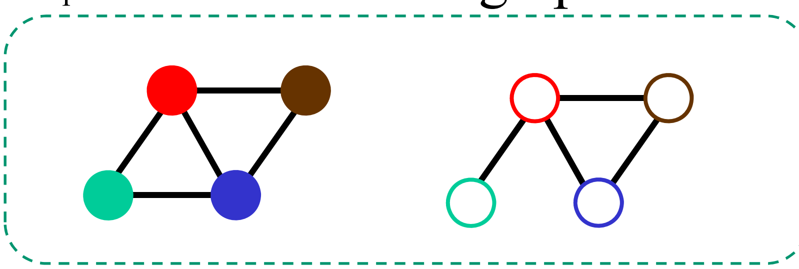
Our Graphs Become Larger

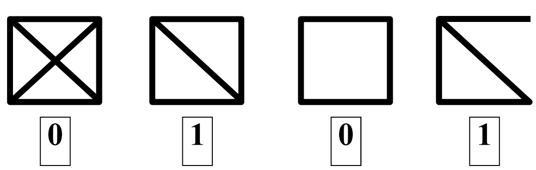
- Simple algorithms do not scale
  - $O(n^k)$ for size k graphlets
- Two approaches:
  - Find clever algorithms for counting small graphlets
  - Approximate count for larger graphlets

Induced Subgraphs

Graph

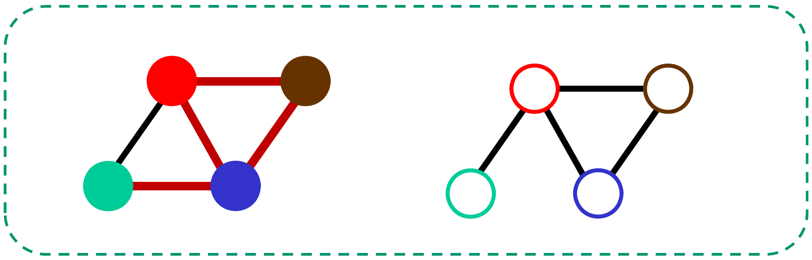


Potential Subgraphs

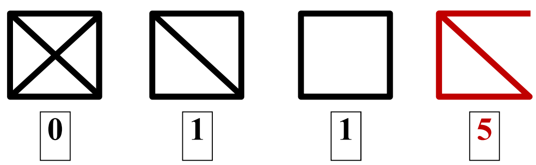


NON-Induced Subgraphs

Graph

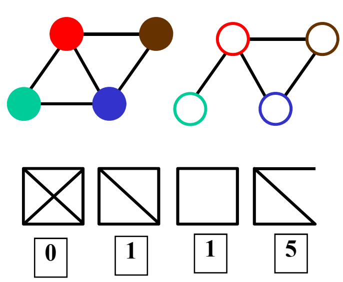


Potential Subgraphs

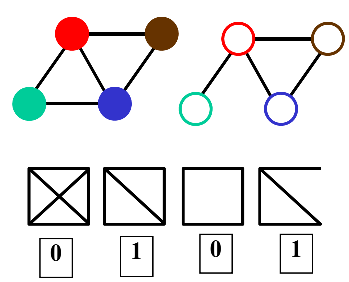


Subgraphs

Non-Induced



Induced



Efficient Counting of Graphlets

Key Observations:

- It is easier to count non-induced graphlets
- Counting all non-induced graphlets can give us all induced graphlets of some size.

Tree Decomposition

A tree decomposition of a graph $G = (V, E)$ is a pair (X, T), where $X = \{X_1, ..., X_n\}$ is a family of subsets of V, and T is a tree whose nodes are the subsets X_i, satisfying the following properties:

- Each graph vertex is associated with at least one tree node.
- For every edge (v, w) in the graph, there is a subset X_i that contains both v and w.
- If X_i and X_j both contain a vertex v, then all nodes X_k of the tree in the (unique) path between X_i and X_j contain v as well.

Treewidth

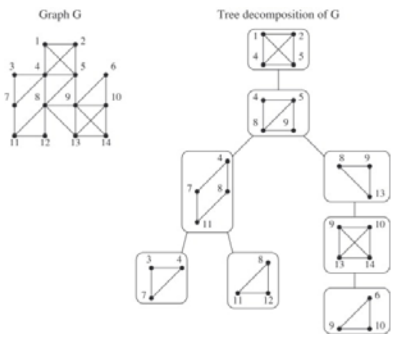
- The *width* of a tree decomposition is the size of its largest set X_i *minus one*.
- The **treewidth** tw(G) of a graph G is the minimum width among all possible tree decompositions of G.
  - *minus one* in order to make the treewidth of a tree equal to one.

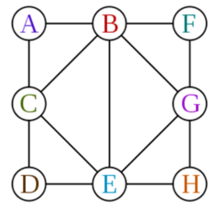
The Treewidth of Graphs

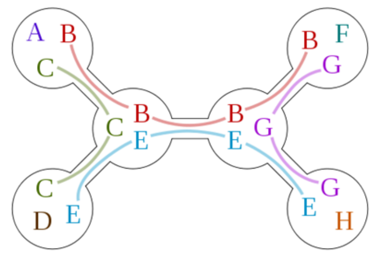
- Every complete graph K_n has treewidth $n - 1$.
- A connected graph with at least two vertices has treewidth 1 if and only if it is a tree.
- If a graph has a cycle, its treewidth is at least two.

- It is NP-complete to determine whether a given graph G has treewidth at most a given variable k.
  - For fixed k, we can check if a graph has a treewidth of k in O(n^k), and find the tree decomposition.

Examples







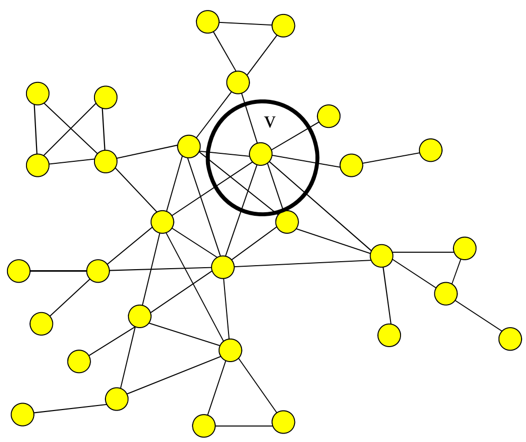
Why Treewidth?

Many algorithmic problems that are NP-complete for arbitrary graphs may be solved efficiently by dynamic programming for graphs of bounded treewidth, using the tree-decompositions of these graphs.

