Subgraph Enumeration in Large Social Contact Networks using Parallel Color Coding and Streaming

Zhao Zhao, Maleq Khan, V.S. Anil Kumar and Madhav V. Marathe
Network Dynamics and Simulation Science Laboratory
Virginia Tech

***Abstract*—Identifying motifs (or commonly occurring subgraphs/templates) has been found to be useful in a number of applications, such as biological and social networks; they have been used to identify building blocks and functional properties, as well as to characterize the underlying networks. Enumerating subgraphs is a challenging computational problem, and all prior results have considered networks with a few thousand nodes. In this paper, we develop a parallel subgraph enumeration algorithm, PARSE, that scales to networks with millions of nodes. Our algorithm is a randomized approximation scheme, that estimates the subgraph frequency to any desired level of accuracy, and allows enumeration of a class of motifs that extends those considered in prior work. Our approach is based on parallelization of an approach called color coding, combined with a stream based partitioning. We also show that PARSE scales well with the number of processors, over a large range.**

I. INTRODUCTION

The problem of enumerating or counting the number of embeddings of a given motif/subgraph T (referred to as a "template" graph in this paper), within a graph G is a fundamental question that arises in a number of areas, including bioinformatics [1; 12], data mining [9; 10] and social networks [11]. In biological applications, such as gene transcription and protein-protein networks, small motifs, whose frequency is significantly different from random networks, have been found to be very useful in characterizing such graphs (e.g., Milo et al. [12]); it has been found that graphs which have similar local properties (e.g., degree and centrality distributions) can be distinguished by means of relative frequencies of such subgraphs. Another related notion is that of *frequent subgraphs* [9], which have been found to have functional significance in such graphs. Lescovec et al. [11] study subgraphs in Blog networks and recommendation systems, and use these to develop insights into user behavior. Two variants which have been studied are the enumeration of *induced* and *non-induced* embeddings of the template subgraph (see Section II for definitions).

A significant challenge in subgraph enumeration is its computational difficulty, and most of the above results (especially in the case of non-induced embeddings) have been restricted to graphs with a few thousand nodes and templates with no more than ten nodes; most templates which have been studied are trees and paths. This has motivated *approximate* counting [1; 6; 8], which is also computationally hard. Many of the approaches for subgraph enumeration are back-tracking algorithms, with novel techniques for careful candidate generation and subgraph isomorphism, e.g., [9; 13]. These approaches are not easy to parallelize because of the type of information that needs to be maintained during the backtracking, with the exception of clique enumeration [13], where there has been quite a bit of work on parallel algorithms, that have crucially exploited the structure of cliques, and is difficult to extend to other templates. Another class of results for subgraph enumeration is based on the powerful technique of *color coding* (which also forms the basis of our paper), e.g., [1; 8], which has been used for approximating the number of embeddings of templates that are trees or "tree-like" (or more precisely, have a low "tree-width" - see II for details). The idea of color coding is to randomly color the graph using k colors (where k denotes the size of the template), and then count the number of embeddings of "colorful" subgraphs - these are embeddings in which all nodes have distinct colors. The power of this technique comes from the fact that the number of colorful embeddings can be determined by a natural dynamic programming approach. However, the dynamic programming has high memory requirements, and this approach has only been used on small graphs with few hundred nodes. Therefore, none of the current techniques for subgraph enumeration scale to social networks with millions of nodes, which sometimes do not even fit in memory.

In this paper, we present a randomized parallel algorithm, PARSE (Parallel Subgraph Enumeration) for enumerating a class of templates T which can be partitioned into sub-templates $T_1 \cup T_2$ by a cut-edge (see Section II for precise definitions); this is broader than most of the templates considered in prior sequential algorithms, e.g., [1; 8], and parallel algorithms [14]. Our algorithm uses color coding in combination with a stream-based cover decomposition. The stream-based approach addresses the space complexity issues of the dynamic programming in the sequential form of color coding. We test the performance of PARSE on graphs with up to a few million nodes and hundred million edges and templates with up to 10 nodes; our implementation shows good scaling up to 350 processors, and runs in a few hours. We also derive analytical bounds on the performance, which yields explicit bounds on the extent to which the scaling holds. All prior results have been for graphs which are at least two orders of magnitude smaller.

ParSE is a randomized (ϵ, δ)-approximation scheme, which estimates the number of non-induced occurrences within a factor of $(1 \pm \epsilon)$, with probability at least $1 - \delta$, where ϵ and δ are user defined parameters. The algorithm averages the count over multiple iterations, and the number of iterations (and hence, the running time) depends on ϵ and δ. In practice, we find that the approximation error is very low (less than 0.1%) with high probability (more than 0.95), within a small number of iterations (less than 20). The algorithm involves a two-step approach: (i) we split the template T into multiple parts $T_1, \ldots, T_j$, and compute the number of "colorful" embeddings of the parts T_i within each partition (see Section II for precise definitions), and (ii) we use color coding to exchange information about the counts of the parts T_i (along with the specific colorings) - this corresponds to one step of the dynamic programming approach of [1; 8]. Together, this provides a systematic way to deal with the high memory requirement and irregular graph structure, which are among the biggest challenges for subgraph enumeration. In addition, we also discuss how we can count templates with small radius (see Section IV-E) that does not require color coding and still has less memory requirement.

Organization. We discuss definitions and the necessary background in Section II. Section II-A describes the Color-coding approach. Section III describes our main algorithm and results on some large social contact networks. Section IV validates our algorithm from many aspects through experiments. Section V gives some specifications in implementation of the algorithm. Finally, Section VII concludes the paper.

II. Background and Definitions

We consider the problem of counting the number of non-induced subgraphs of an undirected graph $G(V, E)$, which are isomorphic to a given template $T(V^T, E^T)$; we let $|V| = n$ and the templates we consider are small and have constant size. A subgraph $H = (V', E')$ is said to be isomorphic to the template T if there is a bijection $f : V^T \rightarrow V'$ such that if $(u, v) \in E^T$ then $(f(u), f(v)) \in E'$; such a subgraph H is said to be a *non-induced* occurrence of T (in contrast to an *induced* occurrence, in which $(u, v) \in E^T$ if and only if $(f(u), f(v)) \in E'$). These concepts are illustrated in Figure 1, which shows a template T with $V^T = \{1, \ldots, 5\}$ and graph G with $V = \{a, \ldots, k\}$. Two subgraphs H_1 and H_2, isomorphic to T are shown; the specific bijections which imply isomorphism are also shown. H_1 is an induced occurrence, while H_2 is a non-induced occurrence of T. Let $emb(T, G)$ denote the number of non-induced occurrences of template T in the graph G.

