FASCIA

Fast Approximate Subgraph Counting and Enumeration

George M. Slota Kamesh Madduri

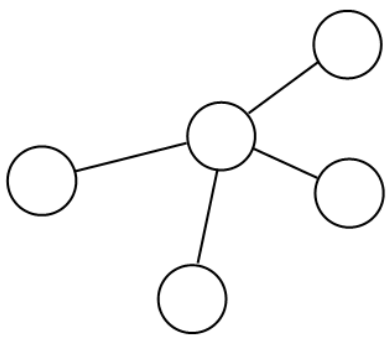
Scalable Computing Laboratory
The Pennsylvania State University

2 Oct. 2013

Overview

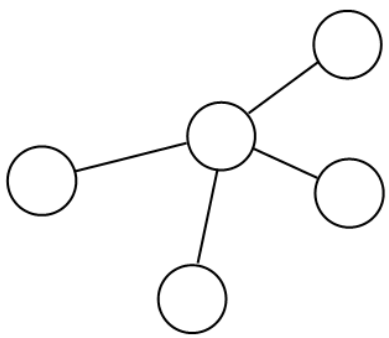
- Background
- Motivation
- Color-coding
- Related Work
- FASCIA
- Results
- Future Work

Background
Subgraph Counting

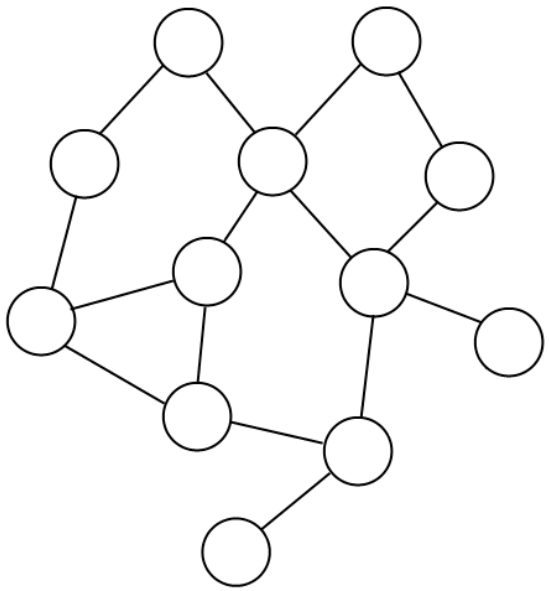


Template

Background
Subgraph Counting

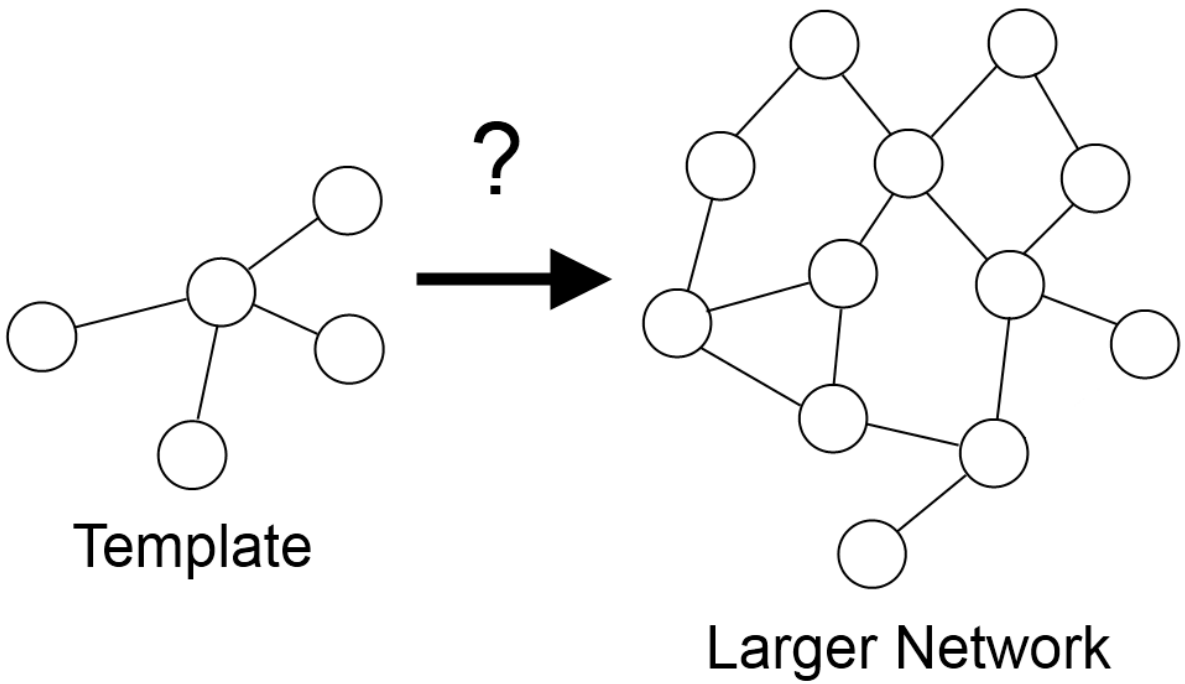


Template

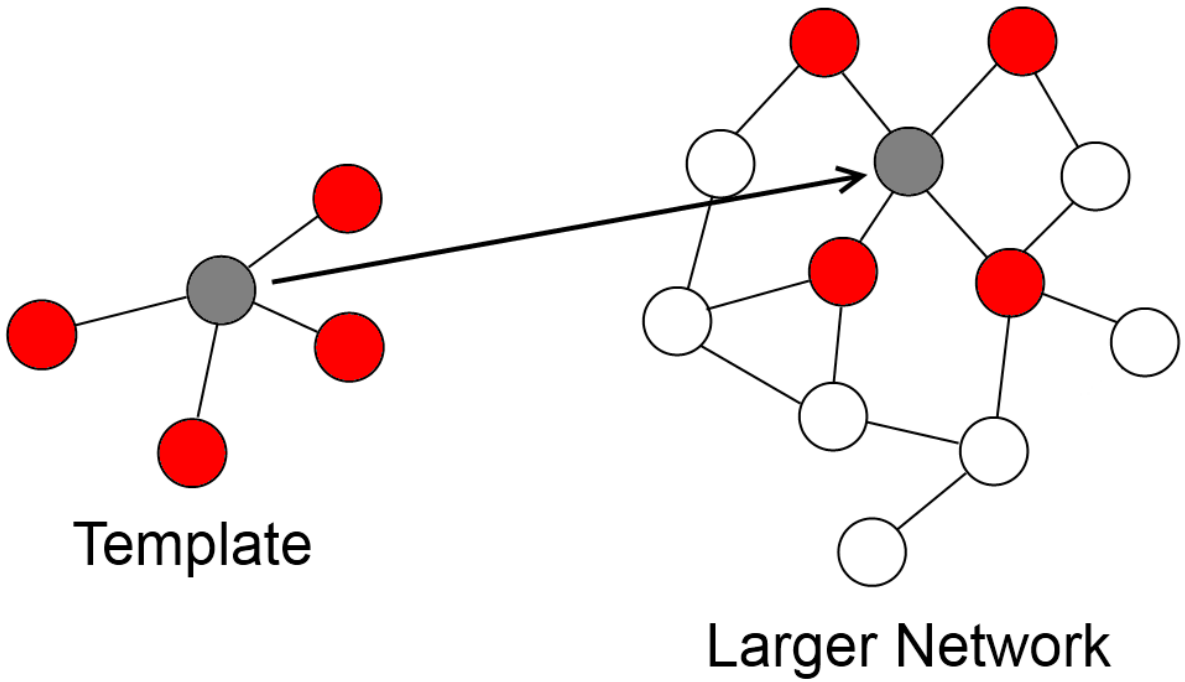


Larger Network

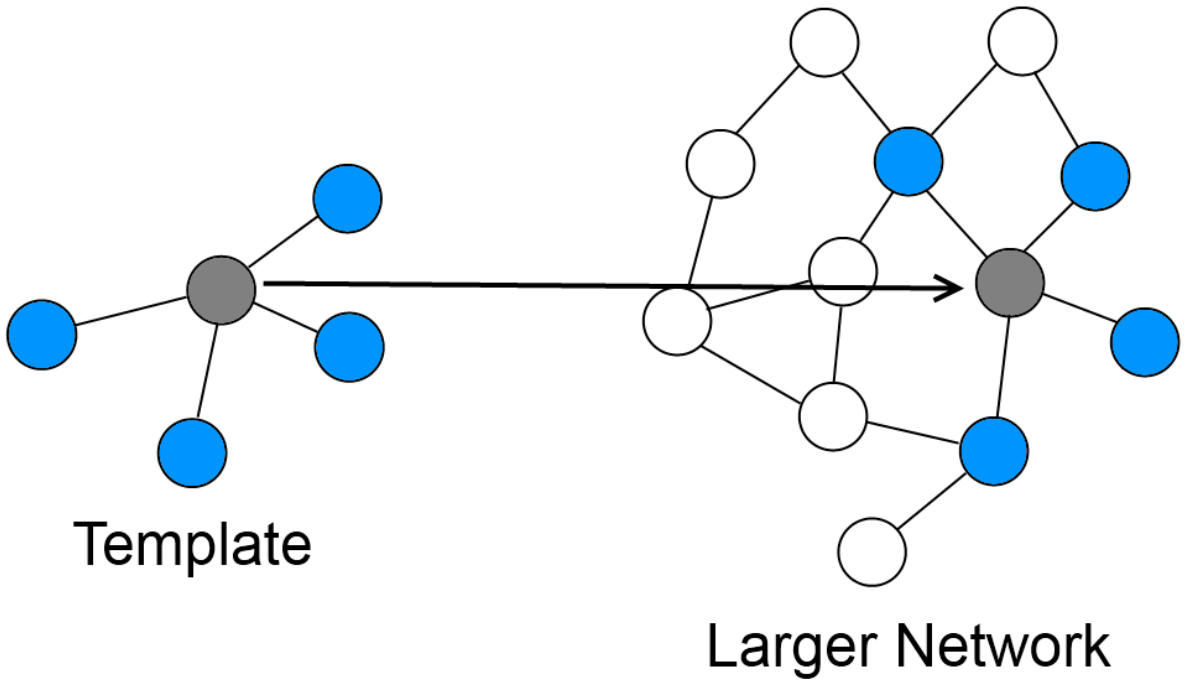
Background
Subgraph Counting



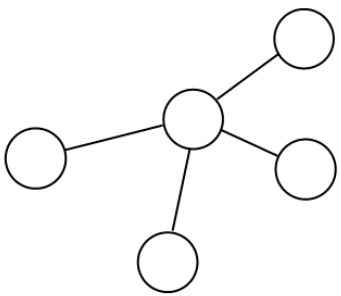
Background
Subgraph Counting

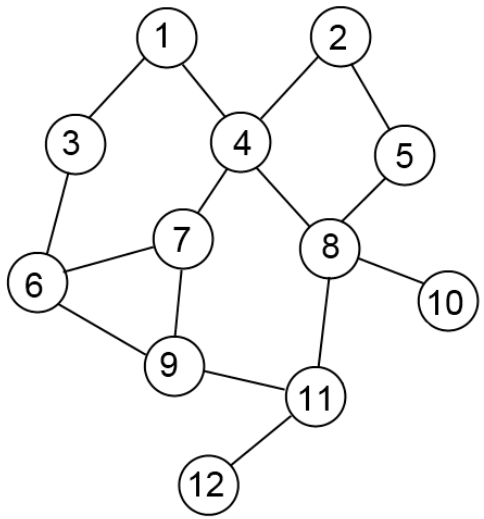