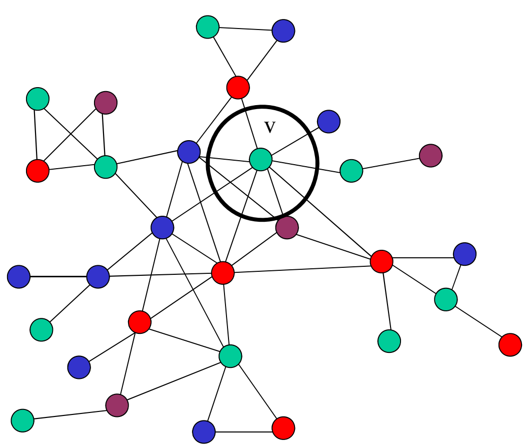
Example: *k*-coloring a graph

We will approximately count some graphlets based on color-coding alg. [*Alon, Yuster, and Zwick, JACM 1995*]

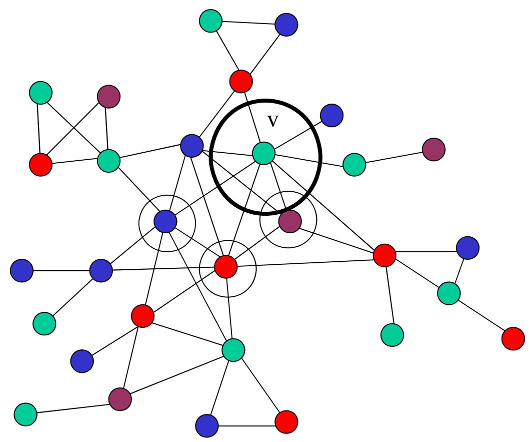
Counting Graphlets Using Color Coding



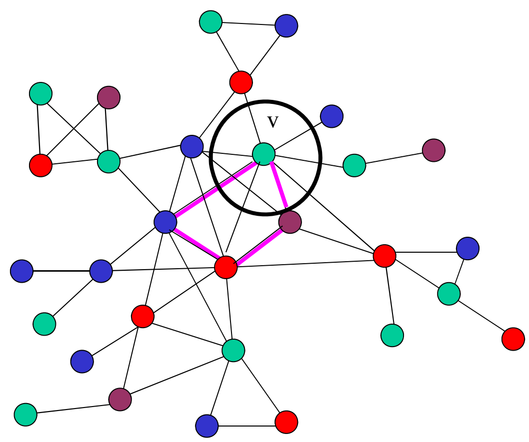
Counting Graphlets Using Color Coding



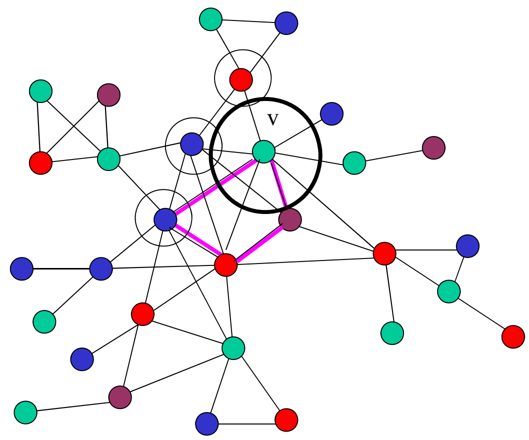
Counting Graphlets Using Color Coding



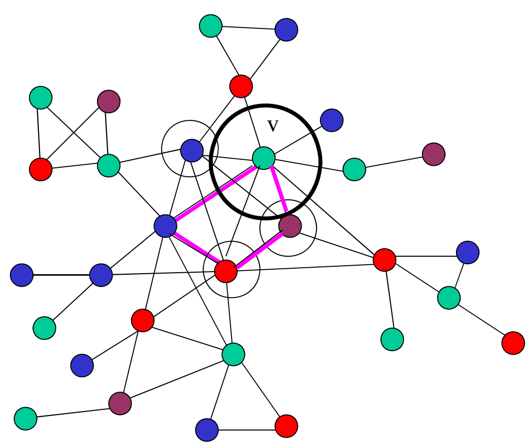
Counting Graphlets Using Color Coding



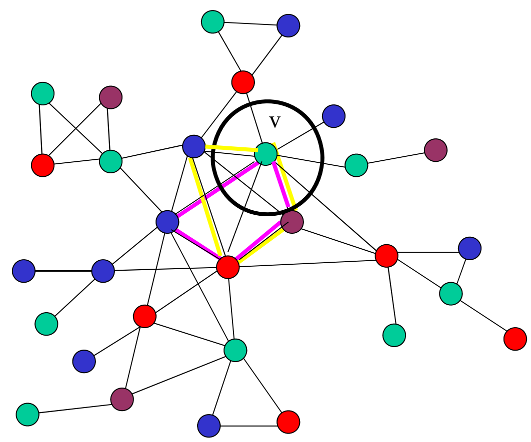
Counting Graphlets Using Color Coding



Counting Graphlets Using Color Coding



Counting Graphlets Using Color Coding



Counting Trees in a Graph

(also graphlets with bounded treewidth)

Given a network G with n vertices and a tree T with k vertices, we consider the problem of counting the number of non-induced subtrees of G that are isomorphic to T.

Assume $k \in O(\log n)$

[Alon, Yuster, and Zwick, JACM 1995]
[Alon *et al*., Bioinformatics 2008]
[Gonen & Shavitt, Internet Mathematics 2009]

The Counting Algorithm

1. **Color coding.** Color each vertex of input graph *G* independently and uniformly at random with one of the *k* colors.
2. **Counting.** Apply a dynamic programming routine (explained later) to count the number of non-induced occurrences of *T* in which each vertex has a unique color.
3. Repeat the above two steps $O(e^k)$ times and add up the number of occurrences of *T* to get an estimate on the number of its occurrences in *G*.

Color Coding

r - the total number of copies of *T* in *G*.

We assign a color (ind. and uar) to each vertex of *G* from the color set $[k]=\{1,\ldots,k\}$.

For some occurrence of T, the probability that all T's vertices are assigned unique colors is $p=k!/k^k$, thus the expected number of colorful copies in *G* is *rp*.

Let $\mathcal{F}$ denote the family of all copies of T in G.

For each such copy $F \in \mathcal{F}$, let x_F denote the indicator random variable whose value is 1 if and only if the copy is colorful in our random k-coloring of $V(G)$, the vertices of G.

Let $X=\sum_{F\in\mathcal{F}} x_F$ be the random variable counting the total number of colorful copies of T.

By linearity of expectation, the expected value of X is $E(X)=rp$.

Estimating X Variance

For every two distinct copies $F,F'\in\mathcal{F}$, the probability that both F and F' are colorful is at most p.