The focus of our paper is to estimate $emb(T, G)$ for given T and G. An additional issue is that depending on the symmetries in the template, there might be more than one bijection between the template and a subgraph (in other words, an automorphism on the template); for instance, in addition to the bijection f shown in Figure 1, another bijection f' is also the following: $f'(1) = a$, $f'(2) = b$, $f'(3) = c$, $f'(4) = e$

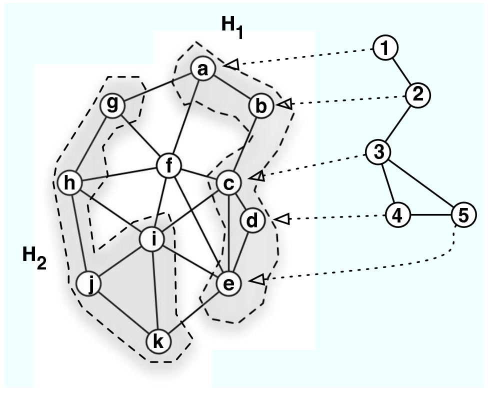

Fig. 1. Example showing a graph G with $V = \{a, \ldots, k\}$ and template T with $V^T = \{1, \ldots, 5\}$. H_1 and H_2 are two subgraphs of G that are isomorphic to T. For H_1, the bijection is $f(1) = a$, $f(2) = b$, $f(3) = c$, $f(4) = d$ and $f(5) = e$, as shown by the dotted arrows. For H_2, the bijection is $f(1) = g$, $f(2) = h$, $f(3) = j$, $f(4) = k$ and $f(5) = i$. H_1 is an induced occurrence, while H_2 is a non-induced occurrence because of the presence of the edge (h, i).

and $f'(5) = d$. Therefore, in order to get a correct count, we will have to divide the number of bijections by the number of automorphisms of T. We focus on randomized algorithms to estimate $emb(T, G)$, instead of computing it exactly. We say that an algorithm $\mathcal{A}$ produces an (ϵ, δ)-approximation to $emb(T, G)$, if the estimate Z produced by $\mathcal{A}$ satisfies: $\Pr[|Z - emb(T, G)| > \epsilon \cdot emb(T, G)] \leq \delta$; in other words, $\mathcal{A}$ is required to produce an estimate that is close to $emb(T, G)$, with high probability.

We introduce some additional notation. $N(v)$ denotes the set of all neighbors of node v. We use $d_H(u, v)$ to denote the shortest path distance between nodes u and v in graph H; when graph H is clear from the context, we simply denote this by $d(u, v)$. The radius of graph H, denoted $rad(H)$, is defined as $rad(H) = \min_{u \in V(H)} \max_{v \in V(H)} \{d_H(u, v)\}$.

A. Color Coding

We briefly discuss the main idea of color coding for the problem of enumeration of paths in G; see [1] for additional details on how to use this for trees and other kinds of subgraphs. The algorithm involves the following steps:

1) For $i = 1$ to $N = O(\frac{e^k \log 1/\delta}{\epsilon^2})$ perform the following steps, where $k = |V^T|$ is the number of nodes in the template, and δ and ϵ are input parameter such that the approximation factor is $1 \pm \epsilon$ with probability at least $1 - \delta$:
   a) (Color coding) Color each node $v \in V(G)$ uniformly at random with a color from $\{1, \ldots, k\}$.
   b) (Counting) Use a dynamic program to count all the "colorful" embeddings of T in G, where an embedding H in G isomorphic to T is said to be colorful (see Figure 2 and the discussion below), if all the nodes in H have distinct colors. Let X_i be the number of embeddings in iteration i.
2) Output an estimate of the actual number of embeddings using the X_i's in the following manner: partition the

N samples above into $t = O(\log 1/\delta)$ sets, and let Y_j be the average of the j set. Output the median Z of $Y_1, \dots, Y_t$.

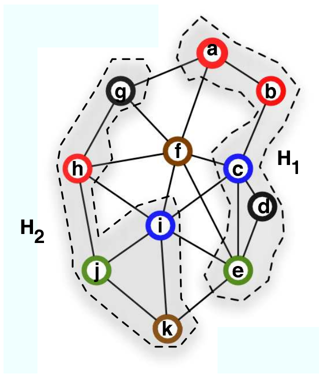

Fig. 2. Example showing the notion of "colorful" embeddings of template T (as in Figure 1) in a coloring of graph G. The nodes of G are colored with $k = 5$ colors (red, blue, black, brown and green). As in Figure 1, both H_1 and H_2 are occurrences of T, but only H_2 is a colorful occurrence.

Figure 2 shows the notion of colorful embeddings. Let $\phi : V \rightarrow \{1, \dots, k\}$ be a k-coloring of the nodes of G. Formally, we say that a subgraph $H = (V', E')$ is a colorful embedding of template T if H is isomorphic to T and all nodes in V' have distinct colors, with respect to $\phi()$. Alon et al. [1] show that the above (randomized) algorithm gives an estimate of the actual number of embeddings of T in G within a multiplicative factor of $(1 \pm \epsilon)$, with probability at least $1 - \delta$, where ϵ and δ are arbitrary input parameters. The main ideas of this algorithm are: (i) the probability that any given embedding H of T with k nodes is colorful is precisely $\frac{k!}{k^k}$, so that the expected number of colorful embeddings is $emb(T, G)\frac{k!}{k^k}$, and (ii) because we are considering colorful paths, there is a simple recurrence relationship for the number of colorful paths, $C(v, S)$ of length $|S|$, with end point v, and using the set S of colors:

$$C(v, S) = \sum_{u \in N(v)} C(u, S \setminus \{\text{color}(v)\})$$

Extension to more general (non-path) templates. This approach has been extended to trees and treewidth bounded graphs [1; 8], with faster algorithms for special kinds of subgraphs [6; 7]. In this paper, we further extend it to a a slightly more general class of templates T, which can be partitioned into sub-templates T_1 and T_2 by the deletion of a cut-edge; a cut-edge in a graph is an edge whose deletion increases the number of connected components. Trees certainly satisfy this constraint, but as shown in Figure 3, template T (the graph at the right hand side), which is not a tree, still satisfies this constraint, since it can be partitioned into sub-templates T_1 and T_2 by removing the cut edge $(2, 3)$.

Now we illustrate the main idea behind our parallel algorithm PARSE using Figure 3. A node adjacent to the cut edge is called the root of the corresponding sub-template, and the root of template T_i is denoted by $\rho(T_i)$. In the figure, template T is partitioned into two subgraphs T_1 and T_2, with roots $\rho(T_1) = 2$ and $\rho(T_2) = 3$, respectively. Let $C(v, \rho(T_i), T_i, S_i)$ denote the number of colorful embeddings of subgraph T_i with root $\rho(T_i)$ mapped to node $v \in V$, and using the colors in S_i, where $|S_i| = |T_i|$. The number of colorful embeddings of T with nodes g and f in G mapped to nodes 2 and 3 of T, respectively, is equal to $\sum_{S_1 \cup S_2 = S} C(g, 2, T_1, S_1)C(f, 3, T_2, S_2)$, where the sum is over all partitions of $S = \{\text{brown, blue, green, black, red}\}$. There are two colorful embeddings of T_2 with root node 3 mapped to node f, and using the colors $S_2 = \{\text{brown, blue, green}\}$, i.e., $C(f, 3, T_2, S_2 = \{\text{brown, blue, green}\}) = 2$. Similarly, we have $C(g, 2, T_1, S_1 = \{\text{black, red}\}) = 2$, as shown in the figure. For all other choices of S_1, we have $C(g, 2, T_1, S_1) = 0$. Therefore, we have $\sum_{S_1 \cup S_2 = S} C(g, 2, T_1, S_1)C(f, 3, T_2, S_2) = 4$.