Background
Subgraph Counting

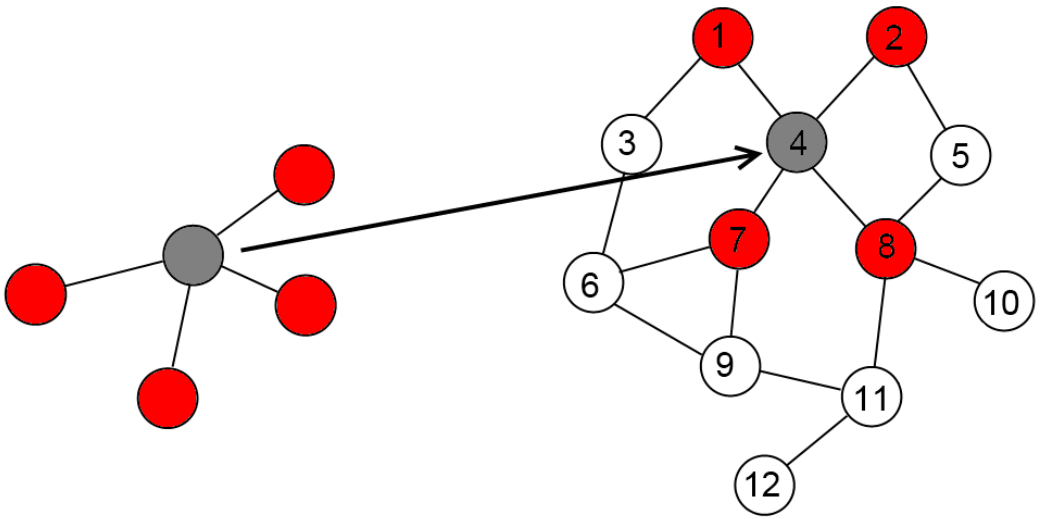


Background
Subgraph Enumeration



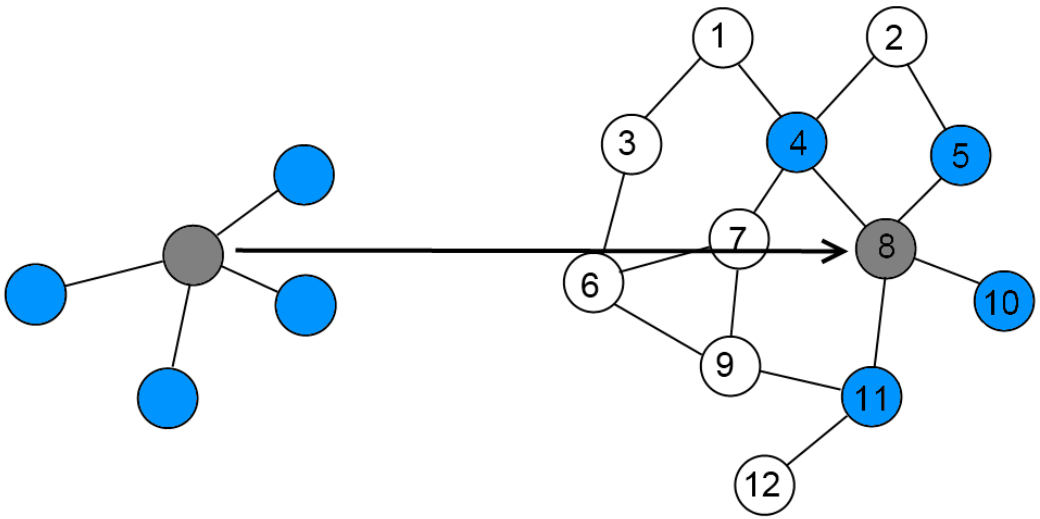


Background
Subgraph Enumeration



Mapped to vertices:
4 1 2 7 8

Background
Subgraph Enumeration



Mapped to vertices:
4 1 2 7 8
8 4 5 11 10

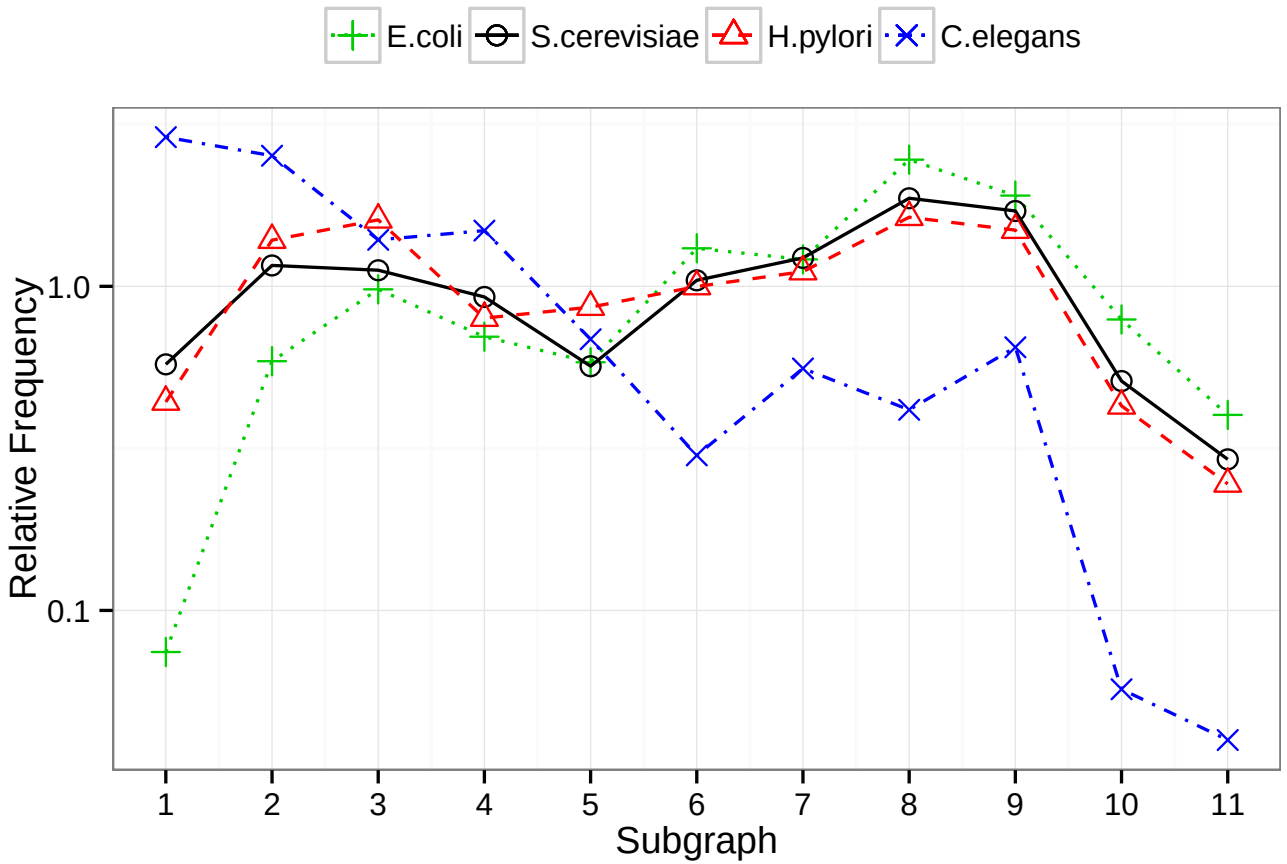
Motivation

Why do we want fast algorithms for subgraph counting?

- Important to bioinformatics, chemoinformatics, social network analysis, communication network analysis, etc.
- Forms basis of more complex analysis
  - Motif finding
  - Graphlet frequency distance (GFD)
  - Graphlet degree distributions (GDD)
- Counting and enumeration on large networks is very tough, $O(n^k)$ complexity for naïve algorithm