- in fact strictly smaller unless both copies have exactly the same set of vertices.

$\Rightarrow$ the covariance $\mathrm{Cov}(x_F,x_{F'})$ satisfies:

$$\mathrm{Cov}(x_F,x_{F'}) = E(x_F x_{F'}) - E(x_F)E(x_{F'}) \le p.$$

Estimating X Variance (cont.)

The variance of X:

$$\mathrm{Var}(X) = \sum_{F \in \mathcal{F}} \mathrm{Var}(x_F) + \sum_{F \neq F' \in \mathcal{F}} \mathrm{Cov}(x_F, x_{F'})$$
$$\leq rp + r(r-1)p = r^2 p.$$

Y = the average of s independent copies of X (obtained by s independent random colorings)

$$E(Y)=E(X)=rp$$

$$\mathrm{Var}(Y) = \mathrm{Var}(X)/s \leq r^2 p/s.$$

Estimating X Variance (cont.)

Applying Chebyshev's inequality

Let X be a random variable with finite expected value μ and finite non-zero variance σ^2. Then for any real number $k > 0$,

$$\Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}.$$

the probability that Y is smaller than (or bigger than) its expectation by at least εrp is at most

$$\frac{r^2 p}{\epsilon^2 r^2 p^2 s} = \frac{1}{\epsilon^2 ps}.$$

if $s=4/\varepsilon^2 p$ this probability is at most ¼.

Decreasing the Error in *Y*

Compute *Y* *t* times independently.

Let *Z* be the median.

The probability that the median is less than $(1-\varepsilon)rp$ is the probability that at least half of the copies of *Y* computed will be less than this quantity, which is at most $\binom{t}{t/2}4^{-t} \leq 2^{-t}$.

A similar estimate holds for the probability that *Z* is bigger than $(1+\varepsilon)rp$.

Decreasing the Error in *Y* (cont.)

If $t=\log(1/\delta)$ then with probability $1-2\delta$ the value of *Z* will lie in $[(1-\varepsilon)rp,(1+\varepsilon)rp]$.

Note that the total number of colorings in the process is *ts*: $O\left(\frac{\log(1/\delta)}{\epsilon^2 p}\right) = O\left(\frac{e^k \log(1/\delta)}{\epsilon^2}\right)$.

Our estimate for *r* is $Z/p = Zk^k/k!$

Counting Paths of length k

Let $C(v,S)$ be the number of colorful paths for which one of the endpoints is v.

S is a subset of the color set $\{1,\ldots,k\}$,

col(v) is the color of vertex v.

Given a color ℓ, for all $v \in V(G)$:

$$C(v,\{\ell\}) = \begin{cases} 1 & \text{if } \mathrm{col}(v) = \ell \\ 0 & \text{otherwise.} \end{cases}$$

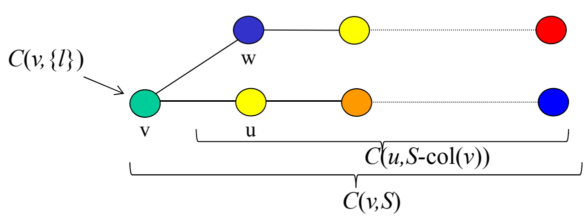
Counting Paths of length k (cont.)

For each vertex v and color set S where $|S|>1$:

$$C(v,S) = \sum_{u;(u,v)\in E(G)} C\big(u, S - col(v)\big).$$

The number of single colorful paths of length k is

$$\frac{1}{2}\sum_{v} C\big(v,\{1,\ldots,k\}\big).$$



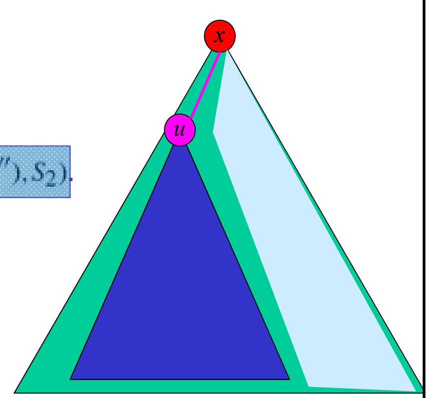
Counting Trees

- ρ – an arbitrary vertex of the tree selected to be the root.
- τ(ρ) – the rooted tree
- For each vertex v of the graph G, we compute $c(v,\tau(\rho),[k])$, the number of $[k]$-colorful rooted subtrees with root v, which are isomorphic to τ(ρ).
- The actual number of $[k]$-colorful occurrences of T in G is $\frac{1}{q}\sum_{v} c(v,\tau(\rho),[k])$

q is equal to the number of vertices u in T, for which the rooted tree τ(u) is isomorphic to τ(ρ).

The Dynamic Program

$$c(x,\tau'(\rho'),S) = \frac{1}{d}\sum_{\forall S_1,S_2:|S_1\cap S_2|=\emptyset} c(x,\tau_1(\rho'),S_1)\cdot c(u,\tau_2(\rho''),S_2).$$



Complexity of the dynamic program: $O(2^k|E|)$
Overall complexity:

$$O\left(|E|\cdot 2^k\cdot e^k \log(1/\delta)\cdot\frac{1}{\epsilon^2}\right).$$

$O(2^k|V||E|)$

Remarks

- The algorithm can be extended for graphlets with bounded treewidth.
- If the graphlet sized in O(log n) the $(2e)^k$ term in the complexity is O(n)
- We can use this algorithms to count graphlets in a graph, graphlets attached to a node, and orbits (at least for trees).

Counting Cycles

Algorithm 1. (An (ϵ, δ)-approximation algorithm for counting simple cycles of length k adjacent to a vertex v)

1. For $j = 1$ to t
   (a) For $i = 1$ to s
      i. Color each vertex of G independently and uniformly at random with one of the k colors.
      ii. For all $x \in V$, $C_i(v, x, [k]) = \text{count-path}(v, x, k)$. [See Algorithm 2.]
      iii. Let $CY_i(v, [k]) = \frac{1}{2} \sum_{u \in N(v)} C_i(v, u, [k])$.
      iv. Let $X_i^v = CY_i(v, [k])$.
   (b) Let $Y_j^v = \frac{\sum_{i=1}^{s} X_i^v}{s}$.
2. Let Z^v be the median of $Y_1^v, \ldots, Y_t^v$.
3. Return $Z^v \cdot k^k / k!$.

