Motif Counting in Preferential Attachment Graphs

Jan Dreier, Peter Rossmanith

December 11, 2019

Theoretical Computer Science
RWTH Aachen University, Germany

FSTTCS 2019

Subgraph Counting

How often does H with k edges occur in G of size n?

Subgraph Counting

How often does H with k edges occur in G of size n?

- important in network analysis and biology

Subgraph Counting

How often does H with k edges occur in G of size n?

- important in network analysis and biology
- many tools that work on real data for small k

Subgraph Counting

How often does H with k edges occur in G of size n?

- important in network analysis and biology
- many tools that work on real data for small k
- cannot be solved in time $f(k)n^{o(k)}$ under #ETH

Subgraph Counting

How often does H with k edges occur in G of size n?

- important in network analysis and biology
- many tools that work on real data for small k
- cannot be solved in time $f(k)n^{o(k)}$ under #ETH
- *Curticapean, Dell, Marx:* in time $k^{O(k)}n^{0.174k+o(k)}$

Average Case

Analyze runtime on *typical* instances.

Average Case

Analyze runtime on *typical* instances.

theoretical computer science:

efficient algorithms

network science:

models for the real world

Average Case

Analyze runtime on *typical* instances.

theoretical computer science:

efficient algorithms

network science:

models for the real world

$\Downarrow$

efficient algorithms for the real world?

Real-World Complex Networks

Some properties of complex networks:

- **Degree distribution**
  Most people have little but a few have a lot.
- **Small world**
  Everyone is close to everyone.
- **Clustering**
  If we have a common friend we are likely friends as well.

Real-World Complex Networks

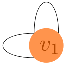
Some properties of complex networks:

- **Degree distribution** ✓
  Most people have little but a few have a lot.
- **Small world** ✓
  Everyone is close to everyone.
- **Clustering** ✗
  If we have a common friend we are likely friends as well.

Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

- G_2^1:



Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

- G_2^1:

Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

- G_2^1:

Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

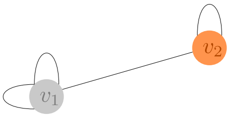
- G_2^1:

Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex
- G_2^1:

Preferential Attachment

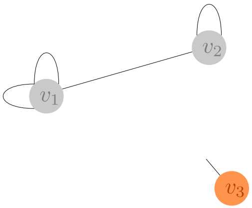
- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex
- G_2^2:



Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

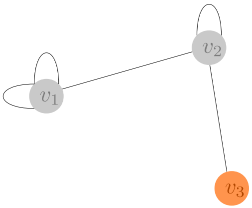
- G_2^2:



Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

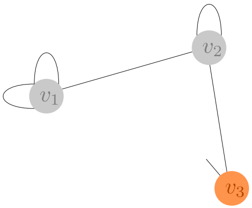
- G_2^2:



Preferential Attachment

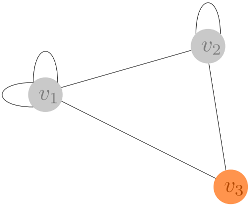
- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

- G_2^2:



Preferential Attachment

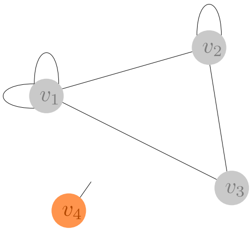
- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex
- G_2^3:



Preferential Attachment

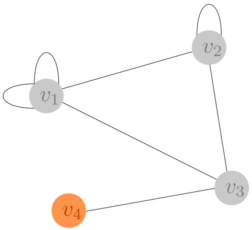
- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

- G_2^3:



Preferential Attachment

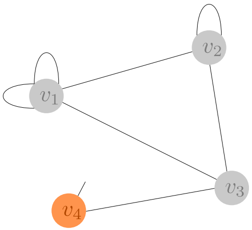
- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex
- G_2^3:



Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

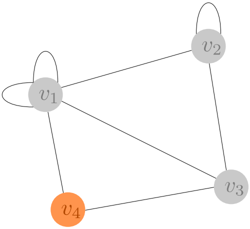
- G_2^3:



Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

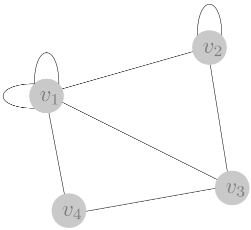
- G_2^4:



Preferential Attachment

- G_m^n graph with vertices $v_1, \ldots, v_n$ and m edges per vertex

- G_2^4:



Our Result

Theorem

One can compute how often H occurs in G_m^n in expected FPT time $f(|H|)m^{O(|H|^6)}\log(n)^{O(|H|^{12})}n$.