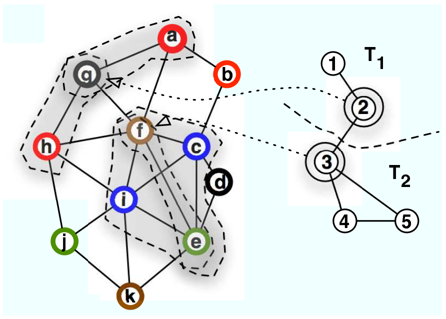

Fig. 3. Illustration of the dynamic programming step of color coding using the examples given in Figures 1 and 2. Template T is partitioned into two subgraphs T_1 and T_2, with roots 2 and 3, respectively. We have $C(g, 2, T_1, S_1 = \{\text{black, red}\}) = 2$ and $C(f, 3, T_2, S_2 = \{\text{brown, blue, green}\}) = 2$.

B. Datasets

Our main focus in this paper is on social contact networks, in which a link represents physical contact between two persons. Since realistic data for such networks is difficult to obtain, we use the approach of Barrett et al. [3] for constructing large scale synthetic social contact networks for urban regions. Specifically, we use the contact networks constructed for the cities of New River Valley (NRV), Miami, and Chicago; the sizes of the these graphs are given in Table I. As discussed in [3], these networks are very unstructured and different from various kinds of random graph models.

TABLE I
SIZES OF THE GRAPHS USED IN THE EXPERIMENTS

Graph	Number of Nodes	Average Degree
NRV	151,783	163.5
Miami	2,092,147	50.4
Chicago	9,038,414	59.5
Miami10	20,921,470	50.4
GNP50	50,000	20
GNP100	100,000	20

In order to test our framework with even larger graphs, we construct an artificial network, Miami10, obtained by composing 10 copies of the Miami network in the following manner: we sample an integer d from a distribution D for each vertex in the new network. Then we choose d vertices from clones of Miami other than the one containing this vertex uniformly at random, and connect them with this vertex. We also generate two random graph using $G(n,p)$ model, which contains n vertices and each pair of vertices has equal probability p to be connected by an edge. The two random graph are named GNP100 and GNP50.

C. *Notations used in this paper*

Table II lists some of the notations commonly used in the rest of the paper.

TABLE II
NOTATIONS IN THE PAPER

Symbol	Definition
G	The main graph
G_p	A partition of G
T	Template graph
p, P	Partition ID, number of partitions
q, Q	Processor ID, number of processors
n, m	Number of vertices and edges in G, respectively
$N(v)$	The set of neighbor of vertex v
$N_r(v)$	r-Neighbor of vertex v
k	Size (number of vertices) of template T
S	Color/Label set
α, β	Factors for counting colorful sub-template, and template

III. ALGORITHM FOR PARALLEL TEMPLATE COUNTING

Now we describe an overview and the basic ideas followed by the complete description of our algorithm PARSE. As discussed earlier, in this paper, we only consider templates T which can be partitioned into sub-templates T_1 and T_2 by removing a cut-edge. In our dynamic program based parallel algorithm for counting all colorful embeddings of T in G, the main idea is to have the color coding counts $C(v, \rho(T_i), T_i, S_i)$ (as defined in Section II-A) exchanged by the processors and combine these counts carefully. While the algorithms of [1; 8] compute the above counts $C(v, \rho(T_i), T_i, S_i)$ as well by dynamic programming (which limits the approach to trees), we compute $C(v, \rho(T_i), T_i, S_i)$ locally within a processor, using a back-tracking based approach, and all processors send these counts to a designated processor called the master processor or master node, which aggregates them. As a result, PARSE is able to handle more complex templates than trees, and the template can have more than constant treewidth.

A. *Overview of* PARSE

Algorithm 1 gives a high level description of PARSE, and Fig. 4 shows a schematic diagram; some of the terms used here are defined in Section III-B. Processor 0 is designated as a master processor (also referred as master node), which performs the initialization and finalization of the algorithm. The other processors are called worker nodes. Once the system is initialized, each worker node q generates its assigned partitions from the original graph G in line 1 - this is explained in Section III-B. The master node also performs the random coloring in line 3 of Algorithm 1. In line 4-6, each processor q independently works on each partition G_p assigned to it (our node labeling scheme allows the processors to identify the partitions independently), and computes the counts $C(v, \rho(T_i), S_i)$ for each i and each possible color set S_i, by invoking the Algorithm COUNTTEMPLATE from Section III-C. This corresponds to the step of computing smaller sub-problems, as part of the dynamic programming. The streaming-based workflow avoids loading the original graph into local memory, which is useful for very large graphs. Also, note that by making COUNTTEMPLATE generic, we are able to enumerate more complex templates. Finally, in lines 7-10, the remaining step of the dynamic program is run to compute the aggregate count by sending the sub-problems to the master node; this is described in detail in Section III-D.

Algorithm 1 High level description of Algorithm PARSE.

```
1:  Partition $G$ and assign processors
2:  Partition template $T$ into two parts $T_1$ and $T_2$; let $\rho(T_i)$
    denote a "root" for the subgraph $T_i$
3:  Assign each node $v \in V$ a random color from $\{1, \ldots, k\}$
4:  for each processor $q$, and each partition $G_p$ assigned to it
    do
5:      for each node $v \in core(G_p)$, each set $S_i \subset$
        $\{1, \ldots, k\}$, $|S_i| = |T_i|$, $i = 1, 2$ do
6:          Compute $C(v, \rho(T_i), T_i, S_i)$
7:  for each edge $e = (u, v) \in E$ do
8:      Compute
9:      $C(e) = \sum_{S_1, S_2} C(u, \rho(T_1), T_1, S_1) C(v, \rho(T_2), T_2, S_2)$
10:     $+C(v, \rho(T_1), T_1, S_1) C(u, \rho(T_2), T_2, S_2)$
11:     where the sum is over all $S_1 \cup S_2 = \{1, \ldots, k\}$
12: $X \leftarrow \sum_e C(e)/\beta$
13: Repeat line 3-12 until the average of $X$ reaches the
    precision requirement
```