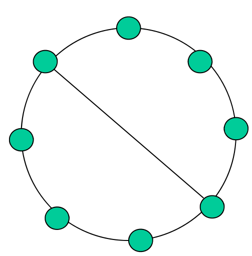
Time Complexity

As before

$O(2^k\ n\ |E|)$ – for calculating the paths for all v

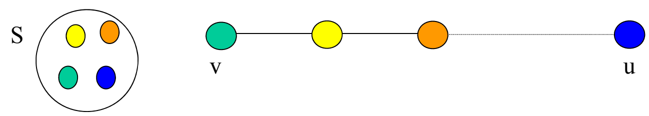
Overall compelxity: $O((2e)^k \cdot |E| \cdot |V| \cdot \log(1/\delta)/\varepsilon^2)$

Cycle with a Chord

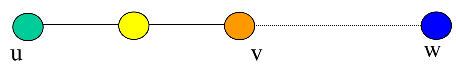


Mathematical Notations

- **C(v,u,S)** = the number of colorful paths from v to u in a specific coloring, using the colors in S.



- **P(v,u,w,S)** = the number of colorful paths from u to w that are adjacent to v in a specific coloring, using the colors in S.



The Algorithm

Algorithm's Input:

A graph G(V,E), a vertex v, fault-tolerance ε, error probability δ

Notation: let $A_{V',z,b}(S)$ be the set of all pairs (S_1,S_2) such that the following hold:

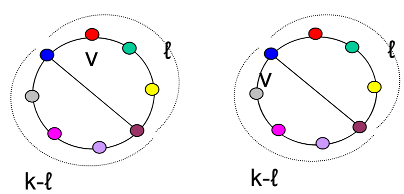
- $|S_1| = z+1$,
- $|S_2|=b-z+1$,
- $S_1 \cup S_2=S$,
- $S_1\backslash\{col(u)|u\in V'\}\cap S_2\backslash\{col(u)|u\in V'\}= \phi$

Repeat $t=\log(1/\delta)$ times

The Algorithm

1. Color each vertex of G independently and uniformly at random with one of the k colors.
2. Compute the number of cycles with a chord in the coloring; there are two cases:

Case 1 Case 2

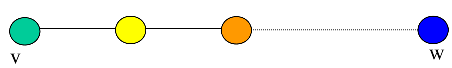


2.1 Compute $X_{1,v}$ = the number of k-length cycles with a chord in case 1.
2.2 Compute $X_{2,v}$ = the number of k-length cycles with a chord in case 2.

3. Let Y_v = the average of all the *s* $X_{1,v}+X_{2,v}$.
4. Return the median of all the t Y_v multiplied by $k^k/k!$.

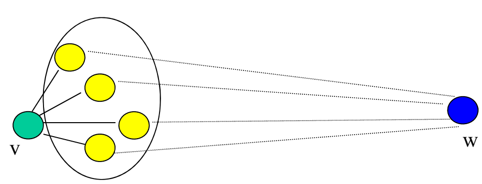
Repeat $s=4k^k/\varepsilon^2 k!$ times

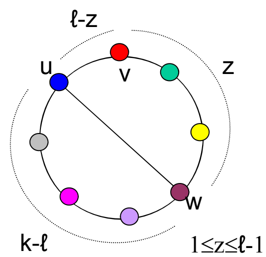
Computing the number of paths between v,w, for every color-set S



- For all $S \subseteq [k]$ s.t $S=\{\ell\}$ $C(v,w,S)=1$ if $col(v)=col(w)=\ell$, and 0 otherwise.
- For $q=2$ to k, for all $S \subseteq [k]$ s.t $|S|=q$

$$\mathbf{C(v,w,S) = \sum_{u \in N(v)} C(u,w,S\setminus\{col(v)\})}$$





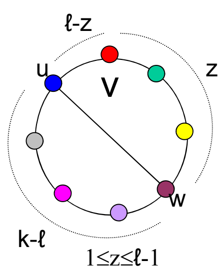
Case 1

- $P(v,u,w,S) = \sum_{1\le z\le \ell-1}\sum C(v,w,S_1)\cdot C(v,u,S_2)$

\# z-length colorful paths between u and w using colors in S_1

\# ℓ-z-length colorful paths between v and u using colors in S_2

The sum is over all (S_1,S_2) in $A_{v,z,\ell}(S)$



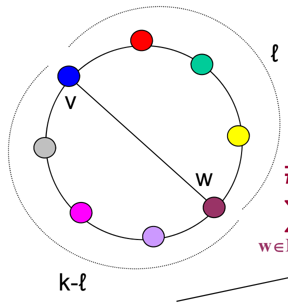
Case 1

- \# of cycles with a chord in case 1 = $\sum_{(u,w)\in E}\sum_{\ell}\sum P(v,u,w,S_3)\cdot C(u,w,S_4)$

\# ℓ-length colorful paths between u and w that are adjacent to v using colors in S_3

\# k-ℓ-length colorful paths between u and w using colors in S_4

The sum is over all (S_3,S_4) in $A_{\{u,w\}\ell,k}([k])$ and over all (S_3,S_4) in $A_{\{u,w\},k-\ell,k}([k])$



Case 2

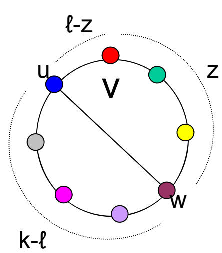
# of cycles with a chord in case 2 =

$$\sum_{w\in N(v)}\sum_{\ell}\sum C(v,w,S_1)\cdot C(v,w,S_2)$$

ℓ-length colorful paths between v and w using colors in S_1

k-ℓ-length colorful paths between v and w using colors in S_2

The sum is over all (S_1,S_2) in $A_{\{v,w\},\ell,k}([k])$



Time Complexity

Case 1

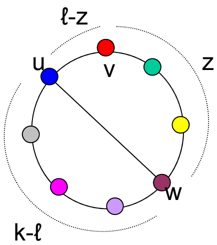
The time complexity of computing C(v,w,S) for every color-set S and every pair of vertices v,w in a given coloring is $O(k\cdot 2^k\cdot|E|\cdot|V|)$.