In particular, for fixed $m \in \mathbf{N}$ and H the run time is $\log(n)^{O(1)}n$.

Outline

Simple Algorithm

$\downarrow$

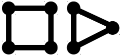
Bound on number of certain subgraphs

$\downarrow$

Bound on degrees of individual vertices

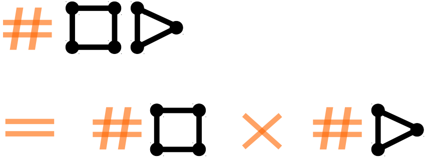
Counting Disconnected Subgraphs

It is sufficient to count connected subgraphs.

\# 

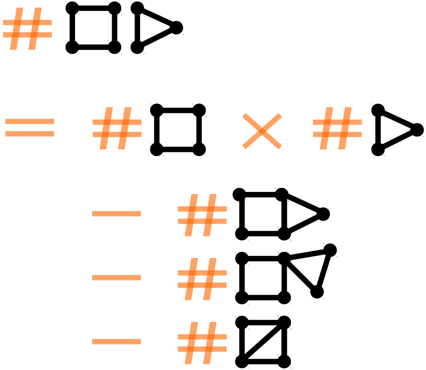
Counting Disconnected Subgraphs

It is sufficient to count connected subgraphs.

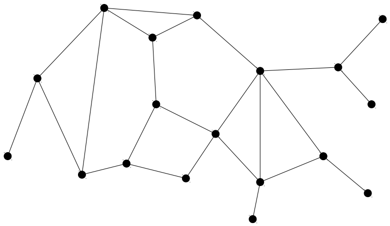


Counting Disconnected Subgraphs

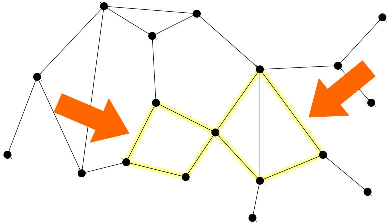
It is sufficient to count connected subgraphs.



