v'_i \in N(v_i)$. For a tree $T' \in \mathcal{S}$, let T'_1, T'_2 be its two child subtrees obtained by splitting it in $\mathcal{S}$. To simplify the notation, we only consider a fixed colored instance G_i, and drop the subscript i. The quantities $C(A, T', S)$ can be shown to satisfy the following recurrence: $C(A, T', S) = \sum_{A' \in N(A)} \sum_{S_1, S_2} C(A, T'_1, S_1) C(A', T'_2, S_2)$, where the sum is over all subsets $S_1 \cup S_2 = S$, such that $|S_i|$ equals the number of vertices in $H[V_H(T'_i)]$. Following the analysis of [10], it can be shown that the dynamic program can be adapted to keep track of all these tuples by solving the recurrence repeatedly in time $O(2^k e^k |E| n^\alpha)$. We note that we do not need to keep track of the counts $C(A, T', S)$ for all α-tuples, but only those in which the vertices in A are within pairwise distance at most the diameter of H, which can be seen to be at most $2d$.

We briefly sketch the main changes that have to be made for the dynamic case. Suppose edge $e = (u, v)$ is added to the graph $G = G_t$. The algorithm of Section IV is adapted to update the counts $C(A, T', S)$ for all tuples $A = (v_1, \ldots, v_\alpha)$ and trees $T' \in \mathcal{T}$ such that: there exists a vertex $v_j \in A$ that is within distance d of u or v in H. For any such tuple A, if v_j is within distance d from u or v, it follows that all the remaining vertices $v_{j'}$ are within distance at most $3d$ of u or v. Therefore, the number of such tuples A which need to be updated is at most $O(k|V(G_{3d})|^\alpha 2^k)$, where G_{3d} denotes the subgraph of G induced by vertices within distance at most $3d$ of u or v. The performance is summarized below.

Lemma 12. *Suppose H is a graph with treewidth α, and has a tree decomposition T_H of radius d. The* DSE$(\mathcal{G}, H)$ *can be solved with $O(k|V(G_{3d})|^\alpha 2^k)$ time per update.*

VI. Empirical Results

We now discuss the empirical performance of our algorithms by analyzing it on 14 different real networks selected from [20] (these are summarized in Table I). We focus on update time and space requirements for the algorithm in Fig. 6 for $d = 2$ (Section IV-A) as well as the algorithm based on the dynamic adaptation of color-coding in Fig. 9 (Section IV-B); this comparison allows us to understand the specific efficiency resulting from the use of the h-index data structure. We consider three kinds of updates to the graph: (1) **Add:** at each step an edge is added randomly, (2) **Delete:** at each step an existing edge is deleted randomly and (3) **Mixed:** with probability $1/2$, an edge is added or deleted randomly. Our results are summarized below.

1. Update time. Fig. 10 shows the ratio of the average update time to the static time, for paths of varying lengths and a complete binary tree in $\mathcal{T}(2, 2)$ on all the networks in Table I. The results show that the dynamic update time for paths is hardly ever more than 10% of the static enumeration time for random updates, and often around 1%. For the tree in $\mathcal{T}(2, 2)$, the ratios are quite variable, with a few cases with up to 50%; this happens more often with the smaller graphs, and the ratios are generally much smaller for the larger graphs. Fig. 11 shows the update time in seconds for the add, delete and mixed operations for paths of lengths ranging from 3 to 9 in three of the networks. An interesting observation is that the delete operations take more time than add and mixed type updates. Also, for paths (Fig. 11), the run time scales with 2^k, where k is the path length, which is expected from our analysis. Figure 12 shows a comparison of the algorithm using the h-index data structure (Fig. 6) with the direct color-coding based algorithm (Figure 9); observe that the use of the h-index data structure gives significant improvements in most graphs.