The time complexity of computing P(v,u,w,S) for every color-set S and fixed v,u,w (assuming C(v,w,S) is known) is

$$O\left(\sum_{\ell=1}^{k}\sum_{z=1}^{\ell-1}\binom{k}{\ell}\binom{\ell}{z}\right)=O\left(\sum_{\ell=1}^{k}\binom{k}{\ell}\cdot 2^{\ell}\right)=O(3^k)$$

Choosing the colors of the path between u and w that is going through v

Choosing the colors of the path between v and w

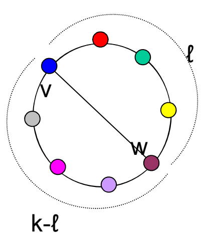


Time Complexity

Case 1

→The time complexity of the first case, for every edge (u,w), every vertex v, and every color-set S, in a given coloring is

$$O\left(\sum_{v\in V}\sum_{(u,w)\in E} 3^k\right) = O(3^k|E||V|)$$



Time Complexity

Case 2

The time complexity of the second case, for every vertex v, every neighbor of v, and every color-set S, in a given coloring is

$$O\left(\sum_{v\in V}\sum_{w\in N(v)}\sum_{\ell=1}^{k}\binom{k}{\ell}\right) = O(2^k \cdot |E|)$$

Choosing the colors of the path between v and w

Time Complexity

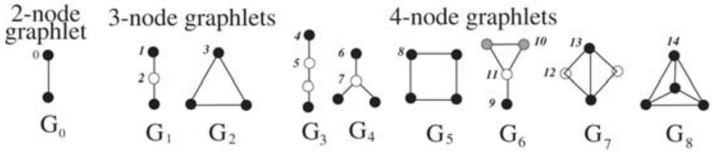
→The total time complexity, for every vertex v and every color-set S, in a given coloring, is

$O(3^k \cdot |E| \cdot |V|)$.

Time complexity =
$O((3e)^k \cdot |E| \cdot |V| \cdot \log(1/\delta)/\varepsilon^2)$

Small Graphlets

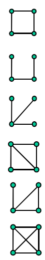
- In many practical cases we are only interested in the graphlets of small size, ≤5
- We have already seen how to count triangles (and 3 node paths)
- How efficient can we find 4-node graphlets?



[Marcus & Shavitt, Computer Networks 2012]

Maybe we can use our approx.?

Motif **Time Complexity**



O(|E|·|V|·log(1/δ)/ε²)

O(|E|·log(1/δ)/ε²)

O(|E|·log(1/δ)/ε²)

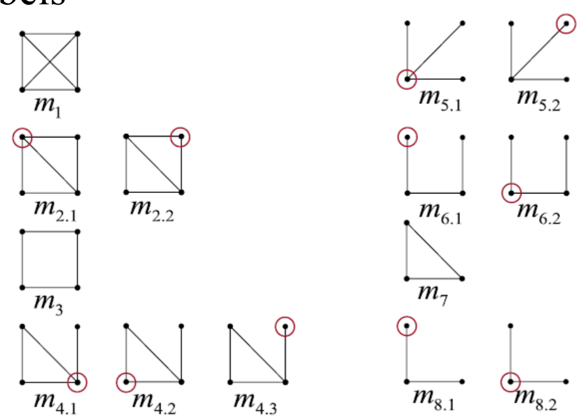
O(|E|·|V|·log(1/δ)/ε²)

<u>(ε,δ) – Approximation:</u>

$\Pr[(1-\varepsilon)\cdot\#f \le y \le (1+\varepsilon)\cdot\#f] \ge 1-2\delta$

Optimal Algorithms

- Counting all orbits
- Assume nodes labels are {1,2,…,n}



Triangle Counting Per Node

All algs use Merge

In Merge:

NodeArray is not useful for counting Δs, will be needed by other algorithms.

$O(d|E|)$

Algorithm 1. Counting triangles

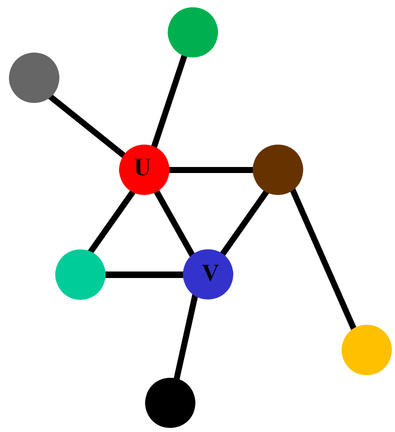
```
procedure TriangleCount(G)
  for all v ∈ V(G) do
    for all u ∈ N(v), v < u do
      Merged ← Merge(G, v, u, V_arr)
      m_7[v] ← m_7[v] + |Merged|
      m_7[u] ← m_7[u] + |Merged|
  for all v ∈ V(G) do
    m_7[v] ← m_7[v]/2
procedure Merge(G, v, u, NodeArray)
  for all w ∈ N(v) do
    NodeArray[w] ← 0
  for all w ∈ N(u) s.t. w ≠ v do
    NodeArray[w] ← 1
  for all w ∈ N(v), w ≠ u do
    if NodeArray[w] = 1 then
      NodeArray[w] ← 3
      list ← AppendToList list, w
    else
      NodeArray[w] ← 2
  return list
```

Counting Cycles With Chord

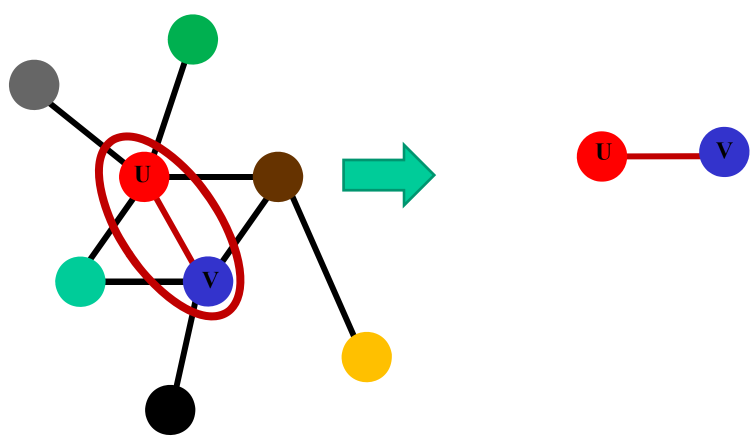
Algorithm 2. Counting non-induced 4-cycles with a chord

```
procedure ChordCycleCount(G)
  for all v ∈ V(G) do
    for all u ∈ N(v), v < u do
      Merged ← MergeG, v, u, V_arr
      m_2.1[v] ← m_2.1[v]+
                |Merged| · (|Merged| − 1)/2
      m_2.1[u] ← m_2.1[u]+
                |Merged| · (|Merged| − 1)/2
      for all w ∈ Merged do
        m_2.2[w] ← m_2.2[w] + (|Merged| − 1)
```