Motivation
Motif finding, GFD, and GDD

- Motif finding: Look for all subgraphs of a certain size



Motivation
Motif finding, GFD, and GDD

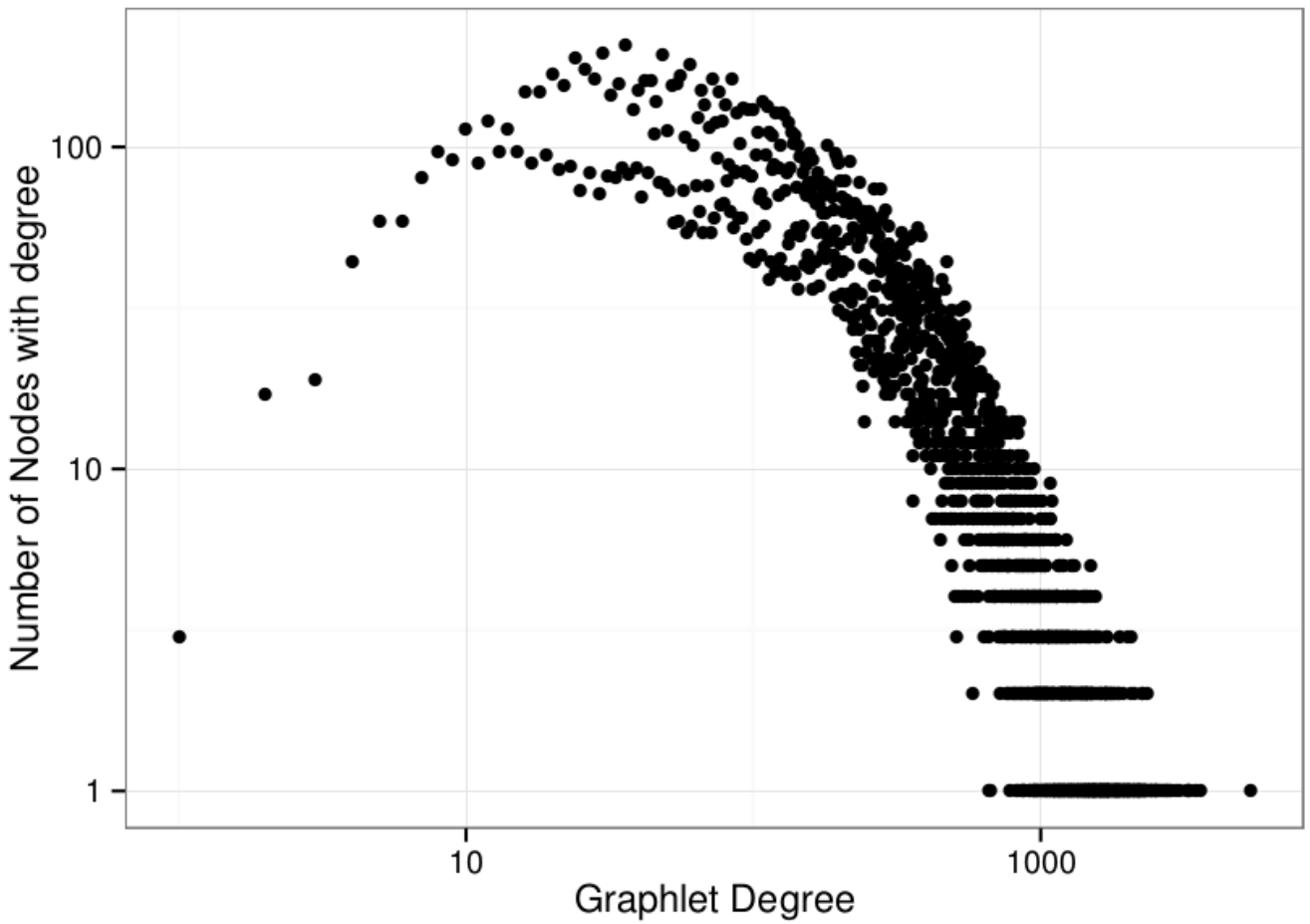
- Motif finding: Look for all subgraphs of a certain size
- GFD: Numerically compare occurrence frequency to other networks

$$S_i(G) = -\log\left(\frac{C_i(G)}{\sum_{i=1}^{\mathrm{n}} C_i(G)}\right)$$

$$D(G,H) = \sum_{i=1}^{\mathrm{n}} |S_i(G) - S_i(H)|$$

Motivation
Motif finding, GFD, and GDD

- Motif finding: Look for all subgraphs of a certain size
- GFD: Numerically compare occurrence frequency to other networks
- GDD: Numerically compare embeddings/vertex distribution



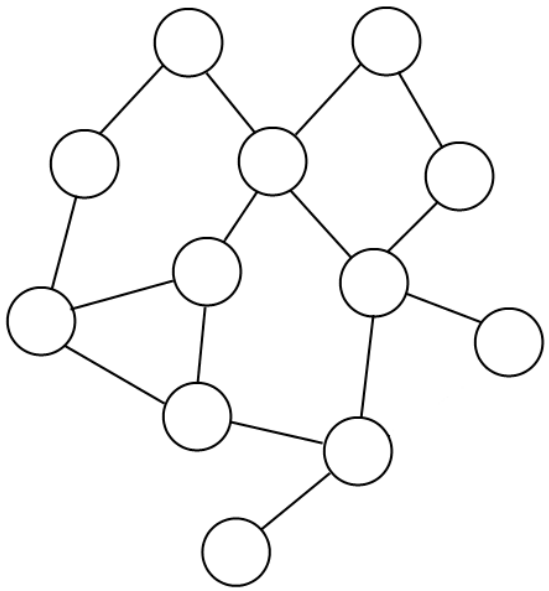
Color-coding

Color-coding for Approximate Subgraph Counting

- Color-coding [Alon et al., 1995] is a way to approximate the number of *tree-like non-induced subgraph* embeddings in a network
- We randomly color a graph with k colors and count the number of *colorful* subgraph embeddings
- We retrieve an approximate total count by scaling the colorful count by the probability that any given embedding would be colorful
- Run multiple times with unique graph colorings to get final average
- Using dynamic programming approach (to be explained), each iteration can run in $O(m \cdot 2^k \cdot e^k)$

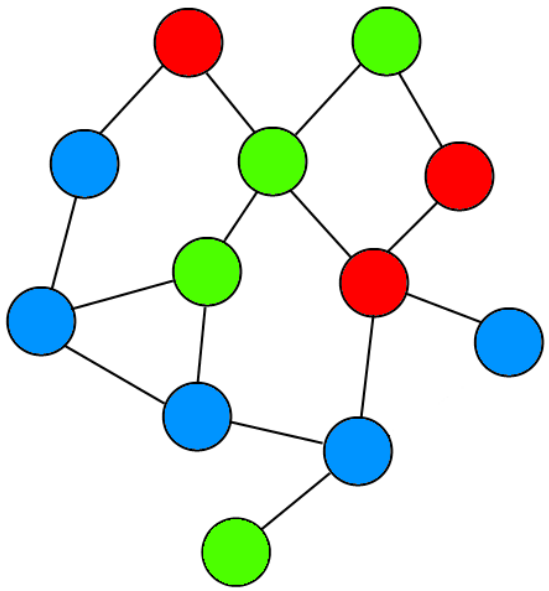
Color-coding

Color-coding for Approximate Subgraph Counting



Color-coding

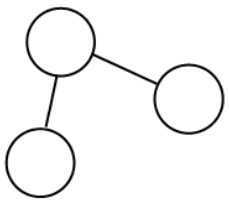
Color-coding for Approximate Subgraph Counting

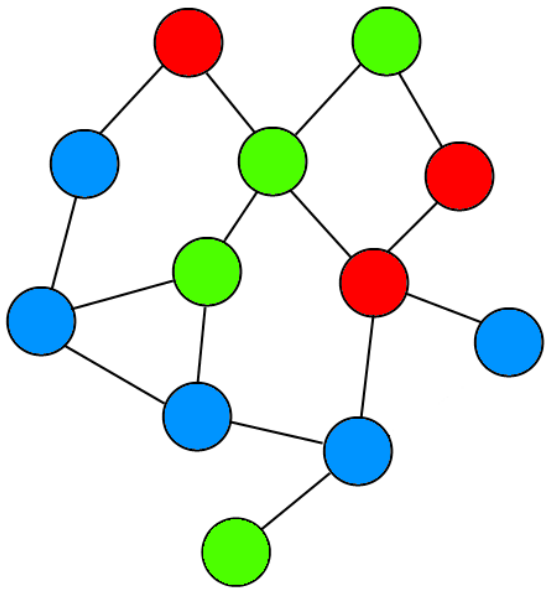


Color-coding

Color-coding for Approximate Subgraph Counting

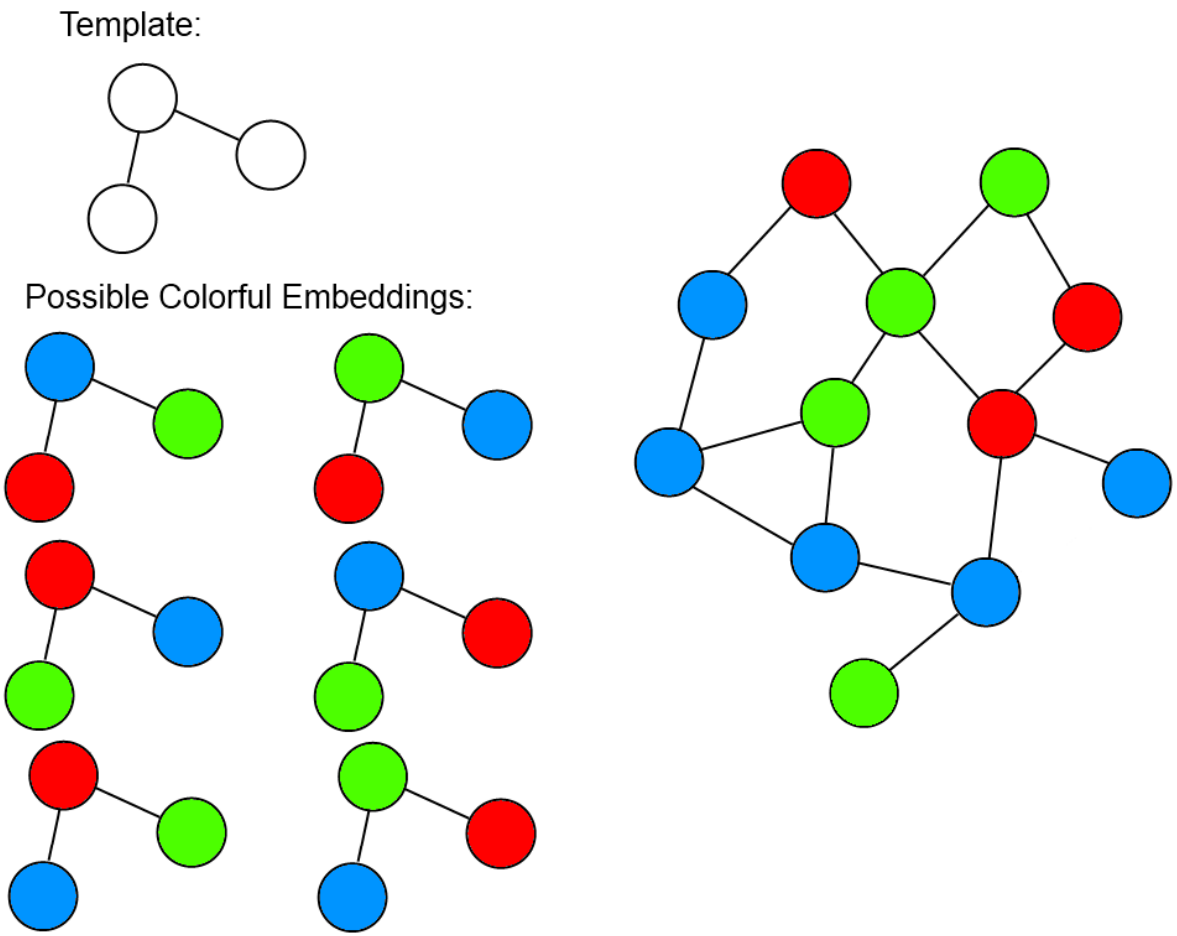
Template:





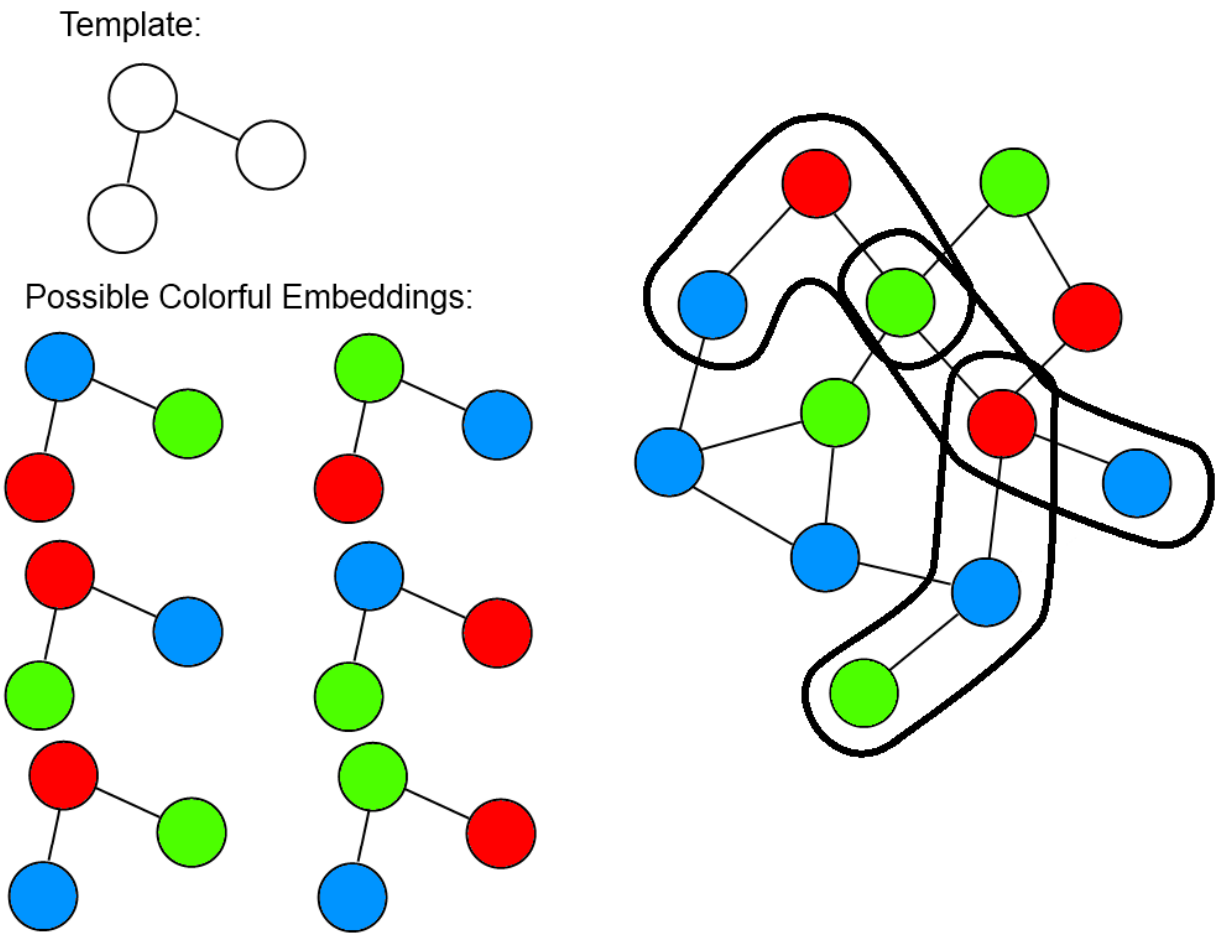
Color-coding

Color-coding for Approximate Subgraph Counting



Color-coding

Color-coding for Approximate Subgraph Counting

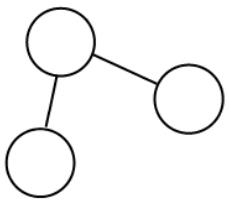


Color-coding

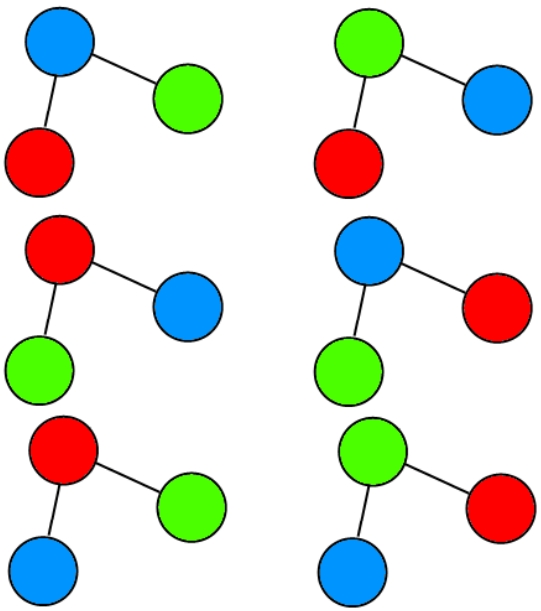
Color-coding for Approximate Subgraph Counting

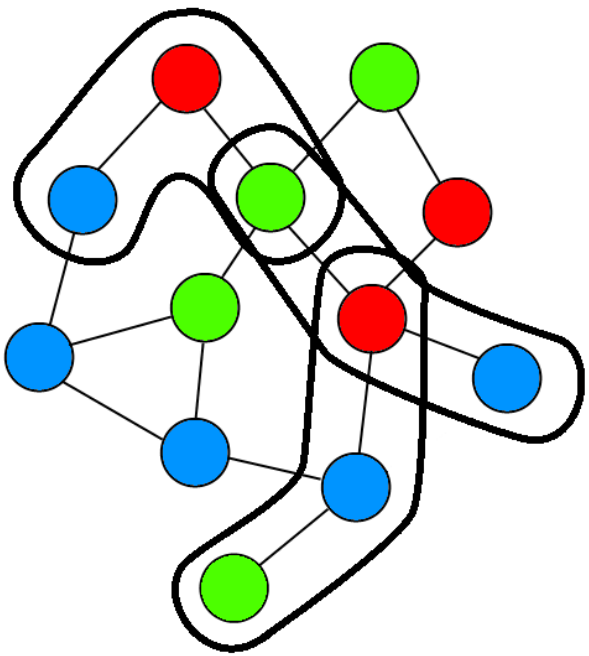
- $cnt_{colorful} = 3, C_{total} = 3^3, C_{colorful} = 3!, P = \frac{3!}{3^3}$
- $cnt_{estimate} = \frac{cnt_{colorful}}{P} = \mathbf{13.5}$

Template:



Possible Colorful Embeddings:

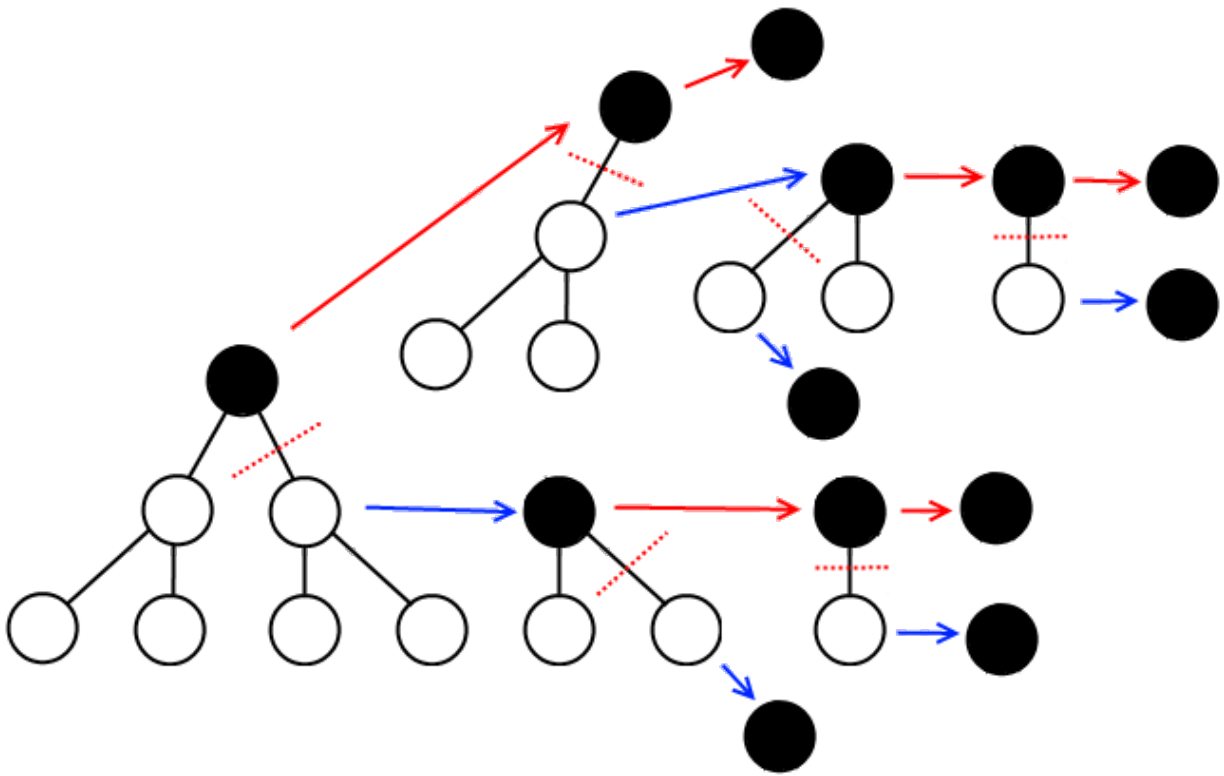




Color-coding
Algorithmic Overview Description

1: Partition input template T (k vertices) into subtemplates S_i using *single edge cuts*.
2: Select *Niter* to be performed
3: **for** $i = 1$ to *Niter* **do**
4: Randomly assign to each vertex v in graph G a color between 0 and $k - 1$.
5: **for all** $v \in G$ **do**
6: Use a **dynamic programming scheme** to count *colorful* non-induced occurrences of T rooted at v.
7: **end for**
8: **end for**
9: Take average of all *Niter* counts to be final count.