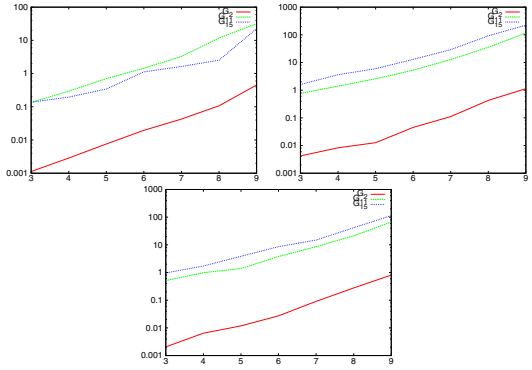

Fig. 11. Average time in seconds (y-axis) for dynamic update for fixed graphs with path motifs of varying lengths (x-axis): G_2 (red), G_{11} (green), G_{15} (blue), for (a) only add operations, (b) only delete operations and (c) both operations.

2. Space complexity. Fig. 13 shows the memory usage for different graphs, for paths of length 5 and 9. We measured the maximum resident set size of our implementation for only the initial static enumeration step. This initial step captures the size of all the relevant data structures, and the dynamic updates modify the memory used by a negligible amount. For the 9-path instance, the maximum memory usage is about 6GB. Fig. 14 shows the memory used for fixed graphs but with subgraphs of size ranging from 3 to 9. The memory usage in the figure appears to be bounded by an exponential in the path length.

Graph	Number of vertices	Number of edges	h-index	Max. deg. (Δ)	Avg. deg.
as20000102.txt (G_1)	6474	13233	49	1460	4.09
ca-grqc.giant.uel (G_2)	4158	13422	45	81	6.46
oregon1-010331.txt (G_3)	10670	22002	65	2312	4.12
ca-hepth.giant.uel (G_4)	8638	24806	39	65	5.74
oregon2-010331.txt (G_5)	10900	31180	86	2343	5.72
p2p-Gnutella04.txt (G_6)	10876	39994	43	103	7.35
tweet.uel (G_7)	22405	59925	111	888	5.35
p2p-Gnutella24.txt (G_8)	26518	65369	33	355	4.93
ca-condmat.giant.uel (G_9)	21363	91286	76	279	8.55
regular-20.uel (G_{10})	10000	100000	20	20	20.00
wiki.giant.uel (G_{11})	7066	103663	187	1065	29.34
ca-astroph.giant.uel (G_{12})	17903	196972	151	504	22.00
cit-HepTh.txt (G_{13})	27770	352807	192	2468	25.41
cit-hepph.giant.uel (G_{14})	34401	420783	179	846	24.46
epin.giant.uel (G_{15})	75877	508836	40	19816	13.41

TABLE I
NETWORKS STUDIED AND THEIR CHARACTERISTICS. WE USE THE NOTATION $G_1, \ldots, G_{15}$ TO REFER TO THEM IN OUR PLOTS.

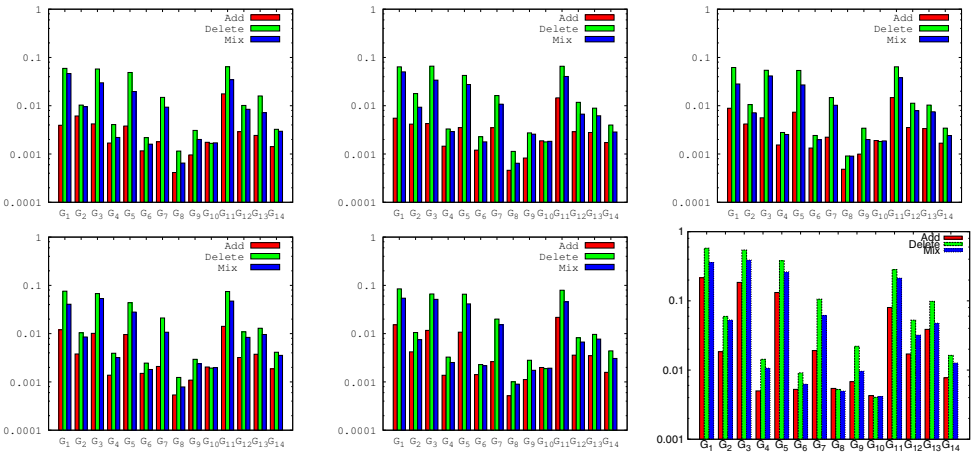

Fig. 10. Ratio of the dynamic update time to the static enumeration time in the various graphs $G_1, \ldots, G_{14}$ in Table I for paths of length 5, 6, 7, 8, 9 and a tree in $T(2, 2)$, respectively. In each plot, the graphs are ordered on the x-axis in increasing order of size. The red, green and blue bars represent the times for the add/delete/mixed types of updates, respectively.