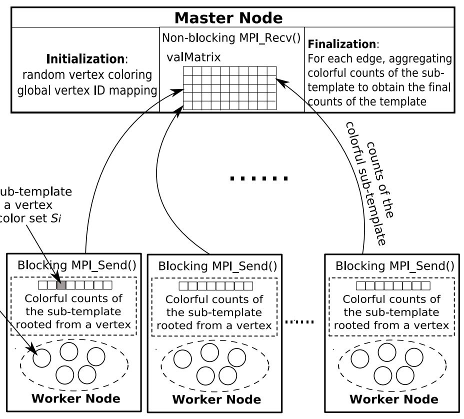
Master Node
Initialization:
random vertex coloring
global vertex ID mapping
Non-blocking MPI_Recv()
valMatrix
Finalization:
For each edge, aggregating colorful counts of the sub-template to obtain the final counts of the template
Counts of the sub-template rooted from a vertex specifying a color set Si
counts of the colorful sub-template
......
Graph partitions
Blocking MPI_Send()
Colorful counts of the sub-template rooted from a vertex
Worker Node
Blocking MPI_Send()
Colorful counts of the sub-template rooted from a vertex
Worker Node
......
Blocking MPI_Send()
Colorful counts of the sub-template rooted from a vertex
Worker Node

Fig. 4. A schematic description of PARSE.

B. *Cover-based Graph Partitioning*

Let $N_r(v)$ be the r-neighborhood of node v, which is defined as $N_r(v) = \{u : d(u,v) \leq r\}$, where $d(u,v)$ is the distance between u and v. For a subgraph $G_p = (V_p, E_p)$

of G, we define $core(G_p)$ as the set of nodes whose r-neighborhood is completely contained in G_p, i.e., $core(G_p) = \{v : N_r(v) \subset V_p\}$, for a parameter r. A "cover decomposition" is a partitioning of the graph into overlapped parts G_p so that the following properties hold [2]:

$$i)\ \bigcup_{1\leq p\leq P} core(G_p) = V$$

$$ii)\ \forall p_1 \neq p_2,\ \ core(G_{p_1}) \cap core(G_{p_2}) = \phi$$

Algorithm 2 describes GENERATEPARTITION, which partitions the graph G into parts G_p. To generate partition G_p, Algorithm 2 first reads vertex set W_p, the set of core vertices from the graph file, with size $|W_p| = n/P$ (line 1-2). Then a subgraph $G_p = (V_p, E_p)$ is computed which includes the r-neighborhood of W_p, and W_p itself, so that $core(G_p) = W_p$. The algorithm is implemented in a streaming manner and makes $r+2$ passes through the file; it does not load the entire graph G into main memory, allowing us to work with very large graphs that do not fit in memory. In the first pass (Line 1-2), GENERATEPARTITION reads core nodes W_p. In the next r passes, it determines and includes the nodes in r-neighborhood of W_p. In the last pass, it reads the edges incident on nodes in V_p.

Algorithm 2 Cover-based-partition

```
GENERATEPARTITION($G(V,E), p, r, P$)
0: $nCore \leftarrow \frac{n}{P}$
1: Skip $nCore \times (p-1)$ nodes
2: $V_p \leftarrow$ the set of next $nCore$ nodes
3: $E_p \leftarrow \phi$
4: for $i$ from 1 to $r$ do
5:     for each edges $(v,u)$ in $G$ do
6:         If $v \in V_p$ and $u \notin V_p$, $V_p \leftarrow V_p \cup \{u\}$
7: for each edges $(v,u)$ in $G$ do
8:     If $v, u \in V_p$, add $(v,u)$ to $E_p$
9: return  $G_p(V_p, E_p)$
```

Algorithm GENERATEPARTITION makes $r + 2$ passes through the graph file: Line 1-3 makes 1 pass, Line 4-6 makes r passes and Line 7-8 makes 1 pass. The graph file is stored as an adjacency list and thus the file size is $O(n+m)$ where n is the number vertices and m is the number of edges in the graph. Thus, reading the graph file to generate one partition takes $O((n+m)r)$ time. Therefore, to generate P partitions by Q processors takes $O((n+m)rP/Q)$ time as processor read the graph file independently and simultaneously in parallel. Fig. 5 shows running time for generating partitions with $r = 0, 1, 2$ on network Miami. It shows for fixed Q and r, running time increases linearly to the number of partitions P.

The hop-size r needs to be chosen carefully - it should be large enough to cover each sub-template, i.e. $r \geq rad(T_i)$, for each sub-template T_i of T; $rad(T_i)$ denotes the radius of T_i, as defined in Section II. However, the partition size, $N_r(v)$, grows rapidly with r in social networks, as shown in Fig. 6 for the contact graph for Miami; note that when

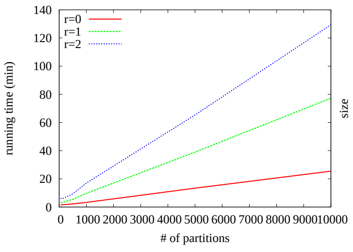

Fig. 5. Running time of partitioning vs. number of partitions, varing the cover size r.

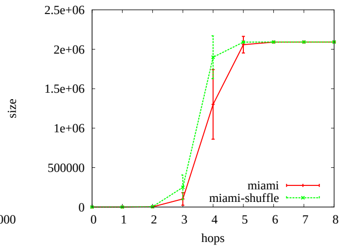

Fig. 6. Example showing how the size of $N_r(v)$ from a single vertex v increasing in our social contact graph.

$r = 5$, the r-neighborhood of a random node is almost the entire graph. This is also true in the random graph obtained by randomly switching pairs of edges of the Miami graph (denoted by Miami-shuffle in Fig. 6), while preserving the node degrees. Therefore, when the radius of the template grows, its enumeration cannot be done locally within a partition, and PARSE uses color coding to distribute the computation over sub-templates.

C. Template Enumeration

Algorithm 3 COUNTTEMPLATE describes the subroutine which determines the local counts $C(v, \rho(T_i), S_i)$ (line 6 of Algorithm 1); this is done locally within $core(G_p)$ for each part G_p. COUNTTEMPLATE uses a BFS based backtracking scheme that invokes MATCHSUBGRAPH (Algorithm 4), in order to compute these quantities.

