Scalable Subgraph Counting: Methods Behind the Madness

C. Seshadhri
(UC Santa Cruz)

Srikanta Tirthapura
(Iowa State Univ)

WWW 2019 tutorial

Supported by NSF TRIPODS grant CCF-1740850, CCF-1813165, IIS-1527541 and CCF-1725702
Subgraph Counting: The Problem

- G is a large graph (the input)
- H is a small “pattern” graph
- Count/find all occurrences of H in G
- Other names: graphlet analysis, motif counting
Why subgraph counting
The significance of triads

- [Holland-Leinhardt Am. J. Soc. 70, Sherwin World Int. Sur. 71] “the equipment is now ready for deriving the algorithms for finding balanced and unbalanced triads in a matrix composed of positive and negative relations”
Why Subgraph Counting: Clustering coefficients


- $W =$ no. of wedges (paths of length 2)
- $T =$ no. of triangles
- Transitivity $= \tau = 3T/W$
  - Sometimes “global clustering coefficient”
Local Clustering coefficients

- Clustering coeff. of $v = cc_v = (\text{triangles at } v)/(\text{wedges centered at } v)$
  - How often are two of your friends, also friends?
- Degree-wise $cc_d = \text{average } cc \text{ of degree } d \text{ vertices}$
- Annoyingly, sometimes, global $cc = \sum_v cc_v/(\# \text{ vertices})$

$cc_A = 2/3$  $cc_3 = 1/3 \times (2/3 + 2/3 + 0) = 4/9$

[Watts-Strogatz Science 99]
### Some numbers

<table>
<thead>
<tr>
<th>Graph</th>
<th>n</th>
<th>m</th>
<th>T</th>
<th>$\tau = 3T/W$</th>
<th>cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca-AstroPh</td>
<td>18K</td>
<td>200K</td>
<td>1.3M</td>
<td>0.32</td>
<td>0.63</td>
</tr>
<tr>
<td>web-Google</td>
<td>880K</td>
<td>4.3M</td>
<td>13M</td>
<td>0.06</td>
<td>0.51</td>
</tr>
<tr>
<td>as-skitter</td>
<td>1.7M</td>
<td>11M</td>
<td>29M</td>
<td>0.01</td>
<td>0.26</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>3.8M</td>
<td>17M</td>
<td>7.5M</td>
<td>0.07</td>
<td>0.08</td>
</tr>
<tr>
<td>soc-LiveJournal</td>
<td>4.9M</td>
<td>86M</td>
<td>286M</td>
<td>0.12</td>
<td>0.27</td>
</tr>
<tr>
<td>com-Orkut</td>
<td>3.1M</td>
<td>234M</td>
<td>628M</td>
<td>0.04</td>
<td>0.17</td>
</tr>
</tbody>
</table>

For Erdos-Renyi graph, transitivity/clustering coefficient less than $10^{-5}$
Plots of degree-wise clustering coefficients

- Degree-wise clustering coefficients: average clustering coefficient of degree \( d \) vertices
- Powerful modeling input, very common network analysis measure
Modeling uses

- [Sala-Cao-Wilson-Zablit-Zheng-Zhao WWW 10, Seshadhri-Pinar-Kolda PRE 12, Pfeiffer-Moreno-Neville PASSAT12, Pfeiffer-Moreno-LaFond-Neville-Gallagher WWW 14] Models fail to capture clustering coefficients; very hard to model accurately
Graph comparison

[Milo Shen-Orr, Itzkovitz, Kashtan, Chklovskii, Alon Science 02, Pržulj Bioinf. 07] Motif counts, graphlet degree distribution

WWW 2019 tutorial
Graph summaries: the graphlet kernel

Consider counts of all subgraphs of size $k$, $k=4, 5$

$G_1$  $G_2$

[Shervashidze-Vishwanathan-Petri-Mehlhorn-Borgwardt AISTATS09]: Graphlet kernel

$4D$ vector of $3$-vertex subgraph counts

$6D$ vector of connected $4$-vertex counts

$10$  $24$  $18$  $4$
Local subgraph counting

- Local counts. Also, subgraph counting in the neighborhoods
- Gives “features” that can used for some downstream task
  - Role discovery, community detection, higher order statistics