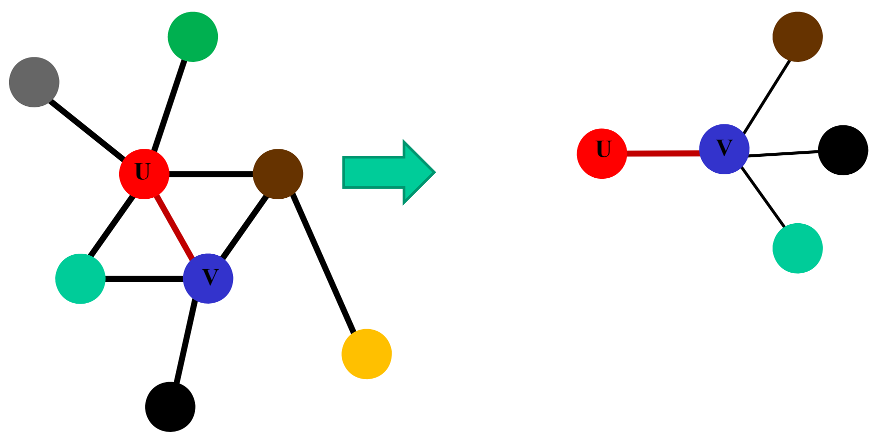
Counting Cycles With Chord



Counting Cycles With Chord

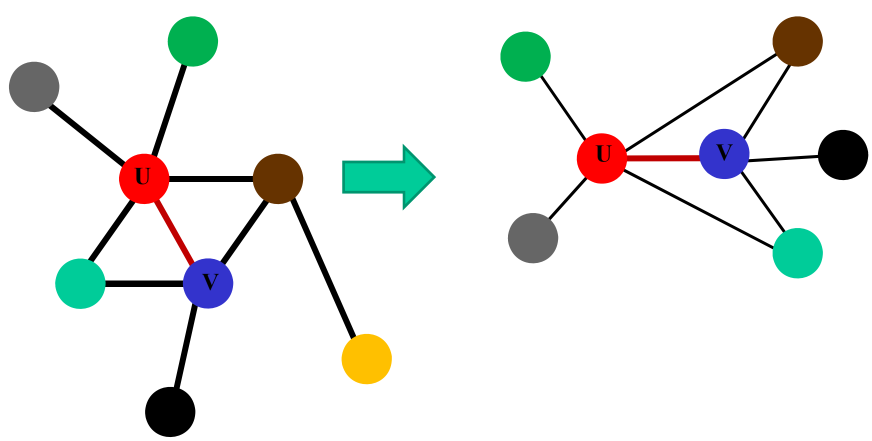


Counting Cycles With Chord

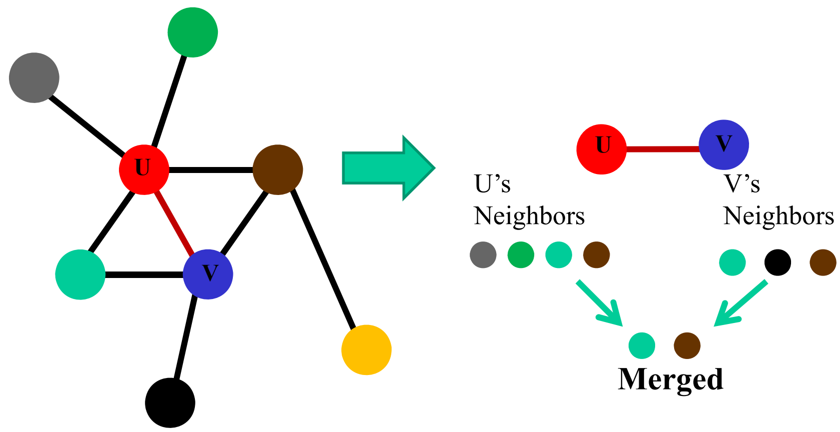


Counting Cycles With Chord

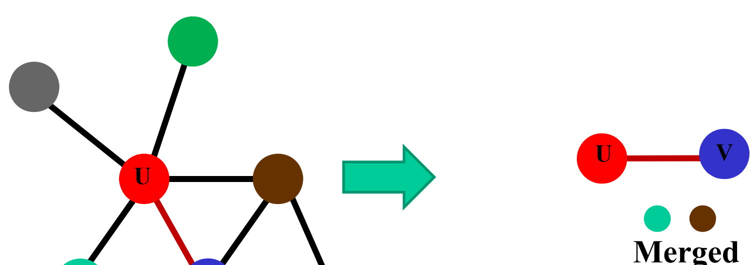
Example



Counting Cycles With Chord



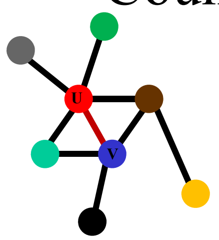
Counting Cycles With Chord



$$\binom{|Merged|}{2}$$

Counting Cycles With Chord

Runtime Analysis



$$O(|E|*(2*d\log d+d+d))=O(|E|*d\log d)$$

Iterate over all Edges

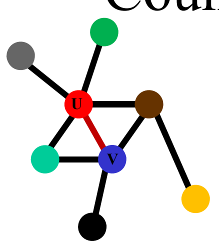
Sort both Neighbor lists

Merge Neighbor lists

Update count for relevant nodes

Counting Cycles With Chord

Runtime Analysis



$$O(|E|*(2*d\log d+d+d))=O(|E|*d\log d)$$

*we can get time complexity of $\mathbf{O(|E|*}\boldsymbol{d}\mathbf{)}$

Assuming graph veracities are labeled by the integers {1…n}

Counting Tail Triangles

Algorithm 3. Counting non-induced tailed triangles

procedure TAILTRIANGLECOUNT(G)
  TriangleCount(G)
  for all $v \in V(G)$ **do**
    for all $u \in N(v)$, $v < u$ **do**
      $Merged \leftarrow$ Merge(G, v, u, V_{arr})
      $tails_v \leftarrow \max\{0,(|N(v)| - 2)\}$
      $tails_u \leftarrow \max\{0, (|N(u)| - 2)\}$
      for all $w \in Merged$ **do**
        $m_{4.2}[w] \leftarrow m_{4.2}[w] + tails_v + tails_u$
  for all $v \in V(G)$ **do**
    $m_{4.1}[v] \leftarrow \max\{0, (m_7[v] \cdot (|N(v)| - 2))\}$
    for all $u \in N(v)$ **do**
      $m_{4.3}[v] \leftarrow m_{4.3}[v] + m_7[u]$
    $m_{4.3}[v] \leftarrow m_{4.3}[v] - 2 \cdot m_7[v]$

O(d|E|)