Color-coding
Template partitioning

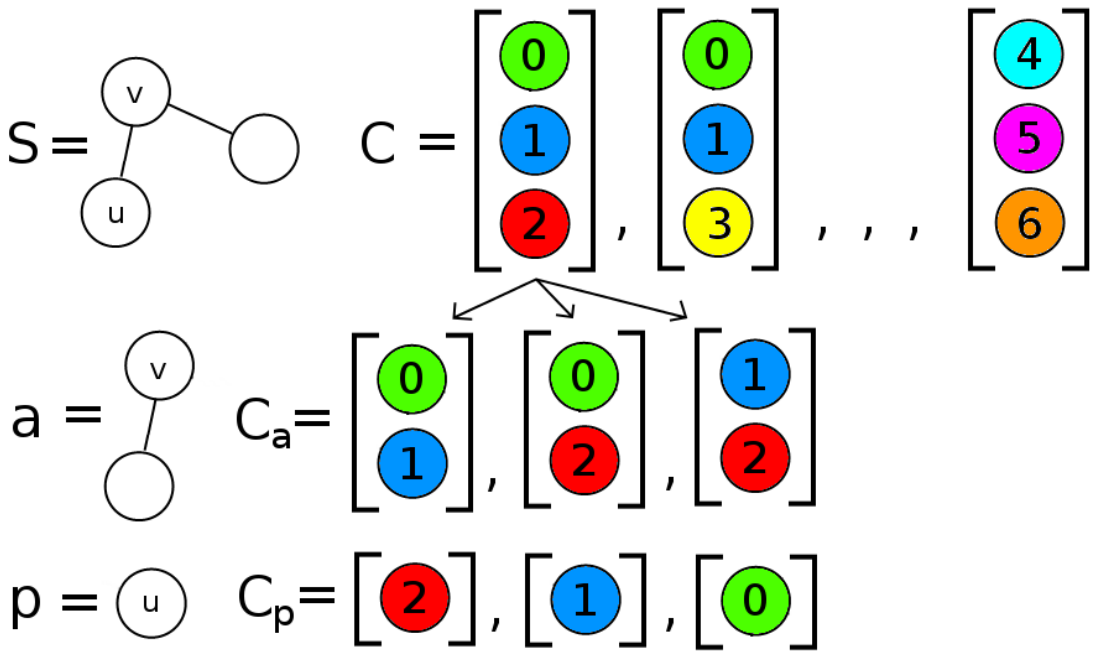


Color-coding
Dynamic Programming Step

1: **for all** sub-templates S_i created from partitioning T, in reverse order they were created during partitioning **do**
2: **for all** vertices $v \in G$ **do**
3: **if** S_i consists of a single node **then**
4: Set **table**$[S_i][v][color\ of\ v] := 1$
5: **else**
6: S_i consists of active child a_i and passive child p_i
7: **for all** colorsets C of unique values mapped to S **do**
8: Set $count := 0$
9: **for all** $u \in N(v)$, where $N(v)$ is the neighborhood of v **do**
10: **for all** possible combinations C_a and C_p created by splitting C and mapping onto a_i and p_i **do**
11: $count$ += table$[a_i][v][C_a]$·table$[p_i][u][C_p]$
12: **end for**
13: **end for**
14: Set table$[S_i][v][C] := count$
15: **end for**
16: **end if**
17: **end for**
18: **end for**
19: $templateCount := \sum_v \sum_C$ table$[T][v][C]$

Color-coding

Colorset and count calculation



$$table[S][v][C] = \sum_{C_a C_p} \sum_{u} table[a][v][C] * table[p][u][C]$$

Related Work

Color-coding, non-induced subgraph enumeration

- Alon et al.'s Implementation [Alon et al., 2008]
  - Motif finding on PPI networks
- MODA [Omidi et al., 2009]
  - Uses approximation or exact scheme
  - Motif finding on small networks
- PARSE [Zhao et al., 2010]
  - Distributed color-coding algorithm using MPI
  - Handles large graphs through partitioning
- SAHAD [Zhao et al., 2012]
  - Distributed color-coding algorithm using Hadoop (MapReduce)
  - Handles vertex-labeled graphs, computes graphlet degree distributions

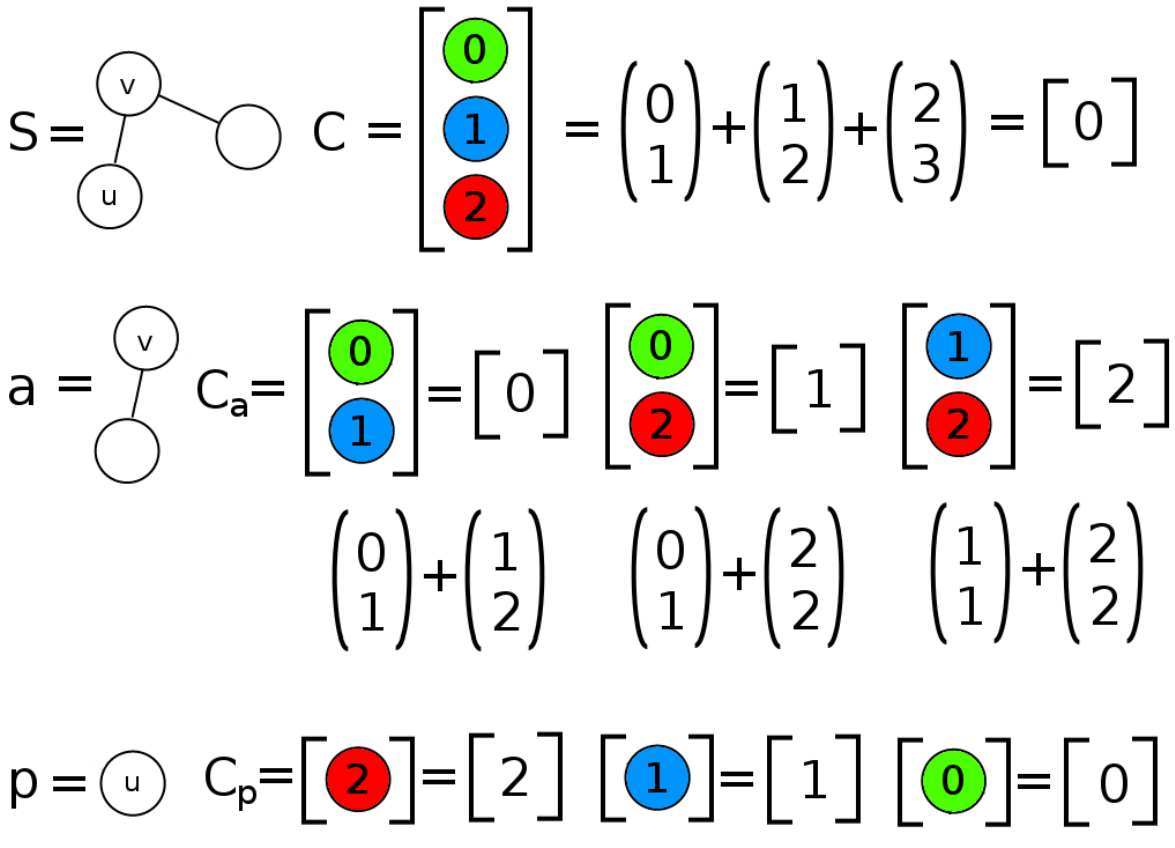
FASCIA
Overview

FASCIA: Fast Approximate Subgraph Counting and Enumeration

- Count and enumerate subgraphs, supports node labels
- Perform motif finding, calculate GDD
- Algorithmic Optimizations:
  - Combinatorial indexing scheme for color mappings
  - Multiple shared-memory parallelization strategies
  - Memory reduction via array design, hashing schemes
  - Speedup via template partitioning and work avoidance

FASCIA
Combinatorial indexing scheme

- Combinatorial number system to represent colorsets:
  $C = \binom{c_1}{1} + \binom{c_2}{2} + \cdots + \binom{c_k}{k}$, where $c_1 < c_2 < \cdots < c_k$
- Precompute indexes and ordering in advance, store in table (<2MB for k =12)
- This avoid avoids explicit handling/passing of colors, or computation of colorset indexes during runtime



FASCIA
Shared memory parallelization

- Inner loop parallelization: **forall** $v \in G$
- Outer loop parallelization: **for** $i = 1$ to *Niter*
- Outer loop requires individual dynamic tables for each separate iteration, storage increases linearly with thread count
- Possible to do arbitrary combinations, e.g. a 12 thread machine with 2 outer loop threads each with 6 inner loop threads

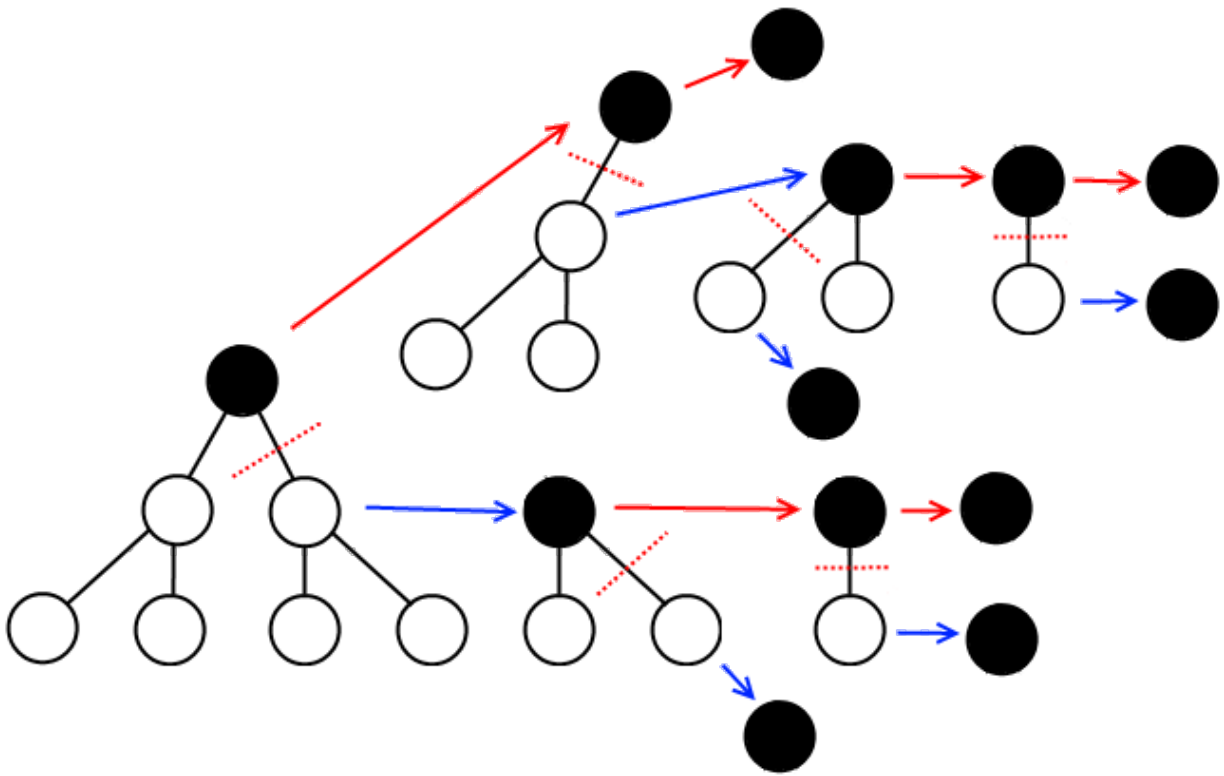
```
1: Partition input template T
2: Select Niter to be performed
3: for i = 1 to Niter in parallel do
4:     Randomly color G
5:     for all S_i created during partitioning, a_i and p_i children do
6:         for all v ∈ G in parallel do
7:             ....
8:         end for
9:     end for
10: end for
11: Take average of all Niter counts to be final count.
```