How many triangles to these participate in?
How many 4-cycles in this edge in?
Local triadic measures

No triangles here!
So vertex has social capital

- [Burt Am. J. Soc. 04] Structural holes
From 4-cycles to weak ties

- [Rotabi-Kamath-Kleinberg-Sharma, WWW17] Weak ties from 4-cycles
More on larger subgraphs

• [Ugander-Backstrom-Kleinberg, WWW13] Social network structure and models through 4-vertex subgraph counts

• [Yin-Benson-Leskovec, PRE18, WSDM19] Higher order clustering coefficients
Community detection

- [Berry-Hendrickson-LaViolette-Phillips, PRE11] Using 4-cycles for edge weights, for better community detection
- [Benson-Gleich-Leskovec, Science16], [Tsourakakis-Pachocki-Mitzenmacher WWW17] Using triangle counts as weights, generalize to “motif conductance”
Dense subgraph discovery

Maximize internal edges

Maximize internal cliques

- [Sariyuce-Seshadhri-Pinar-Catalyurek, WWW15], [Tsourakakis WWW15]

Using 3, 4-cliques counts as vertex weights, generalize to “clique degrees”
Why Subgraph Counting: Nearest Neighbor Search

Why have a tutorial?
Important to network analysis

- Common network analysis technique, with rich history
- Hopefully, the varied applications convinced you

- Surprisingly clean formulations of core problem
- Algorithmically challenging
Rich literature

- In the past decade, many tens papers on just triangle counting, in data mining/databases venues
- Many tens of papers involving subgraph counting in past three years of WWW+KDD
- Tutorial refers to > 30 practical algorithmic papers
Algorithmic methods for subgraph counting

• Focus on algorithms for subgraph counting
• A few methods, that capture many results
  • So tutorial is structured around the methods, not the algorithms/results
For practitioners

- Tell you some of the main tools
  - In many cases, you can tailor existing algorithms easily
- Point out the relevant results/codes
- Do give us suggestions on how to make codes relevant!

Just tell me what to run!
For researchers in area

• Try to summarize main results
  • So you know what problems are open
• Lot of us (including authors) not comparing with all previous work
• Point out the challenges, and ideas/themes that have emerged

Just tell me what to work on!
The formal problem, and variants
What is an occurrence/match?

1. Subset of **vertices**, whose internal edges exactly form H
   - **Induced** H-subgraph. Total count is number of vertex subsets

2. Subset of **edges** that form H
   - **Non-induced** H-subgraph. Total count is number of edge subsets

Any set of 5 edges give non-induced H
Induced or non-induced subgraph?

• **Induced** subgraph: must match non-edges as well
• **Non-induced** subgraph: must contain H
  • Formalized by graph homomorphisms
• Typically: non-induced easier to count, but induced is more useful
Induced vs non-induced

- The ratio induced/non-induced for various patterns
  - For random graphs, should be nearly 1
- Informative to look at induced numbers, to detect "unstable" patterns

<table>
<thead>
<tr>
<th>Graph</th>
<th>Four-cycle</th>
<th>3-wedge</th>
<th>4-wheel</th>
<th>5-almost clique</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca-AstroPh</td>
<td>0.03</td>
<td>&lt; 0.01</td>
<td>&lt; 0.01</td>
<td>0.08</td>
</tr>
<tr>
<td>web-Google</td>
<td>0.07</td>
<td>0.04</td>
<td>0.02</td>
<td>0.39</td>
</tr>
<tr>
<td>as-skitter</td>
<td>0.68</td>
<td>0.68</td>
<td>0.15</td>
<td>0.76</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>0.79</td>
<td>0.89</td>
<td>0.44</td>
<td>0.57</td>
</tr>
<tr>
<td>soc-LiveJournal</td>
<td>0.10</td>
<td>0.03</td>
<td>0.01</td>
<td>0.12</td>
</tr>
</tbody>
</table>
Induced vs non-induced