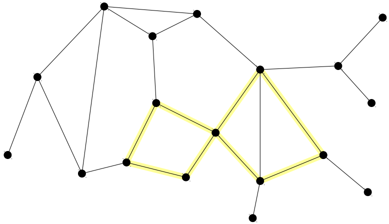
Example: Counting the number of squares



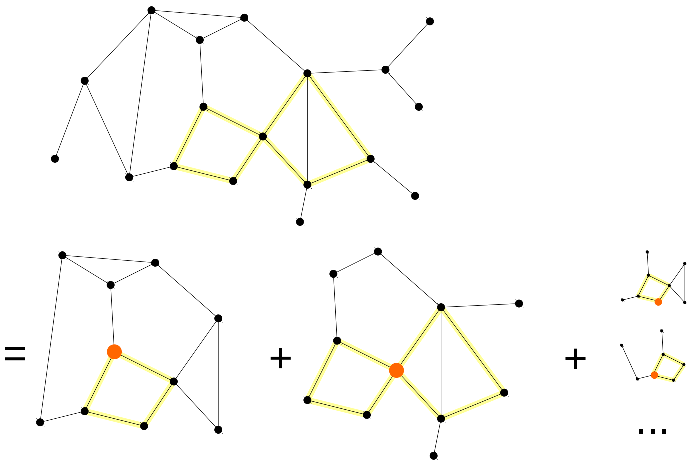
Example: Counting the number of squares



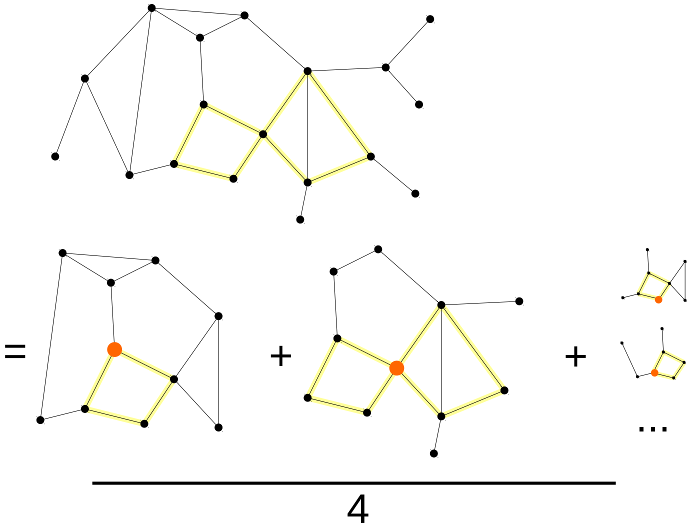
Example: Counting the number of squares



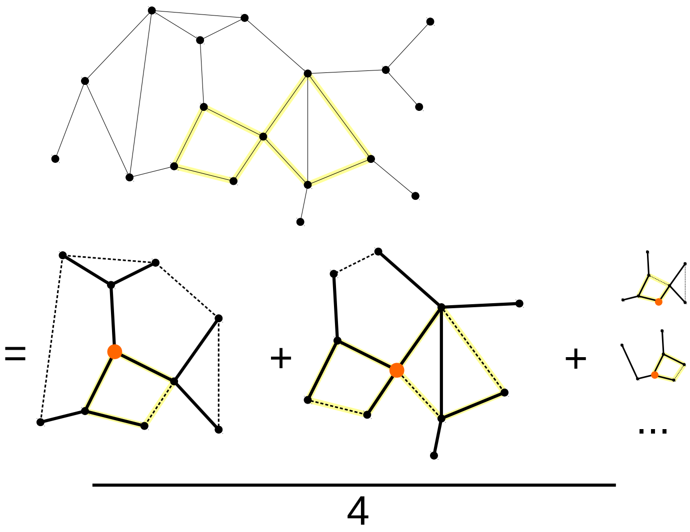
Example: Counting the number of squares



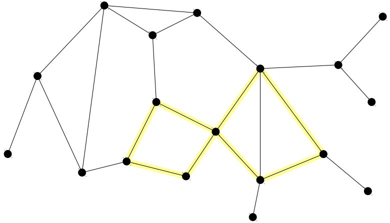
Example: Counting the number of squares

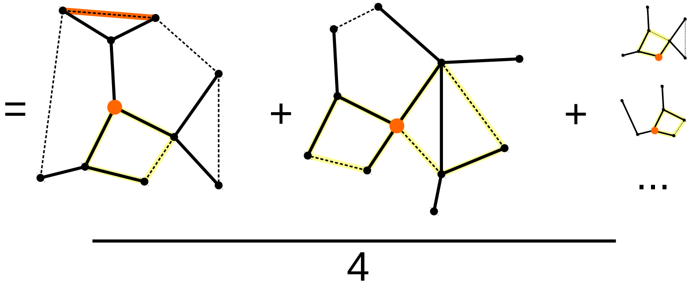