Algorithm 3 Template enumeration in each partition

```
COUNTTEMPLATE($T(V^T, E^T), G_p(V_p, E_p), S$)
1: Perform a breadth-first search (BFS) in $T$. Let $u_1, u_2, \ldots$
   be the nodes in $T$ in the order of their discovery in BFS.
2: for each node $v \in V_p$ do
3:     for each permutation $S'$ of $S$ do
4:         if MATCHVERTEX($u_1, v, S'$) then
5:             MATCHSUBGRAPH($2, S'$)
6: $\alpha$ = number of automorphisms of $T$ with root $\rho(T)$
7: return  $count/\alpha$
```

Algorithm 4 MATCHSUBGRAPH(i, S')

```
MATCHSUBGRAPH($i, S'$)
1: if $i = |V^T| + 1$ then
2:     $count \leftarrow count + 1$
3:     return
4: Find $M = \{f(u_k) \mid k < i \wedge (u_i, u_k) \in E^T\}$, where
   $f(u_k)$ denotes the node in the main graph that $u_k$ has
   been mapped to.
5: Compute $C = \bigcap_{v \in M} N(v)$
6: for each $v \in C$ do
7:     if MATCHVERTEX($u_i, v, S'$) then
8:         MATCHSUBGRAPH($i+1, S'$)
9: return
```

Algorithm 5 MATCHVERTEX(u, v, S')

```
MATCHVERTEX$(u, v, S')$
1: if color of $u$ in permutation $S'$ is as same as color of $v$ then
2:     return true
3: return false
```

Algorithm MATCHSUBGRAPH(i, S') finds the candidates C for matching with u_i, and for each matching candidate, it recursively calls itself to match the rest of the nodes $u_{i+1}, u_{i+2}, \ldots$. Algorithm 3 enumerates all permutation of the color set in order to count the number of colorful embeddings of the template. The matching of the color of vertex u in the template and a candidate v is ensured in Algorithm MATCHVERTEX(u, v, S'). The count is divided by a factor α, which is the number of automorphism of the template when the root is fixed. Since the size of $core(G_p)$ is same among all partitions p, the computational cost of enumeration on different partitions is roughly constant, as shown in the green band in Fig. 8. However, the time to generate partitions from the graph linearly increases according to the ID p of the partition G_p, as the red line in Fig. 8 shows. The reason is, in GENERATEPARTITION, we will first skip $nCore \times (p-1)$ vertices to reach the location in the graph file where $core(G_p)$ is located (Line 1 in Algorithm 2). Therefore, the time to reach the location in the first pass is proportional to the partition ID p. The rest $r+1$ passes (Line 4-8) take same amount of the time among partitions. To balance the computational load among processors, we assign the partitions to the processors in a round robin fashion. This keeps the total running time on each processor to be roughly the same, as shown in Fig. 7; the small slightly lower segment in the figure is due to the fact that the corresponding processors are assigned one partition less than the others, and this imbalance can be corrected by careful choice of P and Q.

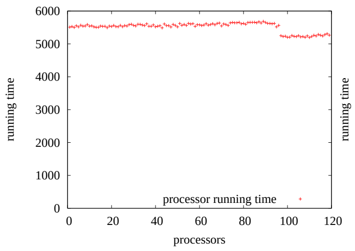

Fig. 7. Running time on different processors.

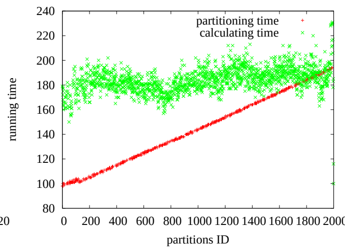

Fig. 8. Partition and calculation time for each partition p.

D. Complete description of Algorithm PARSE

Finally, we describe PARSE in detail in Algorithm 6. The algorithm is a randomized approximation scheme, which repeats the basic steps in lines 5-31 $N_1 \cdot N_2$ times, where $N_1 = O(\log 1/\delta)$ and $N_2 = O(\frac{e^k}{\epsilon^2})$; each iteration computes the number of colorful embeddings of the template for a random coloring of G, and the final result is some form of average of these estimates in lines 32-34. The graph is first partitioned to P partitions by the processors, in Line 1-2. Processor 0 acts as the master/control node and performs the random coloring in lines 5-6. Lines 8-17 are performed by each worker processor, which repeatedly gets a partition G_p (line 11), and computes and sends the counts for the sub-templates for each vertex $v \in core(G_p)$ in lines 12-17. For each sub-template T_i and every possible subset S_i^π of distinct colors for nodes in T_i, the processor computes $C(v, \rho(T_i), S_i^\pi)$. In order to exploit bulk-synchronization, each processor sends one large message corresponding to each node v, which encodes the counts for all possible ways of partitioning the set $S = S_1^\pi \cup S_2^\pi$. This corresponds to the lower level steps of the dynamic program [1]; however, COUNTTEMPLATE directly computes the counts needed in these steps by a backtracking algorithm.

The top level steps of the color coding dynamic program [1] are performed in lines 18-31 of Algorithm 6 by the master processor. For each edge (u, v), the master computes $\sum_{S_1,S_2:S_1\cup S_2=\{1,\ldots,k\}} C(u, \rho(T_1), S_1)C(v, \rho(T_2), S_2) + C(v, \rho(T_1), S_1)C(u, \rho(T_2), S_2)$, for all partitions $S_1 \cup S_2$ of S. The count is divided by a factor β to remove the duplicated enumeration of the same colorful template. It is the number of the cut-edge (u, v) in T, for which the template is isomorphic to itself. Note that the the quantity $\alpha_1 \cdot \alpha_2 \cdot \beta$ is the automorphism of the template T, where α_1 and α_2 are the factors used in Algorithm 3 for the two sub-templates.

E. Performance analysis

Following [1], Algorithm PARSE is a randomized approximation scheme, that estimates the number of embeddings of the template T within any desired level of accuracy, which is a user input; this is summarized below.

Theorem 3.1: For any given parameters ϵ, δ, let $N_1 = O(\log 1/\delta)$ and $N_2 = O(\frac{e^k}{\epsilon^2})$. Then, the estimated number of embeddings of template T in graph G (denoted by Z) as output by Algorithm PARSE, satisfies

$$\Pr[|Z - emb(T,G)| \geq \epsilon \cdot emb(T,G)] \leq \delta,$$

where $emb(T,G)$ denotes the exact count of the number of embeddings.

We now bound the time and space complexity. Let P and Q denote the number of partitions and processors, respectively, and Δ be the average degree in G. Define $k' = \max\{k_1, k - k_1\}$, $n = |V|$, and $m = |E|$.

Theorem 3.2: The total running time and the sum of the sizes of all messages in Algorithm PARSE can be bounded by $O\left(\frac{e^k \log 1/\delta}{\epsilon^2}\left(\frac{n}{Q}\Delta^{k'} + (n+m)k^{k'}\right)\right)$ and $O(nk^{k'})$, respectively.