For any size k: Invertible linear transformation between vectors of induced and non-induced counts

- For k=4, it is 11X11 matrix
  - Each row is subgraph count, for a subgraph

- Convenient to count non-induced, and then get induced
  - “Equivalence” messier if one looks at approximate counts
Connected vs disconnected

• [Pinar-Seshadhri-Vishal 17] For any size k: All counts can be obtained from connected upto k-vertex subgraph counts
  •Disconnected counts are polynomial function of connected counts
  •Disconnected counts typically very large, not as interesting

• Get non-induced ≤ k-vertex connected subgraph counts
  •End up with induced ≤ k-vertex all subgraph counts

• “Graphlet” often means connected induced subgraph (CIS)
Counting vs. Enumeration

- Counting: count the number of H-subgraphs in G
  - Can think of approximate counting
- Enumeration/list: output all H-subgraphs

- In some cases, counting is faster
  - \# non-induced wedges = \( \sum_v \binom{d_v}{2} \)
Local Counting

• For every $v$, count number of $H$-subgraphs $v$ participates in
  • Local counts, subgraph profiles, orbit counting

• Summarize this information, by averaging over degree, etc.
  • Clustering coefficient plots

• Counting in all neighborhoods $\implies$ Local counts

• Enumeration $\implies$ Local counts

• Can we do better? Approximate?
Local Counting: Orbits

• The count depends on “role” in H
  • Formalized through automorphism group of H, and vertex orbits under automorphisms

• So we ask for count of specific orbit
  • Within subgraph, some orbits can be easier

• Great source of features for machine learning, role discovery

• Not as many results here...
2-node graphlet

3-node graphlets

4-node graphlets

5-node graphlets

[Pržulj Bioinf. 07]
Attributed vertices/edges

• Vertices have colors, edges have direction/timestamp
  • [Wang-Fu-Cheng ICBD14, SaneiMehri-Sarıyüce-Tirthapura KDD18] Bipartite graphs

• The number of possible subgraphs explodes

• Not as many results here...
Special cases of interest

• Triangle: most well-studied case
• Cycles: 4-cycles appearing increasingly in literature
• Small-cliques (k < 10)
What we won’t talk about

• General subgraph isomorphism (or graph isomorphism)
• Techniques tend to be different
What we won’t talk about

• Clique counting for large sizes
  • Dense subgraph discovery

• Shortest path, centrality problems

• Focus on $k = 3, 4, 5, \ldots < 9$
• Plethora of models, many results, hard to summarize
The tutorial’s approach

• Classic RAM model (single processor, everything in memory)
  • Also, streaming model
  • These cover most of the techniques

• Surprisingly effective algorithms
  • Enumerate all triangles in > 100M edge graph, in 4 minutes on laptop

• Is your data really that big?
  • Graphs of < 100M edges capture a lot of relevant research, and are still challenging
The tutorial’s approach

• Often, core algorithmic methods are similar across models, and originate from RAM model algorithms
  • Challenge is to implement method in “big data” model

• Each “section” of tutorial talks of different method
  • Try to cover usage of method in alternate computational models
Why is this hard?
The sheer numbers of subgraphs

- Typically, triangles $>>$ edges by an order of magnitude
  - Looking at super-linear time
- Beyond a point, enumeration is almost impossible
The sheer numbers of subgraphs

Connected 4-vertex subgraphs
Brute force fails

• Wedge enumeration an obvious method for triangle counting

• Clearly super-linear, significant overhead over triangle count
  • Not clear how to avoid “wasted work”
  • Complexity theory suggests this is unavoidable

• Curse of heavy tailed degree distribution
  • Parallelizing non-trivial
  • Problem easy if all degrees bounded
Sampling is non-trivial

- Naïve sampling has large variance
  - Sample edge + 2 vertices. Likelihood of 4-clique is $10^{-10}$
- Curse of sparsity
  - Simple concentration inequalities fail
  - Approximation easy if graph is dense (property testing in TCS, graphons)
The methods
Techniques behind the results