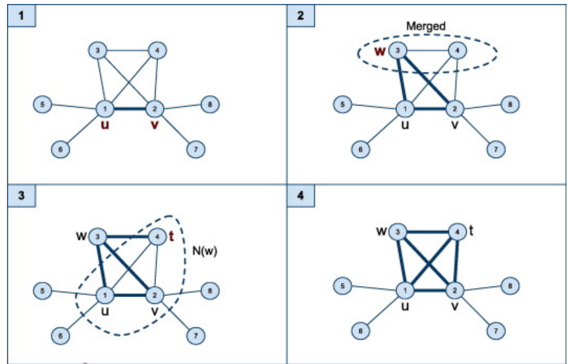
Counting 4 Cliques

Algorithm 4. Counting 4-cliques

1: **procedure** CLIQUECOUNT(G)
2:   **for all** $v \in V(G)$ **do**
3:     $V_{arr}[v] \leftarrow 0$
4:   **for all** $v \in V(G)$ **do**
5:     **for all** $u \in N(v)$, $v < u$ **do**
6:       $Merged \leftarrow$ Merge$G$, v, u, V_{arr}
7:       **for all** $w \in Merged$ **do**
8:         **for all** $t \in N(w)$, $w < t$ **do**
9:           **if** $V_{arr}[t] = 3$ **then**//(∗)
10:             $m_1[v] \leftarrow m_1[v] + 1$
11:             $m_1[u] \leftarrow m_1[u] + 1$
12:       **for all** $w \in N(v) \cup N(u)$ **do**
13:         $V_{arr}[w] \leftarrow 0$
14:   **for all** $v \in V(G)$ **do**
15:     $m_1[v] \leftarrow m_1[v]/3$

(∗) V_{arr} is set in the MERGE call (line 6). $V_{arr}[t] = 3$ iff t is a neighbor of both v and u

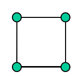
Complexity



$$O\left(\sum_{v\in V}\sum_{u\in N(v)}\left(d+\sum_{w\in(N(v)\setminus N(u))}|N(w)|\right)\right)$$
$$=O\left(\sum_{v\in V}\sum_{u\in N(v)}(d)+\sum_{v\in V}\sum_{u\in N(v)}\sum_{w\in(N(v)\setminus N(u))}|N(w)|\right)$$
$$=O(d|E|)+O(|E|^2). \quad \square$$

In bounded degree graphs $\sum_w N(w) = d^2$ (instead of |E|) so the complexity is O(d^2|E|)

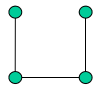
Counting 4 Cycles



Algorithm 5. Counting 4-cycles

```
procedure CycleCount(G)
  for all v ∈ V(G) do
    V_arr[v] ← 0
  for all v ∈ V(G) do
    for all u ∈ N(v) s.t. v < u do
      Merge G, v, u, V_arr
      for all w ∈ N(v)\{u} do
        for all t ∈ N(w) do
          if V_arr[t] = 1 or V_arr[t] = 3 then
            m_3[v] ← m_3[v] + 1
            m_3[u] ← m_3[u] + 1
      for all w ∈ N(v) ∪ N(u) do
        V_arr[w] ← 0
  for all v ∈ V(G) do
    m_3[v] ← m_3[v]/2
```

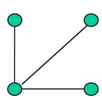
$$\left(\sum_{v\in V}\sum_{u\in N(v)}\left(d+\sum_{w\in N(v)}|N(w)|\right)\right)=O(d|E|)+O(|E|^2)$$



O(d|E|)

Algorithm 6. Counting 4-node paths

```
procedure PathCount(G)
  for all v ∈ V(G) do
    V_arr[v] ← 0
  for all v ∈ V(G) do
    for all u ∈ N(v) s.t. v < u do
      Merged ← MergeG, v, u, V_arr
      m_6.2[v]
←m_6.2[v] + (|N(v)\{u}| · |N(u)\{v}|) − |Merged|
      m_6.2[u]
←m_6.2[u] + (|N(v)\{u}| · |N(u)\{v}|) − |Merged|
      for all w ∈ N(v)\{u} do
        if V_arr[w] = 3 then
          m_6.1[w] ← m_6.1[w] + |N(u)\{v}| − 1
        else
          m_6.1[w] ← m_6.1[w] + |N(u)\{v}|
      for all w ∈ N(u)\{v} do
        if V_arr[w] = 3 then
          m_6.1[w] ← m_6.1[w] + |N(v)\{u}| − 1
        else
          m_6.1[w] ← m_6.1[w] + |N(v)\{u}|
      for all w ∈ N(v) ∪ N(u) do
        V_arr[w] ← 0
```



Counting Claws

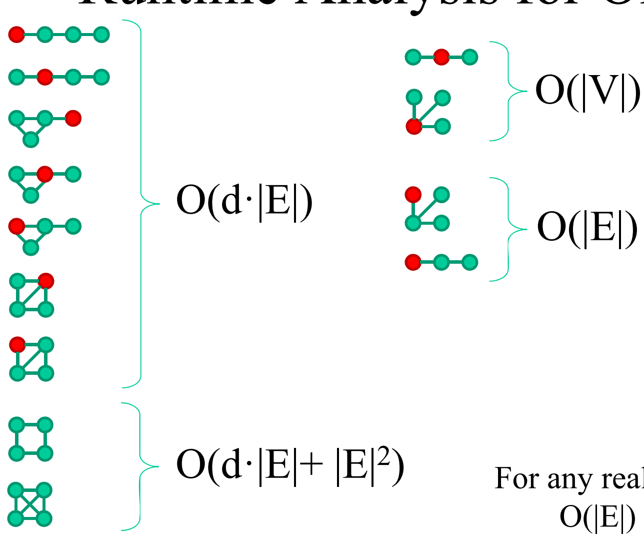
$$m_{5.1}[v] = \binom{|N(v)|}{3},$$

$$m_{5.2}[v] = \sum_{u \in N(v)} \binom{|N(u)| - 1}{2}$$

Summary for Graphlets

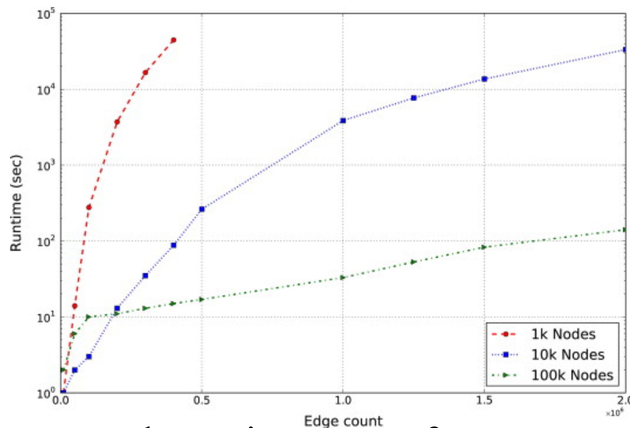
Motif	Time Complexity
	**O(|E|^2)**
	**O(d·|E|)**
	**O(|V|)**
	**O(d·|E|)**
	**O(d·|E|)**
	**O(|E|^2) or O(|E|d^2)**