Example: Counting the number of squares

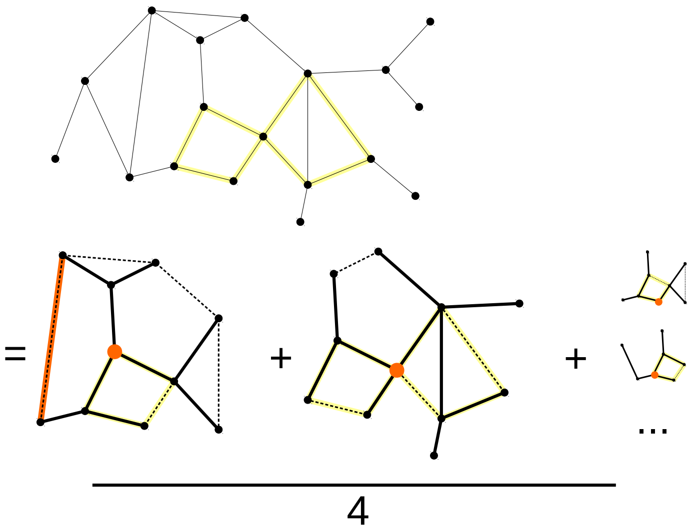


Example: Counting the number of squares

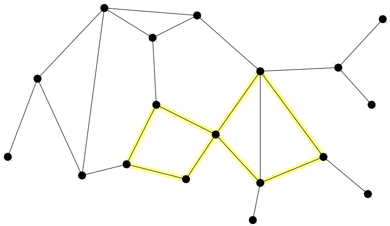


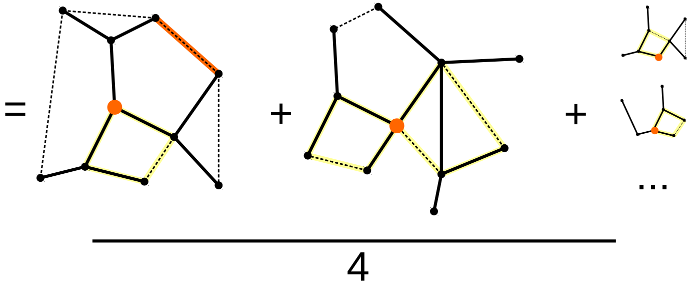


Example: Counting the number of squares

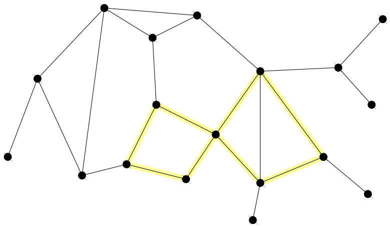


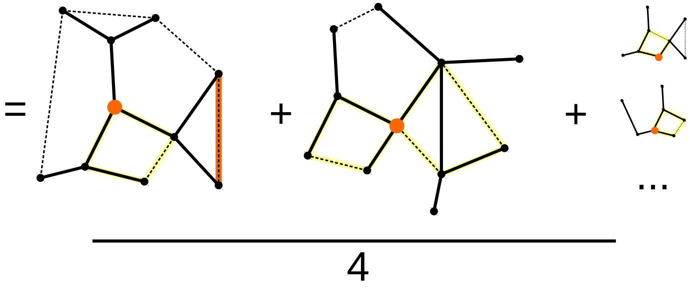
Example: Counting the number of squares



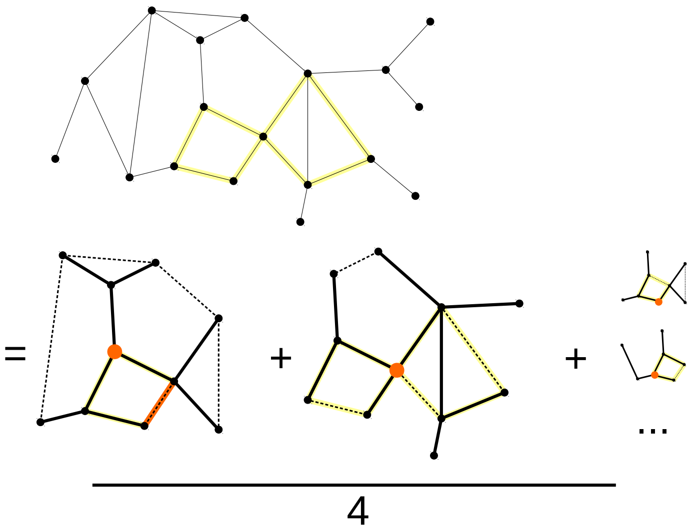


Example: Counting the number of squares