Key common methods, behind many of the results

1. Graph orientations
2. Subgraph reconstruction
3. Color coding
4. Edge sampling
5. Substructure Sampling
Mea culpa

• We forgot to mention your paper

• We didn’t cite your paper in the right places

• We ignored an important facet of subgraph counting

• You don’t like our terminology

• Please talk to us!
Graph orientations

A classic idea that hasn’t aged (yet)
Triangle counting: wedge enumeration

- Simplest, brute force algorithm
- For all vertices $u$:
  - For all pairs of neighbors $v$, $w$: (this is a wedge)
    - Check if $(v,w)$ is edge
Analysis of wedge enumeration

- An edge lookup for every wedge
- Number of edge lookups = \( \sum_v \binom{d_v}{2} \approx \sum_v d_v^2 / 2 \)
  - Heavy tailed degree distribution hurts
  - In Orkut graph (117M edges), 45B wedges

Potential triangles

For all vertices \( u \):
   For all pairs of neighbors \( v, w \):
      Check if \( (v,w) \) is edge
Take acyclic orientation

- Orient graph into DAG. Direct edges, so no directed cycle forms
- Order nodes in increasing “priority”. Edges directed to higher priority
  - Any permutation of nodes is a valid priority/ordering
Degree orientation

- [Chiba-Nishizeki SICOMP 85] Degree orientation. Priority = degree
- Out-degree < degree
- So improvement in triangle counting...?
For all vertices $u$: 
For all pairs of out-neighbors $v, w$: 
Check if $(v, w)$ is edge

- Number of edge lookups $= \sum_v (d_v^+) \cdot (d_v^+) \cdot (d_v^+)$ for the number of neighbors with higher degree
- [Chiba-Nishizeki SICOMP85] Theorem: $O(m\alpha)$ runtime

Edges point towards center, reducing its degree
The improvement

- Degree orientations give significant improvement over vanilla wedge counting

<table>
<thead>
<tr>
<th>Graph</th>
<th>Wedges</th>
<th>Out-wedges</th>
<th>Wedges/Out-wedges</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca-Astro</td>
<td>1.3E7</td>
<td>2.0E6</td>
<td>6.9</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>3.4E8</td>
<td>5.1E7</td>
<td>6.6</td>
</tr>
<tr>
<td>web-Google</td>
<td>7.2E8</td>
<td>1.7E7</td>
<td>42</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>7.3E9</td>
<td>6.7E8</td>
<td>10.9</td>
</tr>
<tr>
<td>as-skitter</td>
<td>1.6E10</td>
<td>9.5E7</td>
<td>168</td>
</tr>
<tr>
<td>com-Orkut</td>
<td>4.6E10</td>
<td>4.0E9</td>
<td>11.5</td>
</tr>
</tbody>
</table>
Out-degree distributions

- Taming the heavy tail
Degeneracy orientations

- Remove min degree vertex
Degeneracy orientations

- Iteratively remove min degree vertex
Degeneracy orientations

- Iteratively remove min degree vertex
- This is the k-core decomposition
- Degeneracy orientation: set priority to removal time
Degeneracy orientation

- [Chrobak-Eppstein TCS91] Priority = removal time
- [Matula-Beck JACM 83] Linear time algorithm
- max out degree is $\alpha = \text{graph degeneracy} \ (\text{also max core number})$
Degeneracy orientations

Potential triangles

Until graph is empty:
Remove min degree vertex
(Update all degrees)

Edges point towards center, reducing its degree

• [Chrobak-Eppstein TCS 91], [Schank-Wagner Exp. Eff. Alg. 05] Triangle counting with degeneracy

\[
\sum_v (d^+_v)^2 \leq \alpha \sum_v d_v = O(m\alpha)
\]
Edge iteration

For every edge (u,v):
(Find common neighbors of u, v)
Look up every neighbor of smaller degree vertex in other list

• Edge lookups = $\sum_v \min(d_u, d_v)$
• Analogous to orienting edge from u to v
• Analysis similar, though cache performance might be better