FASCIA
Memory optimizations

- We implement both a three-level array and hash table
- Items in array are stored/retrieved as table$[S_i][v][C]$
- Hash table, for each subtemplate, $key = v * N_c + C$
- If we initialize and resize hash table to be a factor of $N_v * N_c$, then simple hash function ($key \mod N_v$) results in a relative uniform distribution based on initial random graph coloring
- Generally: array method more memory efficient for dense graphs, hash table more efficient for sparse graphs

FASCIA
Template partitioning and work avoidance

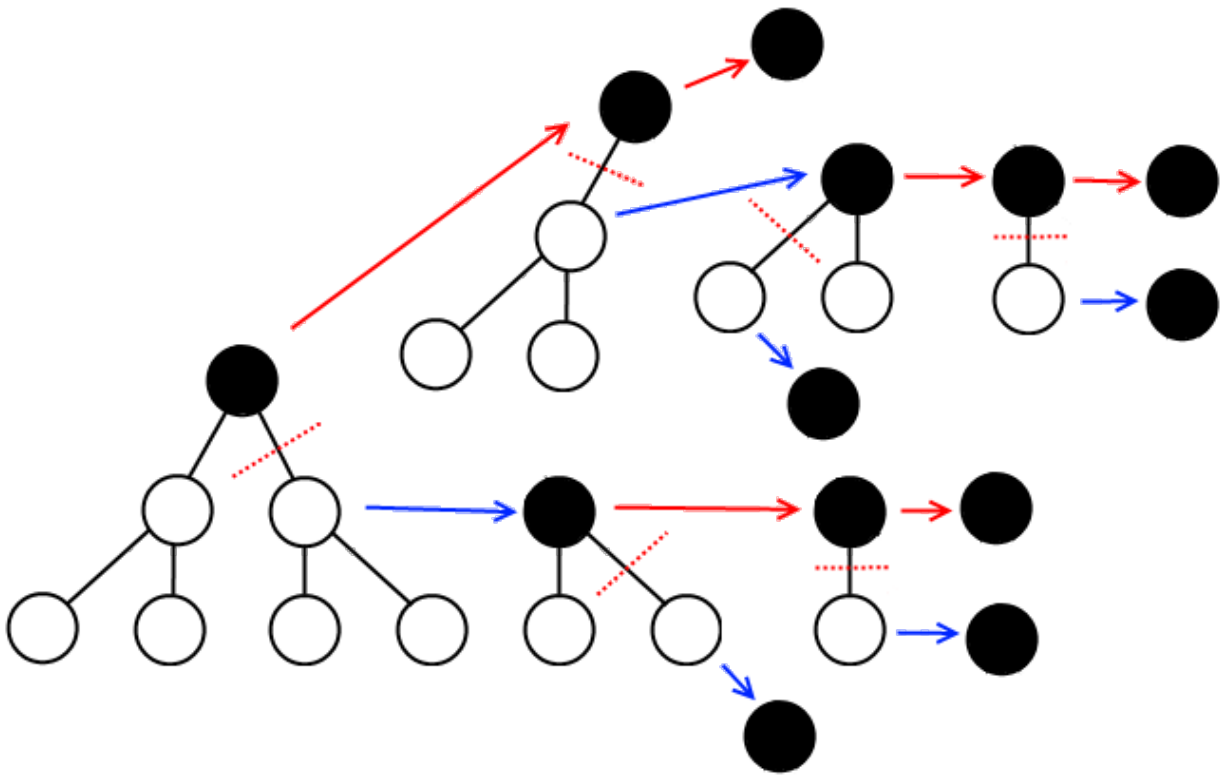
- Basic partitioning: try to evenly partition template



FASCIA

Template partitioning and work avoidance

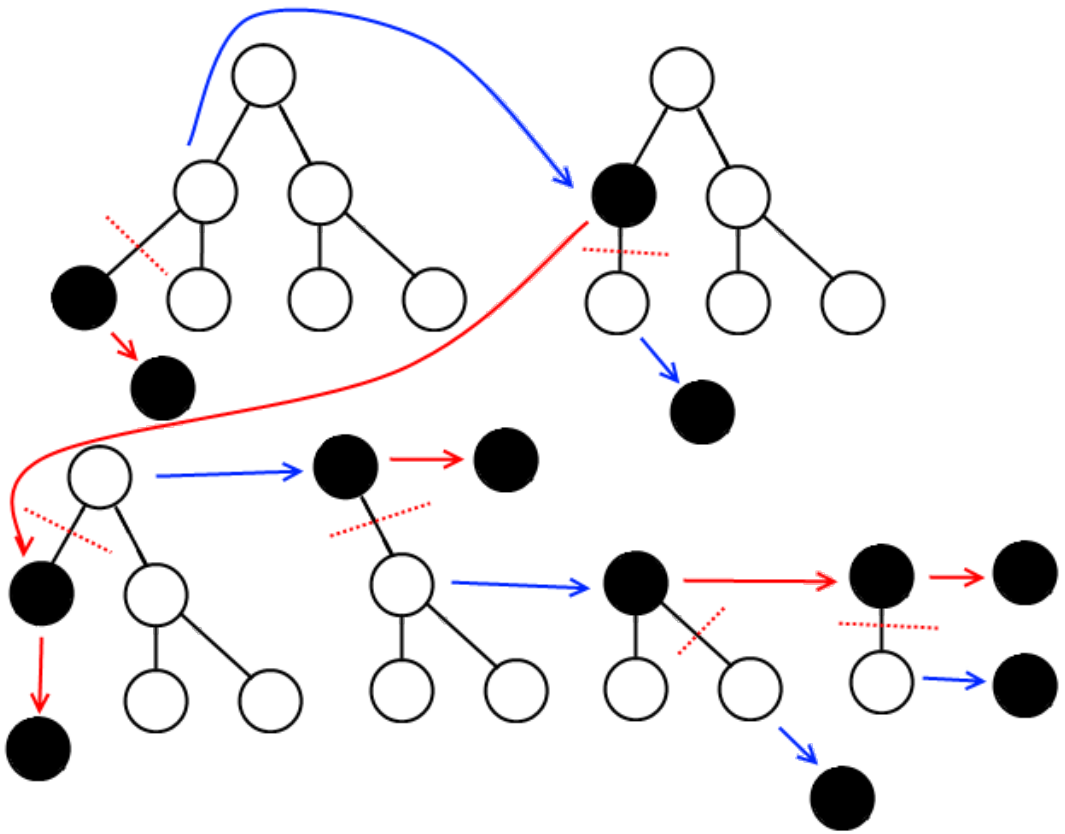
- Basic partitioning: try to evenly partition template
- Observation: algorithm runtime is proportional to $\sum_i \binom{k}{S_i} \cdot \binom{S_i}{a_i}$, i.e. $\sum_i |C_i| \cdot |C_{a_i}|$



FASCIA

Template partitioning and work avoidance

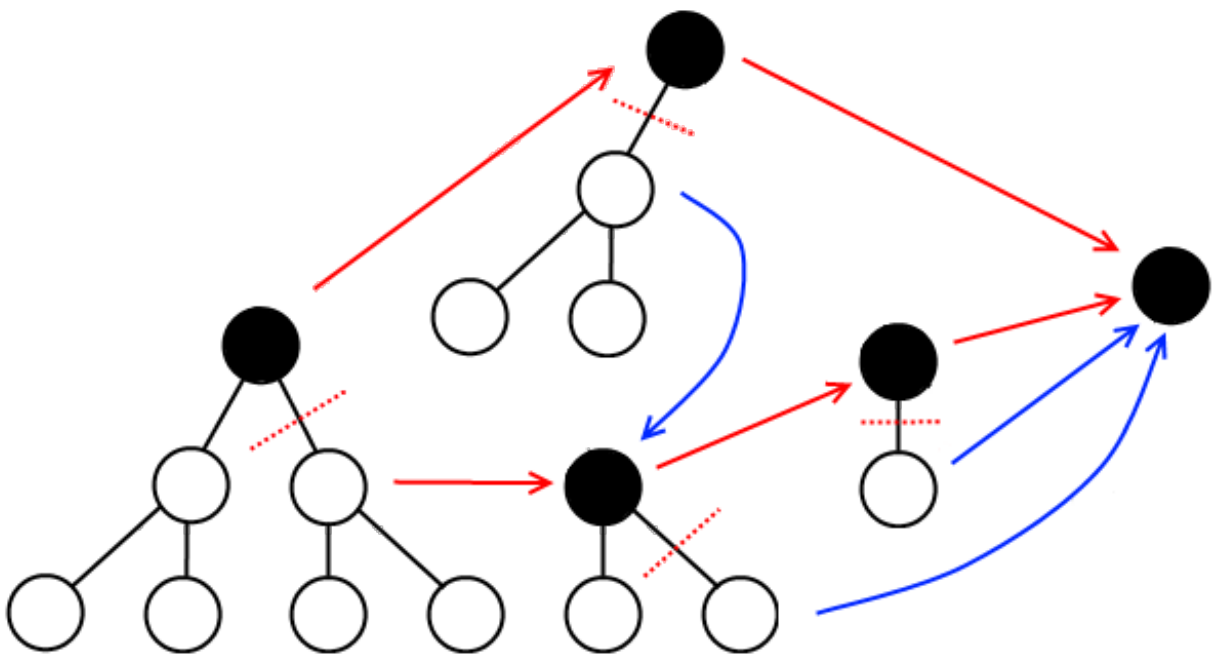
- Basic partitioning: try to evenly partition template
- Observation: algorithm runtime is proportional to $\sum_i \binom{k}{S_i} \cdot \binom{S_i}{a_i}$, i.e. $\sum_i |C_i| \cdot |C_{a_i}|$
- This sum can be minimized by a one-at-a-time partitioning approach



FASCIA

Template partitioning and work avoidance

- Basic partitioning: try to evenly partition template
- Observation: algorithm runtime is proportional to $\sum_i \binom{k}{S_i} \cdot \binom{S_i}{a_i}$, i.e. $\sum_i |C_i| \cdot |C_{a_i}|$
- This sum can be minimized by a one-at-a-time partitioning approach
- On certain templates, this sum can be minimized by exploiting latent symmetry, HOWEVER ...