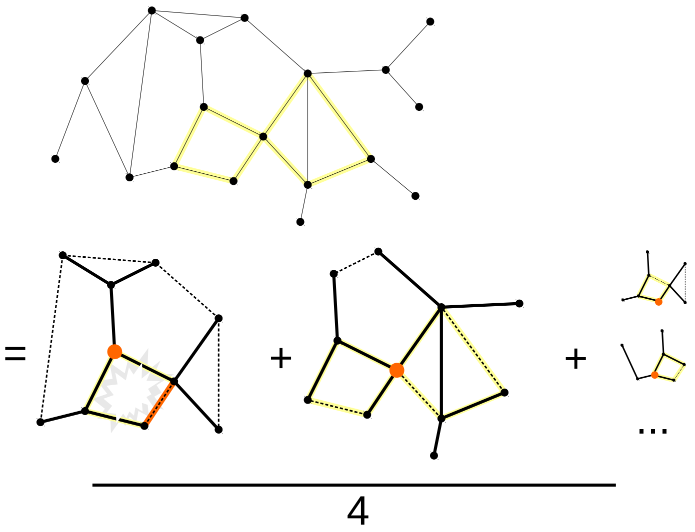




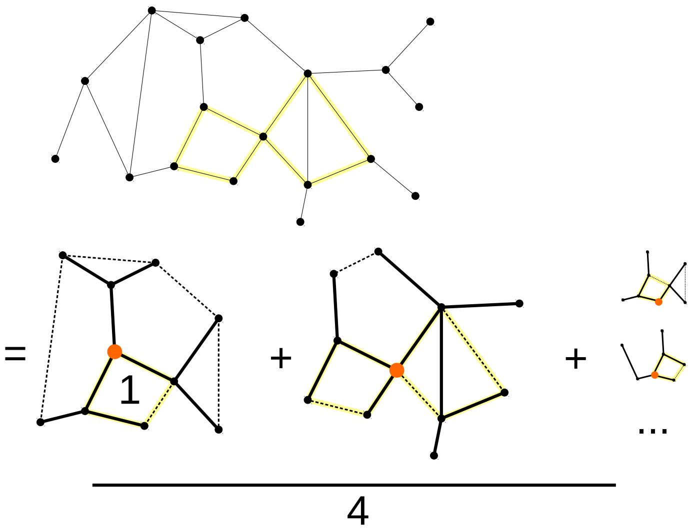
Example: Counting the number of squares



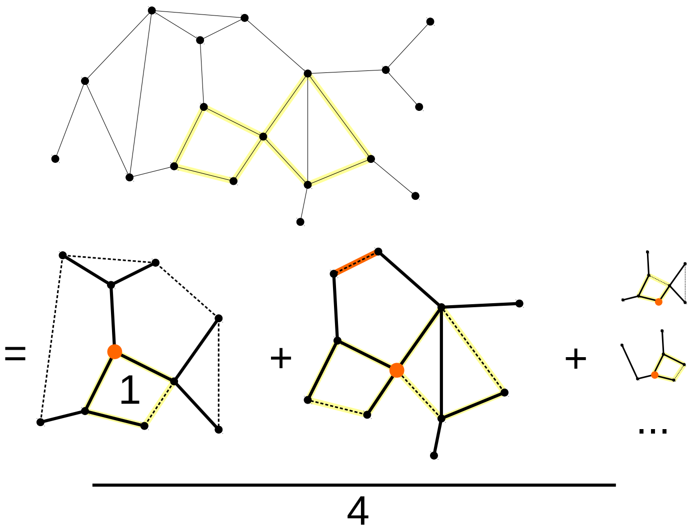
Example: Counting the number of squares



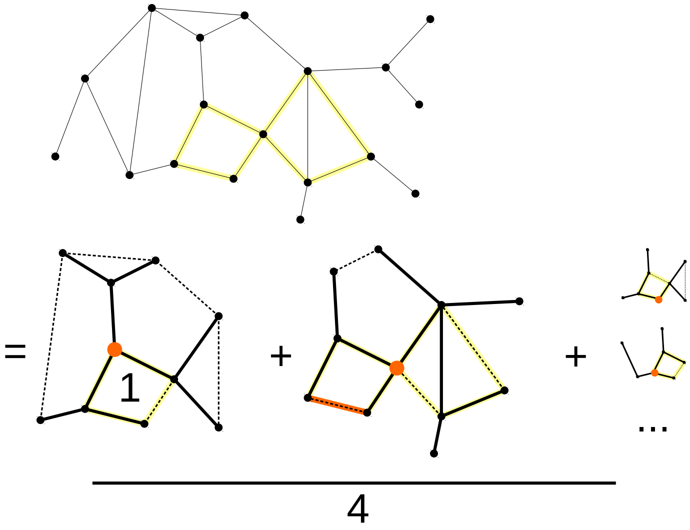
Example: Counting the number of squares



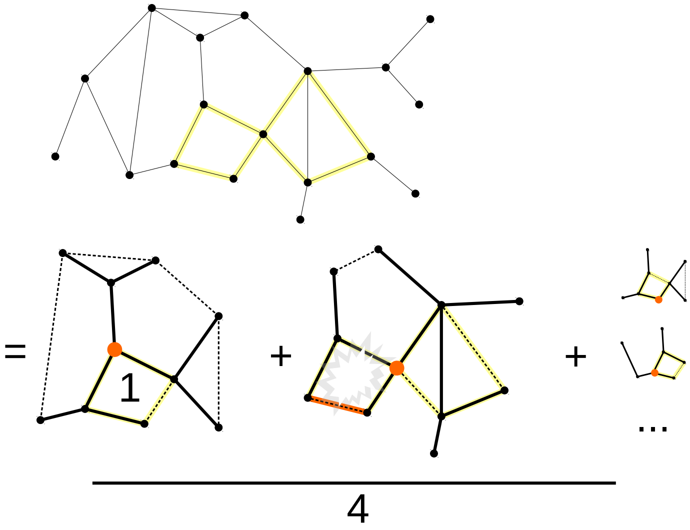
Example: Counting the number of squares



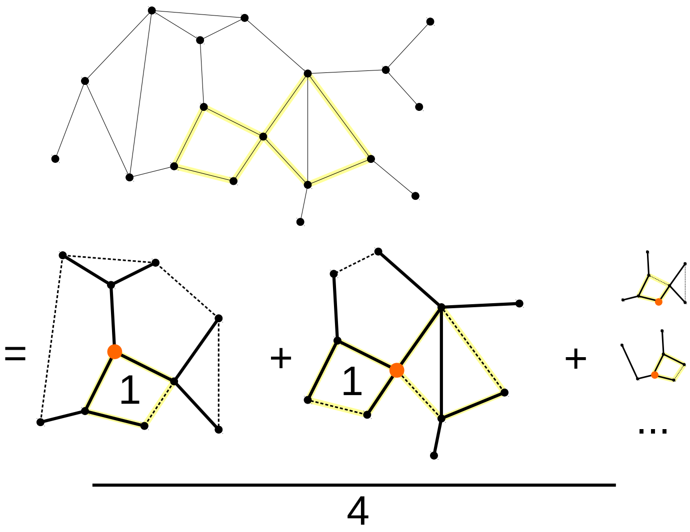
Example: Counting the number of squares