• [Schank-Wagner Exp. Eff. Alg. 05], [Wagner Thesis 05]
Degree and degeneracy orientations

<table>
<thead>
<tr>
<th>Graph</th>
<th>Wedges</th>
<th>Out-wedges (degree)</th>
<th>Out-wedges (degeneracy)</th>
<th>Wedges/Out-wedges (degree)</th>
<th>Wedges/Out-wedges (degeneracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca-Astro</td>
<td>1.3E7</td>
<td>2.0E6</td>
<td>1.9E6</td>
<td>6.9</td>
<td>6.8</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>3.4E8</td>
<td>5.1E7</td>
<td>4.4E7</td>
<td>6.6</td>
<td>7.7</td>
</tr>
<tr>
<td>web-Google</td>
<td>7.2E8</td>
<td>1.7E7</td>
<td>1.6E7</td>
<td>42</td>
<td>45</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>7.3E9</td>
<td>6.7E8</td>
<td>6.4E8</td>
<td>10.9</td>
<td>11.4</td>
</tr>
<tr>
<td>as-skitter</td>
<td>1.6E10</td>
<td>9.5E7</td>
<td>8.7E7</td>
<td>168</td>
<td>184</td>
</tr>
<tr>
<td>com-Orkut</td>
<td>4.6E10</td>
<td>4.0E9</td>
<td>3.8E9</td>
<td>11.5</td>
<td>12.1</td>
</tr>
</tbody>
</table>

- Excellent, practical technique. Go to algorithm for triangle enumeration/exact counting
  - Hard to beat. Fancy matrix methods usually worse
  - Took largest graph from SNAP (com-orkut, 117M edges). Counted 628M triangles in 4.5 min on laptop
Degree vs degeneracy orientations

- Max outdegree smaller for degeneracy (provably so)
Representing your graph

- Adjacency list: collection of neighbor arrays
- What Matlab does: store concatenation of adjacency list, as single list
- Put pointers to start of individual list
Data structure: the CSR representation

- If you store as CSR, your algorithms will be faster. Cache locality great
- Can store DAG as CSR
- And you find additional tricks to speed things up
- [Eppstein-Strash SEA13, Danisch-Balalau-Sozio WWW18] Reordering with neighbor array
Degree orientations

• Can be parallelized/distributed
  • [Cohen Comp. S&E09, Suri-Vassilvitskii WWW11, Arifuzzaman-Khan-Marathe CIKM13]

Map-Reduce/distributed implementations

Figure out who “owns” the edge by degree comparisons
The distributed degree orientation

• [Cohen Comp. S&E09, Suri-Vassilvitskii WWW11] Map-Reduce implementation
  • Use sampling for load balancing

• [Arifuzzaman-Khan-Marathe CIKM13] MPI implementation, use $d_v^{+}$ for load balancing when partitioning graph

Round 1: Communicate vertices over edges

Round 2: Find lower degree vertex “owning” this edge

Round 3: Check pairs of “owned” edges for triangles
Computing degeneracy orientations

• Requires sequential computation
  • Some work on distributed computation (not worth it for triangle counting?)
• Max degree is bounded by max-core number (and quite small)

Until graph is empty:
Remove min degree vertex
(Update all degrees)
Degree vs degeneracy orientations

• Degree: Easier, parallel, but max out-degree can be larger
• Degeneracy: Harder, not parallel, but max is smaller
  • Typically max is 2-3 times smaller

<table>
<thead>
<tr>
<th>Graph</th>
<th>Max degree</th>
<th>Max outdegree (Degree orient)</th>
<th>Max outdegree (Degen orient)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca-Astro</td>
<td>504</td>
<td>66</td>
<td>56</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>793</td>
<td>77</td>
<td>64</td>
</tr>
<tr>
<td>web-Google</td>
<td>6332</td>
<td>93</td>
<td>44</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>20333</td>
<td>686</td>
<td>372</td>
</tr>
<tr>
<td>as-skitter</td>
<td>35455</td>
<td>231</td>
<td>111</td>
</tr>
<tr>
<td>com-Orkut</td>
<td>33313</td>
<td>535</td>
<td>253</td>
</tr>
</tbody>
</table>
Graph orientations: clique counting
k-Clique counting