VII. CONCLUSIONS

Dynamic subgraph enumeration problems arise commonly in numerous data mining applications. In this paper we develop the first provable algorithms for dynamic subgraph enumeration of subgraphs H of size more than 4. Given a dynamic sequence of graphs $\mathcal{G} = G_0, G_1, \cdots$, our algorithm maintains a data structure $\mathcal{D}_t$ corresponding to each G_t, so that the update time for computing number of embeddings in G_{t+1} is sub-linear, and significantly smaller than the best static algorithms in most settings. This is borne out in our empirical evaluations, which show that the update time is within a fraction of 10% of the static time. The main focus of this paper is theoretical, and an important contribution of this paper is the integration of two techniques: the color coding technique, and the dynamic h-index structure of the graph, which we expect would be useful in many other problems.

Acknowledgment We are grateful to the reviewers whose comments have helped improve the paper. This work has been partially supported by the following grants: DTRA Grant HDTRA1-11-1-0016, DTRA CNIMS Contract HDTRA1-11-D-0016-0010, NSF Career CNS 0845700, NSF ICES CCF-1216000, NSF NETSE Grant CNS-1011769 and DOE DE-SC0003957. Also supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D12PC000337, the US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.
Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the US Government.

REFERENCES

[1] M. Bröcheler, A. Pugliese, and V. Subrahmanian, "Cosi: Cloud oriented subgraph identification in massive social networks," in *ASONAM*, 2010,

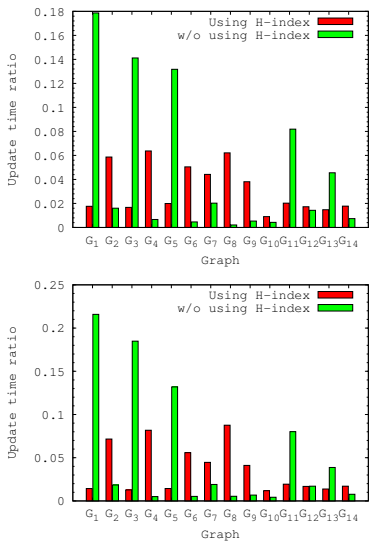

Fig. 12. Ratio of dynamic update time to the static enumeration time for the various graphs $G_1, \ldots, G_{14}$ in Table I for (a) a binary tree in $T(2, 2)$, and (b) a path of length 5. Two algorithms are compared here– the direct dynamic adaptation of color-coding in Fig. 9 (labeled "w/o using H-index") and the one in Fig. 6 that uses the h-index data structure (labeled "using H-index").

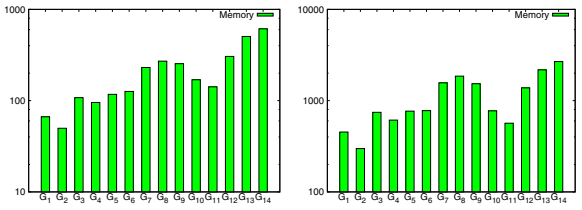

Fig. 13. Memory usage in MB for various graphs in Table I, ordered on the x-axis in increasing order of size for paths of length (a) 5 and (b) 9.

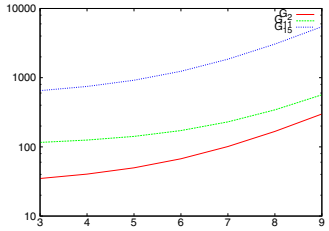

Fig. 14. Memory usage (in MB) for fixed graphs (y-axis) with path motifs of varying size (with length on the x-axis): G_2 (red), G_{11} (green) and G_{15} (blue).