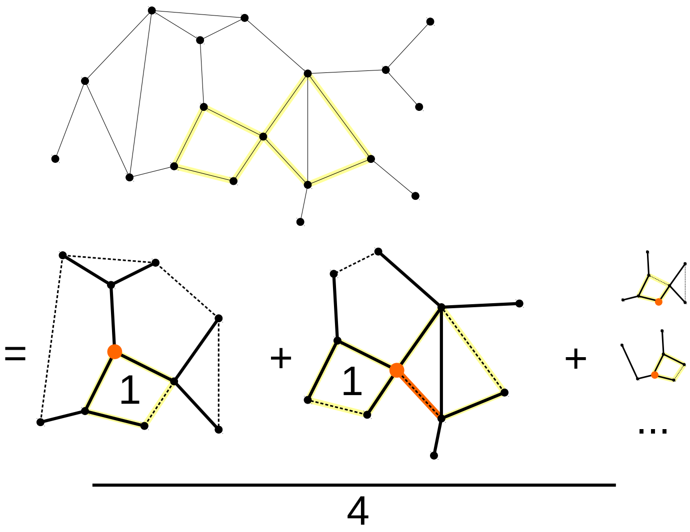
Example: Counting the number of squares



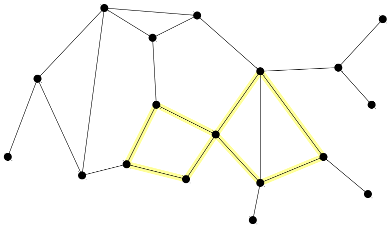
Example: Counting the number of squares

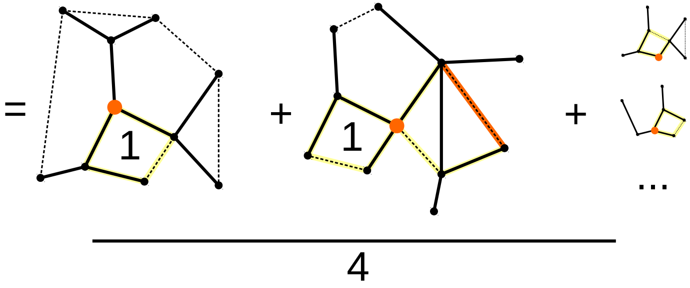


Example: Counting the number of squares

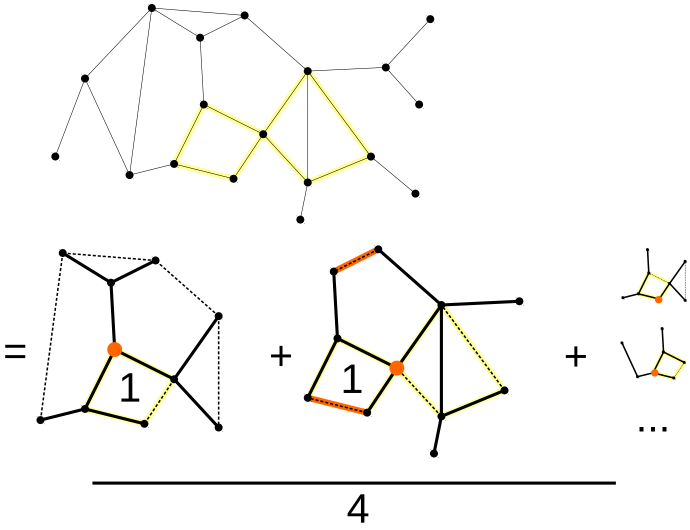


Example: Counting the number of squares

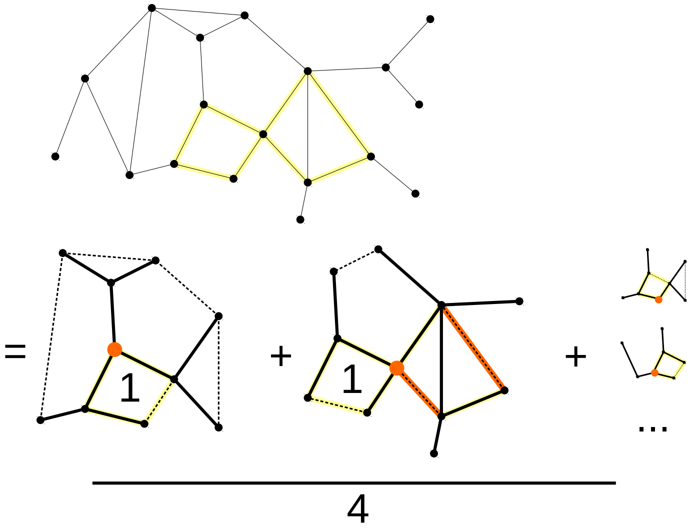




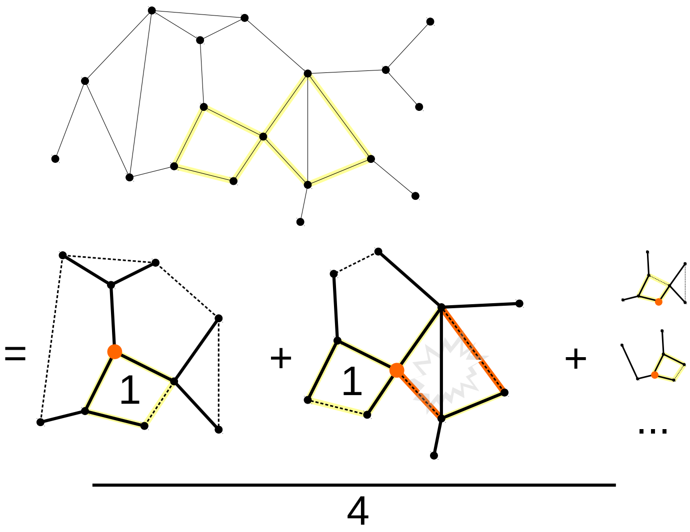