- Simple backtracking/recursive procedure
- In all neighborhoods, recursively count (k-1)-cliques
Take acyclic orientation

• [Chiba-Nishizeki SICOMP 85] Using orientation for k-clique counting
• Consider priority viewpoint: in every clique, some vertex has lowest priority
  • That vertex “responsible” for finding this clique

Find all \((k-1)\)-cliques, in all out-neighborhoods

\(N^+_v: \) out-neighborhood of \(v\)
Triangle counting as 3-clique counting

For all vertices $u$:
   For all pairs of out-neighbors $v, w$:
      Check if $(v, w)$ is 2-clique

- Finding 2-cliques (edges) in all out-neighborhoods
- Number of edge lookups $= \sum_v \binom{d_v^+}{2}$
Take acyclic orientation

• [Chiba-Nishizeki SICOMP 85] Use degree orientation to convert graph into a DAG.
• Recursively count (k-1)-cliques in all out-neighborhoods
• Total number of edge lookups = $O(m\alpha^{k-2})$, generalizing triangle listing
Doing distributed

• Use degree orientation to convert graph into a DAG.
• Recursively count \((k-1)\)-cliques in all \textit{out}-neighborhoods
• [Danisch-Balalau-Sozio WWW18] Multi-threaded implementation, with excellent behavior, up to \(k < 10\)

Each is an independent problem, so easy to parallelize
Degree orientations are quick to find within subproblems
On parallelizing

- [Danisch-Balalau-Sozio WWW18] Multi-threaded implementation, but not so simple
  - Vertex based neighbors don’t parallelize well
Edge-based subproblems

• [Danisch-Balalau-Sozio WWW18] Parallelize edge-based subproblems

• On Orkut social network (117M edges), can exactly count $\sim 10^{13}$ 10-cliques in 8 hours, with 40 threads (single machine)
Benefits of degeneracy

- Use degeneracy orientation to convert graph into a DAG.
- Not convenient for recursion, since finding degeneracy orientation adds overhead
- But each subproblem is small

<table>
<thead>
<tr>
<th>Graph</th>
<th>Max degree</th>
<th>Max outdegree (Degree orient)</th>
<th>α (Degen orient)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca-Astro</td>
<td>504</td>
<td>66</td>
<td>56</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>793</td>
<td>77</td>
<td>64</td>
</tr>
<tr>
<td>web-Google</td>
<td>6332</td>
<td>93</td>
<td>44</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>20333</td>
<td>686</td>
<td>372</td>
</tr>
<tr>
<td>as-skitter</td>
<td>35455</td>
<td>231</td>
<td>111</td>
</tr>
<tr>
<td>com-Orkut</td>
<td>33313</td>
<td>535</td>
<td>253</td>
</tr>
</tbody>
</table>
Benefits of degeneracy

- Use degeneracy orientation to convert graph into a DAG.
  - Each subproblem/neighborhood is small
- Reduce 1 problem with $n$ vertices, to $n$ problems with $\leq \alpha$ vertices
  - Dependence on $\alpha$ in running time
Benefits of degeneracy

• Use degeneracy orientation to convert graph into a DAG.
  • Each subproblem is small
• [Eppstein-Loeffler-Strash JEA13, Finocchi-Finocchi-Fusco JEA15, Jain-Seshadhri WWW17]
  • Combining methods
The upshot

• If you’re doing clique counting, use orientations!
  • Degree or degeneracy, whichever is convenient
  • With CSRs, a very very efficient baseline
  • Recursion
Graph orientations: are we not done yet?
The 4-cycle


• Enumerate all wedges, to find $W_{u,v}$ values

\[ \sum_{u,v} \left( \frac{W_{u,v}}{2} \right) \]
Take acyclic orientation

- Orient graph into DAG. Direct edges, so no directed cycle forms.
- Order nodes in increasing “priority”. Edges directed to higher priority.
  - Any permutation of nodes is a valid priority/ordering.
Using orientations

- Each 4-cycle becomes an acyclic 4-cycle
- Unlike triangles, there are many types of acyclic 4-cycles
4-cycle DAGs
4-cycle DAGs

- [Cohen Comp. S&E09, Pinar-Seshadhri-Vishal WWW17] Look at non-isomorphic acyclic orientations of 4-cycle
  - [Chiba-Nishizeki SICOMP85] have messier version of idea
So...