Runtime Analysis for Orbits



For any real-world graph
O(|E|) = O(|V|)

Performance



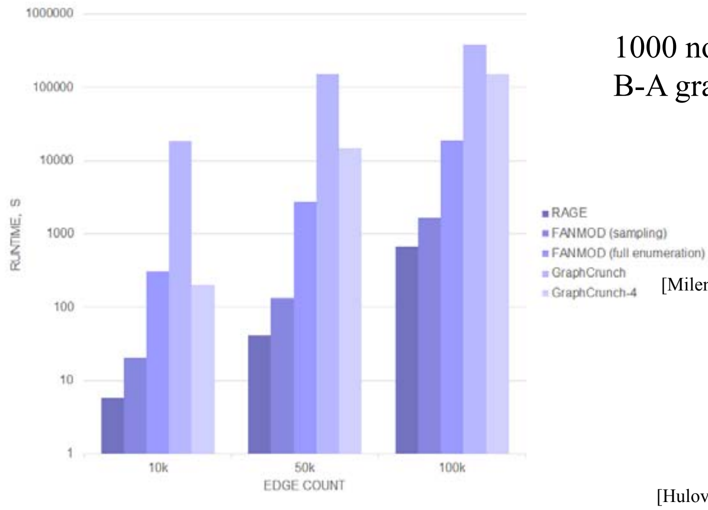
- Density seems to be an important factor

Comparison

	8k edges	20k edges	52k edges	92.6k edges
Method	5k nodes	10k nodes	20k nodes	AS-Graph (26k nodes)
RAGE	11	64	720	2400
FANMOD (sampling)	57	210	1020	7200
FANMOD (full-enum)	420	2040	25200	172800
IML	140	1203	9850	25000

Rage is faster even than FANMOD sampling

Comparison

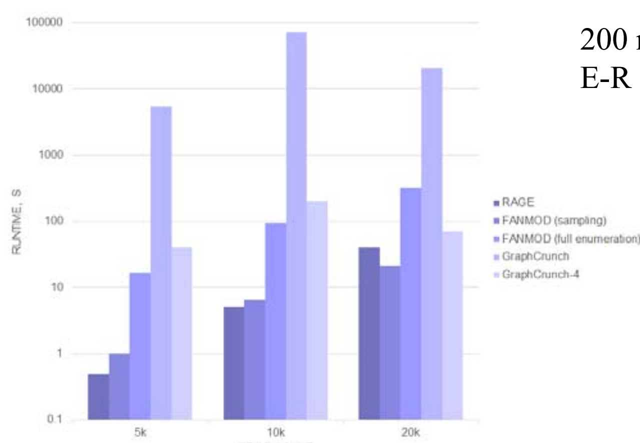


1000 nodes
B-A graphs

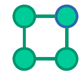
[Milenkovic, Lai, and Przulj, 2008]

[Hulovatyy and Solava, Notre Dame]

Comparison

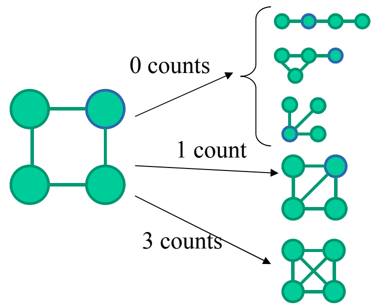


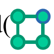
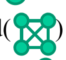
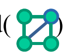
200 nodes
E-R graphs

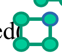
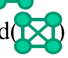
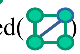
Inferring Induced Count 

4 Node Cycles

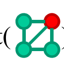
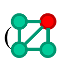
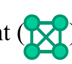
- Fix the over-count occurred due to subgraphs with access edges

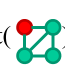
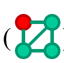
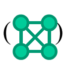


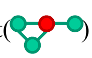
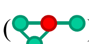
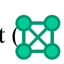
Induced Count = **Non**Induced() – 3*Induced() – 1*Induced()

= **Non**Induced() + 3***Non**Induced() – 1***Non**Induced()

Inferring Induced Counts of Orbits

Induced Count() = Non Induced Count () – 3*Non-Induced Count ()

Induced Count() = Non Induced Count () – 3*Non-Induced Count ()

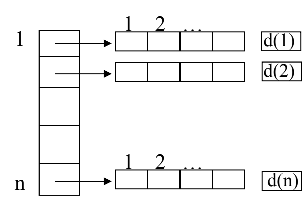
Induced Count() = Non Induced Count ()
– 2*Induced Count ()
– 3*Induced Count ()

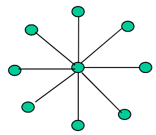
Sublinear Algorithm

- Assume a very large graph
- We can query the graph
  - Minimize the query number
- Sublinear approximation

Counting Stars: Model

- We assume graphs are represented by the incidence lists of the vertices, where each list is accompanied by its length.
- Allowed queries:
  - what is the degree, d(v), of any vertex v?
  - who is the i'th neighbor of v, for any vertex v and index 1≤ i ≤ d(v)?





Approximating Stars

Upper Bound:

- Given an approximation parameter $0 < \varepsilon < 1$ and query access to a graph G, our algorithm outputs an estimate ν'_s such that, with high constant probability,

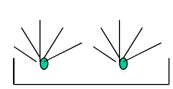
$$(1-\varepsilon)\nu_s(G) \leq \nu'_s \leq (1+\varepsilon)\,\nu_s(G),$$

  where $\nu_s(G)$ denotes the number of stars of size $s+1$ in the graph.
- The expected query complexity and running time of the algorithm are

$$O\left(\frac{n}{\nu_s(G)^{\frac{1}{s+1}}} + \min\left\{n^{1-\frac{1}{s}}, \frac{n^{s-\frac{1}{s}}}{\nu_s(G)^{1-\frac{1}{s}}}\right\} \cdot poly(\log n, 1/\varepsilon)\right)$$

Upper Bound-Main Idea

- Consider a partition of the graph vertices into $O(\log n/\varepsilon)$ buckets where in each bucket all vertices have the same degree (with respect to the entire graph) up to a multiplicative factor of $(1\pm O(\varepsilon))$. The degree in bucket B_i is $\sim(1+\beta)^i$, $\beta=O(\varepsilon)$.
- If we could get a good estimate of the size of each bucket by sampling, then we would have a good estimate of the number of s-stars (since the vertices in each bucket are the centers of approximately the same number of stars).



- The difficulty is that some buckets may be very small and we might not even hit them when sampling vertices. However, these buckets can significantly contribute to the number of stars in the graph.