FASCIA

Template partitioning, work reduction

- With a single node active or passive child we can do $\sim\frac{1}{k}$ of the original work, $C_a = color(v)$ *or* $C_p = color(u)$
- We can avoid all work for a given v if (table[a][v][] == NULL), for any arbitrary subtemplate
- Can also minimize the size of $N(v)$ by checking the initialization for each possible u
- Finally, we can order the way in which we process all S_i to minimize memory usage

```
1:  for all S_i created during partitioning, a_i and p_i children do
2:      for all v ∈ G, where table[a_i][v] != NULL do
3:          for all C possibly mapped to S_i do
4:              for all C_p, C_a from C, where C_a, C_p is fixed if |a_i| = 1 or |p_i| = 1 do
5:                  for all u ∈ N(v), where table[p_i][u] != NULL do
6:                      count += table[a_i][v][C_a]·table[p_i][u][C_p]
7:                  end for
8:              end for
9:              Set table[S_i][v][C] := count
10:         end for
11:     end for
12: end for
```

Results
Overview

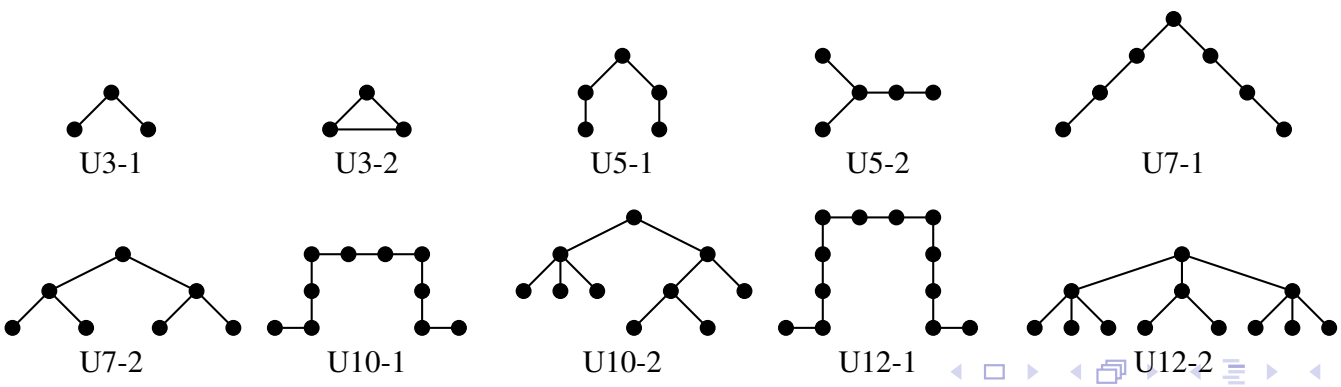
- Networks and templates analyzed
- Large network runtimes
- Approximation Error in template counts
- Memory savings
- Strong scaling

Results

Networks and templates analyzed

- Gordon at SDSC, 2× Xeon E5 @ 2.6GHz (Sandy Bridge), 64 GB DDR3
- Database of Interacting Proteins, SNAP, and Virginia Tech NDSSL

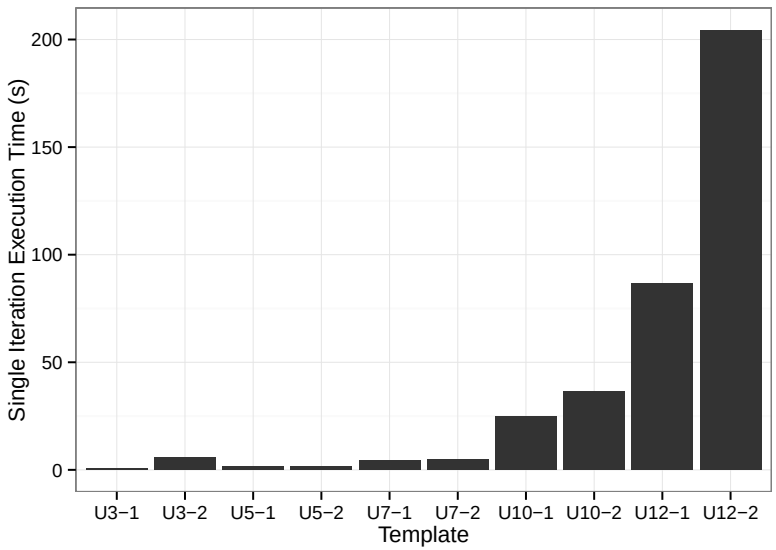
Network	n	m	d_{avg}	d_{max}
Portland	1,588,212	31,204,286	39.3	275
PA Road Net	1,090,917	1,541,898	2.8	9
Slashdot	82,168	438,643	10.7	2510
Enron	33,696	180,811	10.7	1383
E. coli	2,546	11,520	9.0	178
S. cerevisiae	5,021	22,119	8.8	289
H. pylori	687	1,352	3.9	54
C. elegans	2,391	3,831	3.2	187

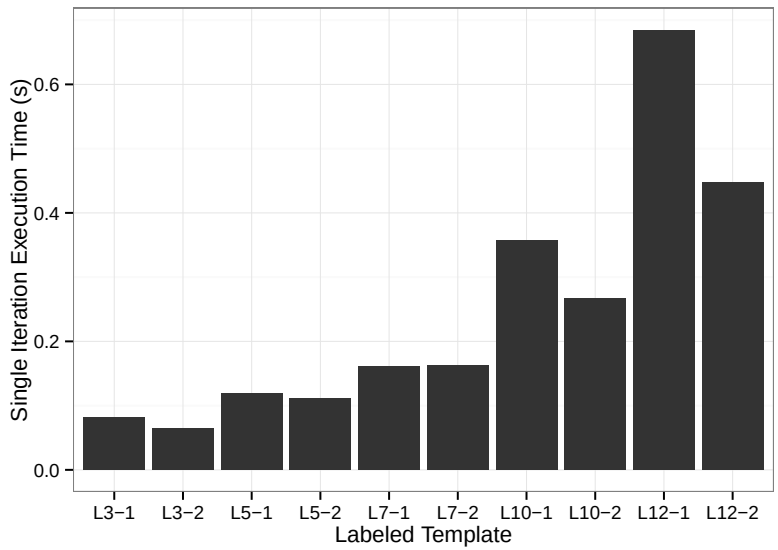


Results

Large network runtimes

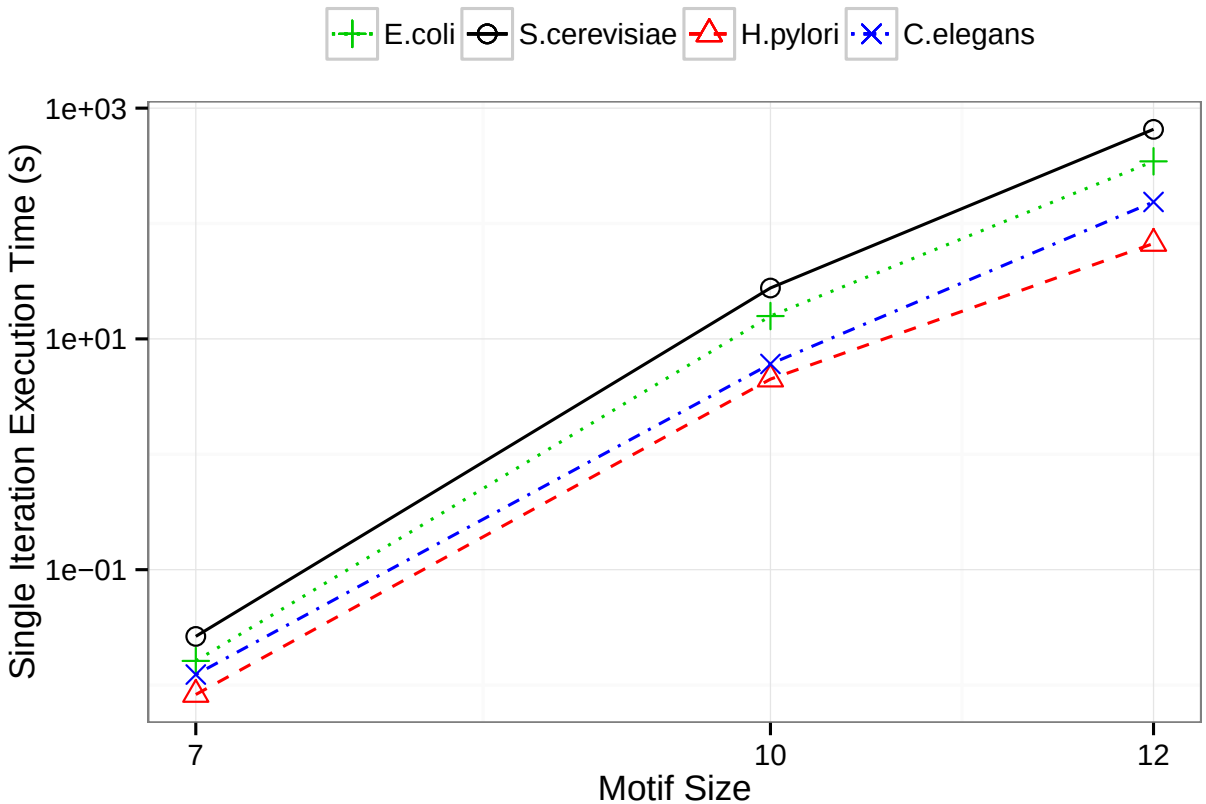
- Parallel (16 cores) results for all unlabeled (left) and labeled (right) templates on Portland network
- 8 possible demographic labels (M/F and kid/youth/adult/senior)
- ~200 seconds for up to 12 vertex unlabeled template, less than 1 second for all labeled templates