pp. 248–255.
[2] E. Bloedorn, N. J. Rothleder, D. DeBarr, and L. Rosen, "Relational graph analysis with real-world constraints: An application in IRS Tax Fraud Detection," in *AAAI*, 2005.
[3] C. Borgelt and M. R. Berhold, "Mining molecular fragments: Finding relevant substructures of molecules," in *ICDM*, 2002.
[4] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, p. 824, 2002.
[5] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *ECML-PKDD*, 2000.
[6] M. Kuramochi and G. Karypis, "Finding frequent patterns in a large sparse graph," *Data mining and knowledge discovery*, vol. 11, no. 3, pp. 243–271, 2005.
[7] Z. Zhao, G. Wang, A. R. Butt, M. Khan, V. S. A. Kumar, and M. V. Marathe, "SAHAD: Subgraph analysis in massive networks using hadoop," in *IPDPS*, 2012.
[8] R. Williams, "Finding paths of length k in $o^*(2^k)$ time," *Inf. Process. Lett.*, vol. 109(6), 2009.
[9] V. Vassilevska and R. Williams, "Finding, minimizing, and counting weighted subgraphs," in *ACM STOC*, 2009.
[10] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. Sahinalp, "Biomolecular network motif counting and discovery by color coding," *Bioinformatics*, vol. 24, no. 13, pp. i241–i249, 2008.
[11] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu, "Graphscope: parameter-free mining of large time-evolving graphs," in *KDD*, 2007, pp. 687–696.
[12] H. Tong, S. Papadimitriou, J. Sun, P. S. Yu, and C. Faloutsos, "Colibri: fast mining of large static and dynamic graphs," in *KDD*, 2008.
[13] T. A. B. Snijders, "Markov chain monte carlo estimation of exponential random graph models," *Journal of Social Structure*, vol. 3(2), pp. 1–40, 2002.
[14] S. Wasserman and P. Pattison, "Logit models and logistic regressions for social networks: I. An introduction to Markov graphs and p^*," *Psychometrika*, vol. 61, no. 3, pp. 401–425, 1996.
[15] D. Eppstein, M. T. Goodrich, D. Strash, and L. Trott, "Extended h-index parameterized data structures for computing dynamic subgraph statistics," in *Proc. COCOA*, 2010.
[16] D. Eppstein and E. S. Spiro, "The h-index of a graph and its application to dynamic subgraph statistics," in *WADS*, 2009.
[17] D. Kane, K. Mehlhorn, T. Sauerwald, and H. Sun, "Counting arbitrary subgraphs in data streams," in *ICALP*, 2012.
[18] M. Manjunath, K. Mehlhorn, K. Panagiotou, and H. Sun, "Approximate counting of cycles in streams," in *Proc. 19th European Symp. on Algorithms (ESA)*, 2011.
[19] J. E. Hirsch, "An index to quantify an individuals scientific research output," *Proc. National Academy of Sciences*, vol. 102(46), 2005.
[20] "Stanford network analysis project," http://snap.stanford.edu/index.html, 2011.
[21] A. Adiga, A. Vullikanti, and D. Wiggins, "Subgraph enumeration in dynamic graphs," http://ndssl.vbi.vt.edu/supplementary-info/vskumar/dynamic.pdf, 2013.
[22] V. Aravind and V. Raman, "Approximate counting of small subgraphs of bounded treewidth and related problems," *ECCC*, 2002.
[23] F. Eisenbrand and F. Grandoni, "On the complexity of fixed parameter clique and dominating set," *Theoretical Computer Science*, vol. 326, 2004.
[24] M. Kowaluk, A. Lingas, and E. Lundell, "Counting and detecting small subgraphs via equations and matrix multiplication," in *ACM SODA*, 2011.
[25] I. Koutis, "Faster algebraic algorithms for path and packing problems," in *Proc. ICALP*, 2008.
[26] F. Hüffner, S. Wernicke, and T. Zichner, "Algorithm engineering for color-coding with applications to signaling pathway detection," *Algorithmica*, vol. 52, no. 2, pp. 114–132, 2008.
[27] Z. Zhao, M. Khan, V. S. A. Kumar, and M. Marathe, "Subgraph enumeration in large social contact networks using parallel color coding and streaming," in *ICPP*, 2010, pp. 594–603.
[28] H. Bodlaender, "Treewidth: Algorithmic techniques and results," in *In Mathematical foundations of computer science*. Springer, 1998, p. pages.