• Most wedges of in-in type, so huge savings!

Only need to enumerate these

Edges point towards center, reducing its degree
Why this works: the numbers

<table>
<thead>
<tr>
<th>Graph</th>
<th>Wedges</th>
<th>Out-out wedge (degree orient)</th>
<th>In-out wedges (degree orient)</th>
<th>Wedges/Out-wedges (degree)</th>
<th>Wedges/(Out-out + in-out wedges)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca-Astro</td>
<td>1.3E7</td>
<td>2.0E6</td>
<td>3.8E6</td>
<td>6.9</td>
<td>2.2</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>3.4E8</td>
<td>5.1E7</td>
<td>9.1E7</td>
<td>6.6</td>
<td>2.4</td>
</tr>
<tr>
<td>web-Google</td>
<td>7.2E8</td>
<td>1.7E7</td>
<td>2.1E7</td>
<td>42</td>
<td>19</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>7.3E9</td>
<td>6.7E8</td>
<td>1.4E9</td>
<td>11</td>
<td>3.5</td>
</tr>
<tr>
<td>as-skitter</td>
<td>1.6E10</td>
<td>9.5E7</td>
<td>2.1E8</td>
<td>168</td>
<td>52</td>
</tr>
<tr>
<td>com-Orkut</td>
<td>4.6E10</td>
<td>4.0E9</td>
<td>8.6E9</td>
<td>11.5</td>
<td>3.8</td>
</tr>
</tbody>
</table>

- **[Pinar-Seshadhri-Vishal WWW17]** Careful implementation generates all local counts with low memory overhead
  - LiveJournal graph with 84M edges, counted 51B 4-cycles in 2 min on laptop
Beyond 4-cycles

- [Pinar-Seshadhri-Vishal WWW17] Works for 5-cycles, much more complicated
  - Seems practically useful for cycles and cliques
- [Ossana de Mendez-Nesetril 12] Theory of graph sparsity, and why orientations are so useful for subgraph counting
The upshot

• If you’re doing clique or cycle counting, use orientations!
  • Degree or degeneracy, whichever is convenient
  • With CSRs, a very efficient baseline (or new result)
• Parallel/distributed/streaming anyone?

Research opportunity!
Codes

• [Pinar-Seshadhri-Vishal WWW17] Triangle and 4-cycle counting using orientations
  • [https://bitbucket.org/seshadhri/escape](https://bitbucket.org/seshadhri/escape)

• [Danisch-Balalau-Sozio WWW18] Clique counting
  • [https://github.com/maxdan94/kClist](https://github.com/maxdan94/kClist)
Subgraph reconstruction

How to count (exactly) without enumerating
Wedge counting

- Non-induced count represented by (linear time computable) formula

\[ \sum_v \binom{d_v}{2} \]

Each pair of nbrs is wedge

• Non-induced count represented by (linear time computable) formula
Some simple (non-induced) formulas

\[ \sum_v \binom{d_v}{2} \]

\[ \sum_e \binom{t_e}{2} \]

\[ \sum_{t=(u,v,w)} (d_u + d_v + d_w - 6) \]

- We can count much faster than enumerating
  - Use local counts of “simpler” pattern
- Is there a general principle here?
Reconstructing H

\[
\sum_v \binom{d_v}{2} \quad \quad \sum_e \binom{t_e}{2} \quad \quad \sum_{t=(u,v,w)} (d_u + d_v + d_w - 6)
\]