Results
Motif finding runtimes

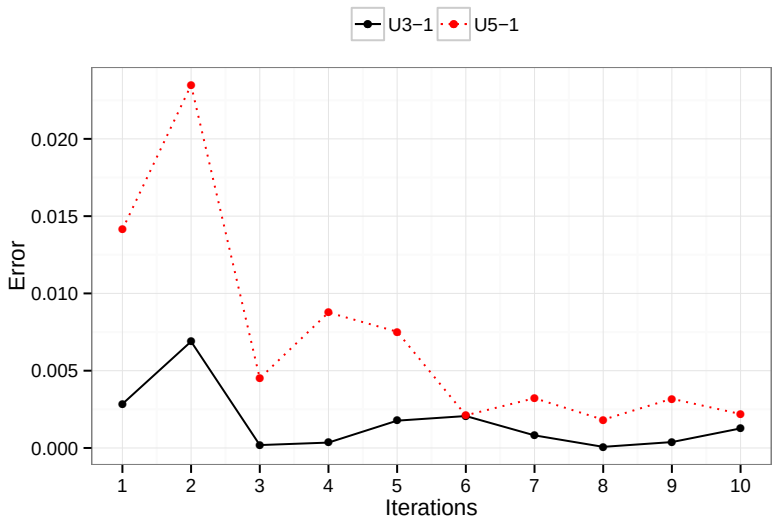
- Parallel (16 cores) runtimes for a single iteration of all 7, 10, and 12 vertex templates on the four biological PPI networks
- $7 \rightarrow 11$ templates, $10 \rightarrow 106$ templates, $12 \rightarrow 556$ templates

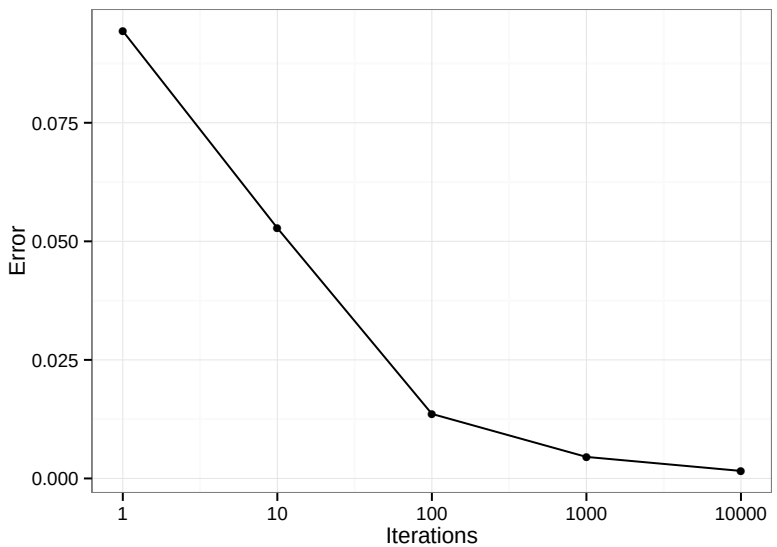


Results

Approximation Error in template counts

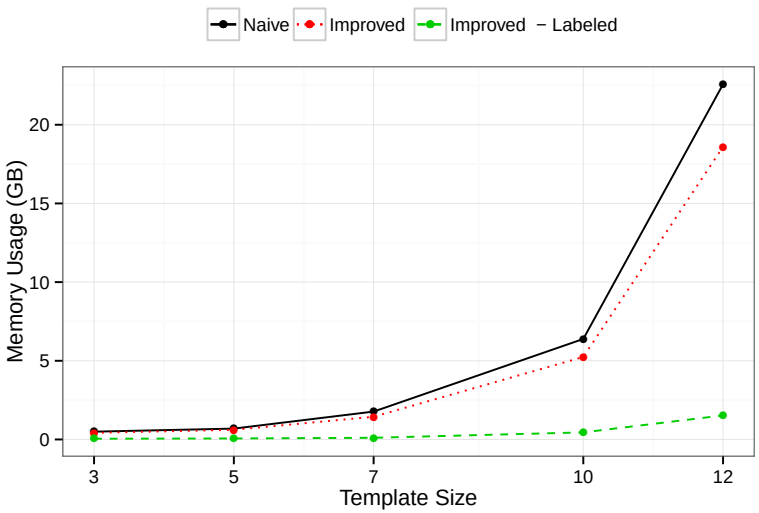
- Enron (left) with U3-1 and U5-1 and *H. Pylori* (right) across all 11 unique templates of size 7
- Error increases with template size and inversely to network size



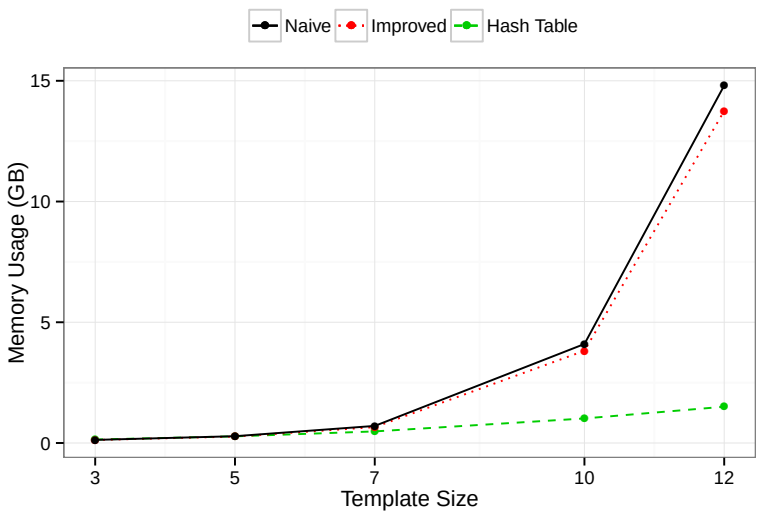


Results
Memory savings

- Memory requirements on Portland and PA Road network for improved array (left) and hash table (right)
- Using all UX-1 chain templates
- Up to 20%-90% savings versus naïve method



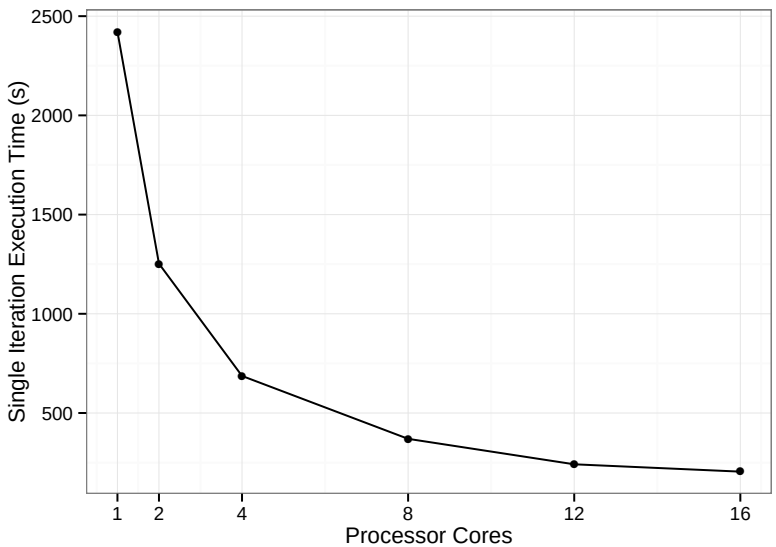
Portland Network with Improved Array

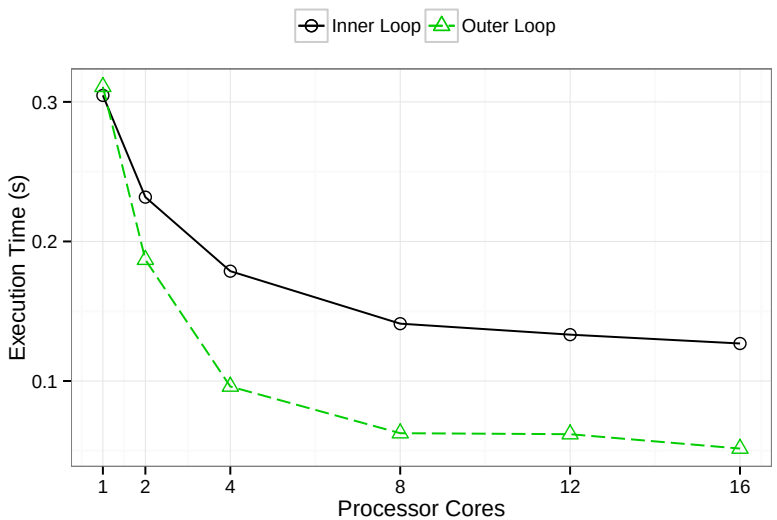


PA Road Network with Hash Table

Results
Strong Scaling

- Inner loop for large graphs (**forall** $v \in G$)
- Outer loop for small graphs (**for** $i = 1$ to $Niter$)
- U12-2 Template on Portland (left) and Enron (right), 16 threads
- About 12× speedup for inner loop on Portland
- About 6× speedup for outer loop on Enron, 3.5× for inner loop





Conclusions and Future (Ongoing) Work
FASCIA

- Memory and algorithmic improvements allow FASCIA to achieve considerable speedup relative to previous work
- Real time count estimate for up to 7 vertex templates on large networks
- Algorithm provides reasonable error after a low number of iterations
- Further optimization for count storage (CSR-like structure)
- Distribution via MPI, graph partitioning to allow larger counts

Bibliography

N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S.C. Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249, 2008.

S. Omidi, F. Schreiber, and A. Masoudi-Nejad. MODA: an efficient algorithm for network motif discovery in biological networks. *Genes Genet Syst*, 84(5):385–395, 2009.

Z. Zhao, M. Khan, V.S.A. Kumar, and M.V. Marathe. Subgraph enumeration in large social contact networks using parallel color coding and streaming. In *Proc. 39th Int'l. Conf. on Parallel Processing (ICPP '10)*, pages 594–603, 2010.

Z. Zhao, G. Wang, A.R. Butt, M. Khan, V.S.A. Kumar, and M.V. Marathe. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proc. 26th IEEE Int'l. Parallel and Distributed Processing Symp. (IPDPS)*, pages 390–401, may 2012.