Proof: Generating partition in Line 2 takes $O((n+m)r)$ time. The loop in Line 1-2 repeats P/Q times for each processor. Therefore, the running time for Line 1-2 is $O((n+m)\frac{rP}{Q})$. Random coloring to the graph in Line 6 takes $O(n)$ time. In Line 11, the time to read a partition by one processor takes $O(n+m)$ time. Function COUNTTEMPLATE (Line 14) matches the root of template T_1 with node v only. Any other node u' of T_1 can be matched only after one of u's neighbor

Algorithm 6 Algorithm PARSE($T(V^T, E^T)$, $G(V, E)$, ϵ, δ)

```
1:  for $p \leftarrow me - 1; p < P; p \leftarrow p + Q - 1$ do
2:      $G_p \leftarrow$ GeneratePartition($G(V, E), p, r, P$) {$me$ is the current processor's ID}
3:  for $j$ from 1 to $N_1$ do
4:      for $i$ from 1 to $N_2$ do
5:          if $me = 0$ then
6:              Color($G(V, E)$, $S$) {randomize coloring the original graph}
7:          Barrier {synchronizing}
8:          if $me > 0$ then
9:              LoadTemplate($T$)
10:             for $p \leftarrow me - 1; p < P; p \leftarrow p + Q - 1$ do
11:                 LoadPartition($G_p$)
12:                 for each vertex $v \in core(G_p)$ do
13:                     for each $S_1^\pi \bigcup S_2^\pi = S, S_1^\pi \bigcap S_2^\pi = \phi$, $\pi \in [1, \binom{k}{k_1}]$ do
14:                         $c_1 \leftarrow$ COUNTTEMPLATE($v, T_1, S_1^\pi$)
15:                         $c_2 \leftarrow$ COUNTTEMPLATE($v, T_2, S_2^\pi$)
16:                         $val[\pi] \leftarrow (c_1, c_2)$
17:                     Send($val$, 0) {send to processor 0}
18:         if $me = 0$ then
19:             $Count \leftarrow 0$
20:             for each $v \in G$ do
21:                 Recv($val$)
22:                 $valMatrix[v] \leftarrow v$
23:             for each $v \in G$ do
24:                 for each $u \in N(v)$ do
25:                     if $u > v$ then
26:                         for $\pi$ from 1 to $\binom{k}{k_1}$ do
27:                             $Count \leftarrow Count + valMatrix[v][\pi].c_1 \times valMatrix[u][\pi].c_2$
28:                             $Count \leftarrow Count + valMatrix[v][\pi].c_2 \times valMatrix[u][\pi].c_1$
29:             $Count \leftarrow Count/\beta$
30:             $Count \leftarrow Count \times \frac{k^k}{k!}$
31:             $X_i = Count$
32:     if $me = 0$ then
33:         $Y_j \leftarrow \sum_i X_i/N_2$
34: Output the median of $\{Y_1, \ldots, Y_{N_1}\}$
```

w' has been matched with some $w \in V$, and u' can only be matched with a neighbor u of w such that color(u') = color(u). Since k colors are randomly assigned, any vertex $u' \in T_1$ can be matched with, on average, Δ/k nodes in V. Therefore the running time of COUNTTEMPLATE (both in Line 14 and 15) is bounded by $O\left(\left(\frac{\Delta}{k}\right)^{k'}\right)$ as $|T_1| = k_1 \leq k'$. The loop in Line 13-16 repeats $k^{k'}$ times and thus it takes $O(\left(\frac{\Delta}{k}\right)^{k'} \cdot k^{k'}) = O(\Delta^{k'})$ time.

The *Send* statement in Line 17 takes $O(k^{k'})$ time and the loop in Line 12-17 repeats n/P times for each partition. Thus this loop takes $O((\Delta^{k'} + k^{k'})n/P)$ time. The loop in Line 10-17 repeats P/Q times for each processor. Then this loop takes $O\left((\Delta^{k'} + k^{k'})\frac{n}{Q} + (n+m)\frac{P}{Q}\right)$ time.

Line 19-31 is executed by Processor 0. Line 20-22 takes $O(nk^{k'})$ time. Loop in Line 26-28 takes $O(k^{k'})$ time. Two loops combined in Line 23-24 repeats $O(m + n)$ times and takes $O((m+n)k^{k'})$ time. Therefore total time for Line 19-31 is $O((m + n)k^{k'})$. Considering that $\frac{rP}{Q} < k^{k'}$, total time for Line 5-31 is given by:

$$O\left((\Delta^{k'} + k^{k'})\frac{n}{Q} + (n+m)\frac{P}{Q} + (n+m)k^{k'}\right)$$
$$= O\left(\frac{n}{Q}\Delta^{k'} + (n+m)k^{k'}\right).$$

The outermost two loops Line 3-4 combined repeat $N_1 \cdot N_2 = \frac{e^k \log 1/\delta}{\epsilon^2}$ times. Therefore, the total running time is:

$$O\left(\frac{e^k \log 1/\delta}{\epsilon^2}\left(\frac{n}{Q}\Delta^{k'} + (n+m)k^{k'}\right) + (n+m)\frac{rP}{Q}\right)$$
$$= O\left(\frac{e^k \log 1/\delta}{\epsilon^2}\left(\frac{n}{Q}\Delta^{k'} + (n+m)k^{k'}\right)\right).$$

Finally, for the total message size, observe that one execution of the *Send* statement (Line 17) sends $O(k^{k'})$ data items and it is executed total n times by all processors combined. Thus total size of messages sent is $O(nk^{k'})$. ■

The bounds in Theorem 3.2 show the limits to which we can expect the current implementation of PARSE to scale -

the communication cost will become the significant bottleneck when $\frac{n}{Q}\Delta^{k'} < (n+m)k^{k'}$, i.e., when $Q > \frac{n\Delta^{k'}}{(n+m)k^{k'}}$.

IV. Experiments

We now discuss the performance of Algorithm ParSE on the datasets discussed in Section II. Our experiments are based on the templates in Fig. 9 with sizes varying from 4 to 10 nodes; each template is divided into two sub-templates using the cut edge; the radius of each sub-template in Fig. 9, relative to its root is either 1 or 2.

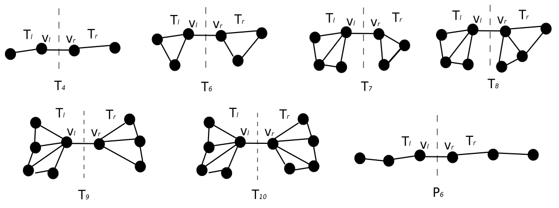

Fig. 9. Templates with size varying from 4 to 10.

We study the following specific aspects: (i) the time, space and communication costs of various steps of ParSE, (ii) the approximation error, and the number of iterations needed in practice, (iii) strong and weak scaling of ParSE and (iv) exact counting on small templates (such as cliques), which fit within a partition, and can be enumerated directly using Algorithm 3 (CountTemplate), without the need for color coding.

Our main observations include: (i) in practice, the number of iterations needed for our algorithm to produce an (ϵ, δ) approximation is much smaller than the theoretical bound of Theorem 3.1, (ii) the algorithm scales quite well up to 350 processors. In contrast, most of the prior results run in a few hours for graphs which are at least two orders of magnitude smaller.