- [Pinar-Seshadhri-Vishal WWW 17] Pattern cutting
Local Counting: Orbits

• The count depends on “role” in H
  • Formalized through automorphism group of H, and vertex orbits under automorphisms

• So we ask for count of specific orbit
  • Within subgraph, some orbits can be easier
The Hocevar-Demsar approach

- Local counting of orbits: for every $v$, for every $r$ in $V(H)$, count the number of $H$-subgraphs where $v$ “occurs as $r$”
The Hocevar-Demsar approach

- Delete vertex in H
- Enumerate resulting subgraph $H_s$
- Any neighbor of $H_s$ leads to potential copies of H (but also others)
- Build system of linear equations of $k$-subgraph orbits through counts of neighbors of $H'$-copies

- [Melckenbeeck-Audenaert-Michoel-Colle-Pickavet PLoS16] Automatic generation of these equations
Induced vs non-induced relationships

- [Ahmed et al ICDM15] Relations between induced and non-induced counts, by looking at wedges involving an edge
  - Focus on $k=4$ vertex patterns
  - Local information on 3-vertex patterns gives relationships among 4-vertex patterns
- [Rossi et al Arxiv19] Apply for counting attributed subgraphs

If dotted edge present: 4-cycle
Else: 3-path

Given count of all induced wedges on all edges, one gets total sum of 4-cycles and 3-paths
The pattern cutting approach

- [Pinar et al WWW17] By knowing the local counts of “fragments” of H, one can get the global count of H
  - Inclusion-exclusion to correct overcounting
  - Formal statement requires automorphism group info for right numbers
- In general, lemma is flexible. Allows for cut edge, or even cut path
  - But needs more complex local counts

\[(\text{#triangles at } v) \times (\text{# edges at } v) = \text{# tailed triangles involving } v \text{ as cut}\]

You have “correct” for edge and triangle intersecting

\[\sum_{t=(u,v,w)} (d_u + d_v + d_w - 6)\]
The pattern cutting approach

• Upshot: local counts for (k-1)-subgraphs give (through polynomial equations) k-subgraph counts

(#triangles at v) \times (# edges at v) = \# tailed triangles involving v as cut

You have “correct” for edge and triangle intersecting

\[ \sum_{t=(u,v,w)} (d_u + d_v + d_w - 6) \]
(Other) Past work

- [Wernicke-Rasche BioInf. 06] FANMOD
  - Also does sampling
  - http://theinf1.informatik.uni-jena.de/~wernicke/motifs/

- [Marcus-Shavitt ICDCS10] RAGE
  - Exact 4-vertex subgraph
  - http://www.eng.tau.ac.il/~shavitt/RAGE/Rage.htm

- [Ribiero-Silva DBSocial12] G-Tries
  - String-inspired data structure to store/query subgraphs
  - http://www.dcc.fc.up.pt/gtries/
Current SOA...?

- [Hocevar-Demsar 14] ORCA
  - http://www.biolab.si/supp/orca/

- [Ahmed-Neville-Rossi-Duffield15] PGD
  - http://nesreennahmed.com/graphlets/

- [Pinar-Seshadhri-Vishal 17] ESCAPE
  - https://bitbucket.org/seshadhri/escape
### About SOA

<table>
<thead>
<tr>
<th>Package</th>
<th>k (subgraph size)</th>
<th>What’s special?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orca [HD14]</td>
<td>( \leq 5 )</td>
<td>Local orbit counts</td>
</tr>
<tr>
<td>PGD [ANRD15]</td>
<td>( \leq 4 )</td>
<td>Support for threads (Open MP)</td>
</tr>
<tr>
<td>ESCAPE [PSV17]</td>
<td>( \leq 5 )</td>
<td>Fastest for exact counting</td>
</tr>
</tbody>
</table>

- Counting all 4-vertex subgraphs surprising efficient
  - For 117M edge social graph, all 4-vertex counts in 22m on laptop
- Counting all 5-vertex subgraphs feasible
  - For graphs with 10M edges, less than 30 minutes
  - For 117M edge social graph, takes 30 hours