A. Number of iterations and approximation error

Theorem 3.1 gives an upper bound of $N_1N_2 = O(\frac{e^k \log 1/\delta}{\epsilon^2})$ on the number of iterations of the basic color coding computation in Algorithm ParSE, in order to get an (ϵ, δ)-approximation to the actual number of embeddings. In practice, we find that this is a fairly loose bound, and as shown in Fig. 10 and 12, choosing even $N_1 = 1$ and $N_2 = 3$ reduces the approximation error ϵ to less than 0.1%. In Fig. 11, choosing $N_1 = 1$ and $N_2 = 4$ reduces the error ϵ to 5%. Therefore, for the remaining experiments in this section, we use only a small number of iterations.

B. Comparison with sequential algorithm

We also compare the running time of ParSE and Huffner's algorithm (HF) [8] on GNP50. Giving the edges equal weights, HF will enumerate all paths with a given length. The results are shown in Fig. 13 and 14, for template T_4 and P_6, respectively. It shows that our algorithm almost reaches 10 times faster than HF algorithm for T_4 and 100 times faster for P_6.

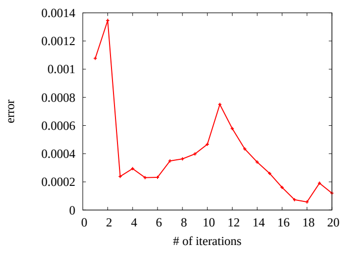

Fig. 10. Error for ParSE, conducted on GNP100, using template T_4

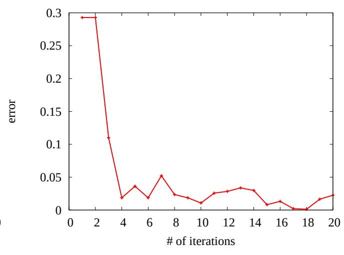

Fig. 11. Error for ParSE, conducted on GNP100, using template T_6

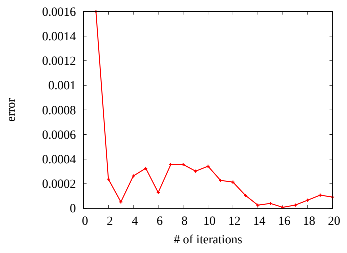

Fig. 12. Error for ParSE, conducted on NRV, using template T_4

C. Running Time of various steps in Algorithm ParSE

The overall running time of ParSE can be roughly split into 4 parts: graph partitioning, initialization (include graph random coloring), local template counting and final aggregation. Figure 15 and 16 show the running time of these parts on NRV, for T_6 and T_8, respectively. We choose $N_1 = 1$, $N_2 = 5$ and $P = 1000$. We observe that for a given template, the initialization and the finalization time do not vary much with different number of processors, since these are done only by the master node. The partitioning and counting times decrease with the number of processors. The partitioning time for both T_6 and T_8 is roughly the same, while the counting time changes significantly between Fig. 15 and 16, since the running time of subgraph counting is exponential in the template size. In Fig. 16, the counting time dominates the total running time. The finalization time also differs in Fig. 15 and 16, because large templates imply more choices of the color set, making the aggregation in Line 26-28 of Algorithm 6 more expensive.

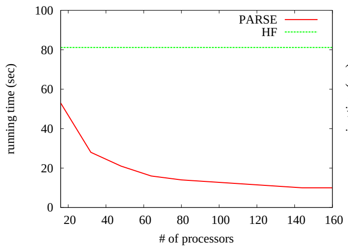

Fig. 13. Running time of ParSE VS. sequential algorithm, conducted on GNP50, using template T_4

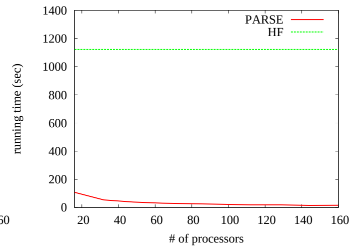

Fig. 14. Running time of ParSE VS. sequential algorithm, conducted on GNP50, using template P_6

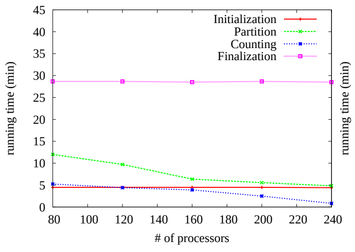

Fig. 15. Time usage on various steps with different number of processors, on a single iteration, on NRV. Template is T_6

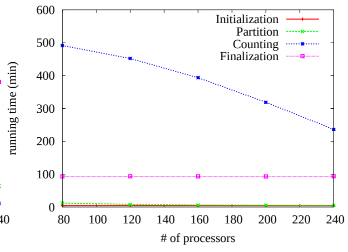

Fig. 16. Time usage on various steps with different number of processors, on a single iteration, on NRV. Template is T_8

D. Scaling of ParSE

Figures 17 shows the strong scaling of running time with number of processors ranging from 160 to 400 on Miami for template T_6. Note that for small template such as T_6, finalization/aggregating time is a major part of the total running time, and it is independent of the number of processors. Thus when we increase the number of processors, the running time does not decrease linearly; however, up to 350 processors, it still scales well. For our graph data, showing weak scaling is not an easy task as varying the graph size without affecting its structural properties is difficult. Fig. 18, shows $\frac{total\ running\ time}{P/Q}$ varying with the number of processors, which is a measure somewhat similar to weak scaling. Due to the communication and finalization cost, the running time on each processor increase with the number of processors. For large template such as T_8, since finalization time is not a major part of total running time, we expect the weak scaling to be better. Fig. 19 shows the total running time increases linearly with the number of partitions. Therefore, given the number of processors, we can obtain the fastest running by letting $P = Q - 1$.

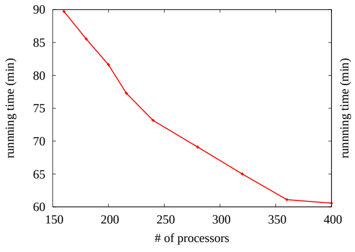

Fig. 17. Running time on Miami (of a single iteration) scaling with # of processors.

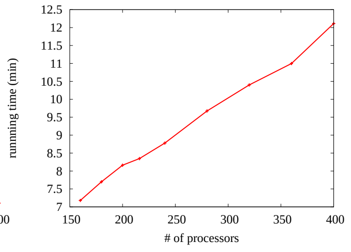

Fig. 18. Running time of a processor counting on a single partition of Miami, scaling with # of processors.

We also study how running time is scaled with the increasing of the template size. In Fig. 20, we use template T_6, T_7, T_8, T_9 and T_{10} from Fig. 9, and experiment on NRV. The number of processor employed here is 80, and the partition number is 1000. We found that for template size up to 10, our algorithm can still finish the counting within 12 hours in a graph with hundreds of thousands nodes and average degree more than a hundred.

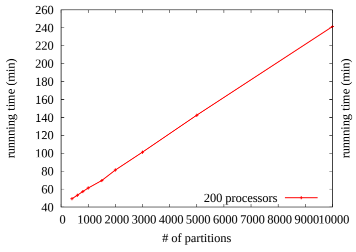

Fig. 19. Running time on Miami scaling with # of partitions.

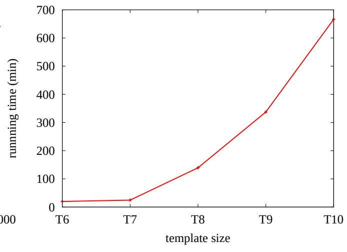

Fig. 20. Running time on NRV scaling with various template size.

E. Counting Template with Small Radius

For a template with small radius, e.g., with $rad(T) = 1$, the template can be contained completely in a partition without splitting it. In such a case, we do not need color coding and can find exact count using less communication time and finalization time. Such a template can be counted in each partitions using the algorithms in Section III-C, but the color set S and related permutation procedure are not necessary; two vertices u and v are matched without the condition that their colors need to be the same. The final count is the summation of the counts on each partition, divided by the number of automorphism of the template. Fig. 21 shows that even for graph which contains 20 million vertices, the total time to count 5-cliques is less than 13 hours. A clique $G_c(V_c, E_c)$ is a complete graph in which for each pair of the vertices $u, v \in V_c$, there is an edge $e(u, v) \in E_c$. A clique containing k vertices is called a k-clique. The experiments have been done with 120 processors and 2000 partitions for each graph. Note that NRV requires approximately the same running time as Chicago, though the graph size is much smaller, for the reason that it has much larger average degree Δ (note that the time bound has a factor $\Delta^{k'}$). Also, the number of 4-clique embeddings on NRV is the same order of magnitude as Chicago and the number of 5-clique embeddings is the largest among all networks, as shown in Fig. 22.

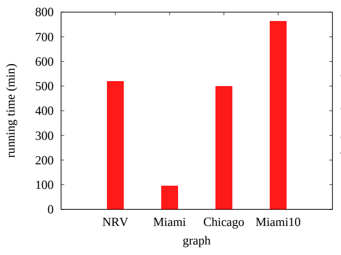

Fig. 21. Running time for counting 5-cliques with 120 processors.

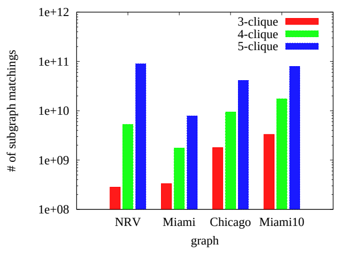

Fig. 22. Number of 3, 4, and 5-cliques in different graphs.

V. Implementation Specifications

Environment: The experiments were performed on an SGI Altix ICE 8200 system. It has 96 nodes and each node has 2 Intel Quad-Core Xeon E5440 processor, and 16GB memory, or 2GB per core.

Data structure: Vertices' IDs are mapped to a continues space which stores the pointers to the neighbor lists of the vertices. We avoid any STL datastructure in the graph storage, due to its high memory cost.
Communication: We use the default **Recv()** function in MPI which is non-blocking and each call to the **Recv()** function which is not matched by any **Send()** will generate a message header. Therefore we need to be cautious to configure the MPI_MAX environment variable, which set the maximum number of message header allowed. In our case for running on Miami, processor number below 160 will cause the number of message headers to exceed the default MPI_MAX value. The solution can be either increasing the MPI_MAX value, which may also increase the running time due to the overhead in system level, or passing larger messages, each of which contain multiple messages we currently use.

VI. Related Work

As discussed earlier, there has been a lot of work on sequential algorithms for determining the counts of specific subgraphs, as well as finding statistically significant subgraph patterns. [1; 5–9; 11]. Many of these approaches are based on back-tracking, dynamic programming, and using techniques such as color coding. These results typically only scale to graphs with few thousand nodes. More efficient algorithms have been developed for the problem of finding frequent subgraphs, e.g., Kuramochi et al. [9; 10] using adaptations of breadth-first and depth-first based candidate generation methods to scale to about 120,000 nodes. There has been limited work on parallel algorithms for subgraph enumeration. One thread of work has been on enumerating maximal cliques. Starting with the parallel algorithm of Bron and Kerbosch [4], there has a lot of subsequent work, e.g., Schmidt et al. [13]; these algorithms are based on making the basic back-tracking procedure more efficient using additional information, but many of these technique is specific to cliques. This has been extended to other subgraphs by Wang et al. [14]; their approach is also based on local enumeration in a suitable partitioning, and is shown to run on biological networks with a few thousand nodes.

VII. Conclusions

Algorithm ParSE is a new approach for parallel subgraph enumeration, and our results are the first to scale to graphs with $10^5 - 10^6$ nodes, for templates of size up to 10 (which can be partitioned by a cut-edge), within a few hours. This basic approach can be extended to larger templates which cannot be partitioned by a cut-edge, by extending the dynamic programming framework, though a new approach is needed to address the space and communication cost. We expect these techniques to broaden the scope of the usage of subgraph enumeration in social networks and other applications.

Acknowledgements. We thank our external collaborators and members of the Network Dynamics and Simulation Science Laboratory (NDSSL) for their suggestions and comments. This work has been partially supported by NSF Nets Grant CNS-0626964, NSF HSD Grant SES-0729441, NIH MIDAS project 2U01GM070694-7, NSF PetaApps Grant OCI-0904844, DTRA R&D Grant HDTRA1-0901-0017, DTRA CNIMS Grant HDTRA1-07-C-0113, NSF NETS CNS-0831633, NSF CAREER 0845700, DHS 4112-31805, DOE DE-SC0003957 and NIH/CDC 1P01CD000284-01.

References

[1] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):241–249, July 2008.

[2] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast network decomposition. In *PODC*, pages 169–177, 1992.

[3] C. Barrett, D. Beckman, M. Khan, V.S. Anil Kumar, M. Marathe, P. Stretz, T. Dutta, and B. Lewis. Generation and analysis of large synthetic social contact networks. In *Winter Simulation Conference*, 2009.

[4] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 1973.

[5] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, October 2004.

[6] M. Gonen, D. Ron, and Y. Shavitt. Counting stars and other small subgraphs in sublinear time. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[7] M. Gonen and Y. Shavitt. Approximating the number of network motifs. In *The 6th Workshop on Algorithms and Models for the Web Graph (WAW)*, 2009.

[8] F. Huffner, S. Wernicke, and T. Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 2007.

[9] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. IEEE International Conference Data Mining (ICDM)*, 2001.

[10] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 2005.

[11] J. Leskovec, A. Singh, and J. Kleinberg. Patterns of influence in a recommendation network. In *Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2006.

[12] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, pages 824–827, 2002.

[13] Matthew C. Schmidt, Nagiza F. Samatova, Kevin Thomas, and Byung-Hoon Park. A scalable, parallel algorithm for maximal clique enumeration. In *Journal of Parallel and Distributed Computing*, 2009.

[14] T. Wang, J. Touchman, W. Zhang, E. Suh, and G. Xue. A parallel algorithm for extracting transcriptional regulatory network motifs. In *IEEE Symposium on Bioinformatics and Bioengineering*, 2005.