Example: Counting the number of squares



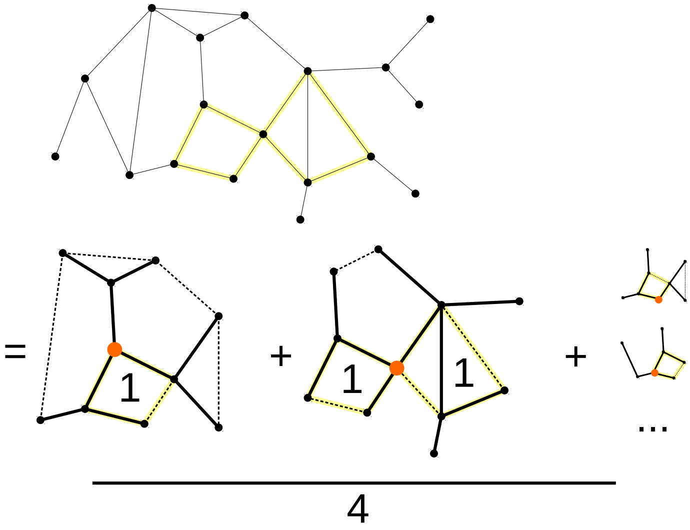
Example: Counting the number of squares



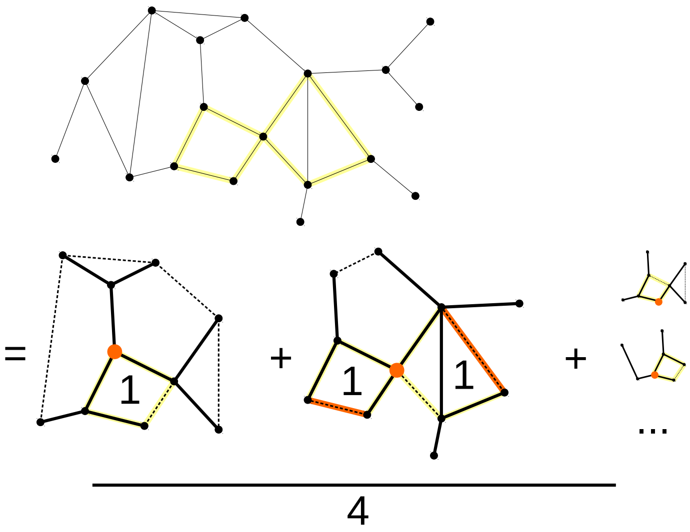
Example: Counting the number of squares



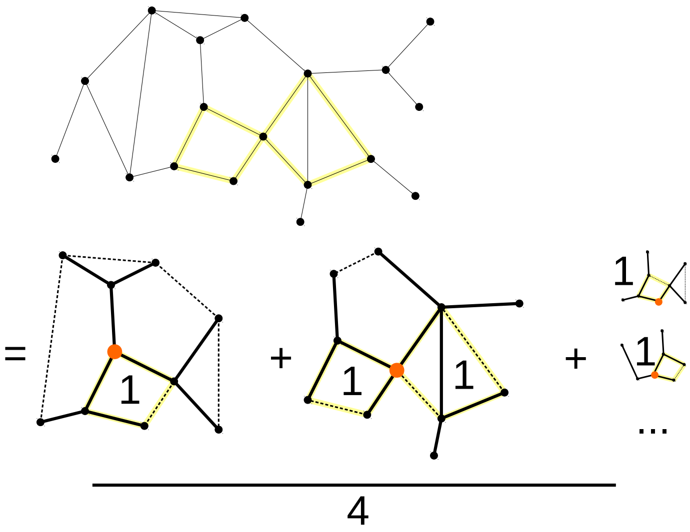
Example: Counting the number of squares



Example: Counting the number of squares

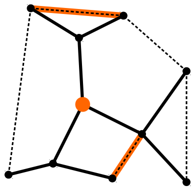


Example: Counting the number of squares



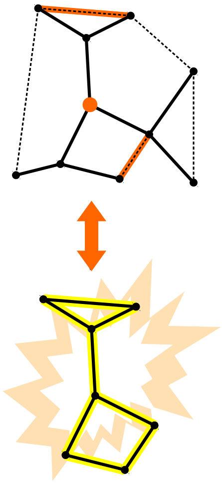
Bounding the Run Time

We need to bound the number
of subroutine calls per neighborhood.



We need to bound the number of subroutine calls per neighborhood.

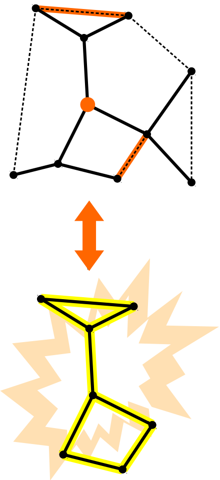
- For every subroutine call there is exactly one small leaf-free subgraph.



Bounding the Run Time

We need to bound the number of subroutine calls per neighborhood.

- For every subroutine call there is exactly one small leaf-free subgraph.
- Number of subroutine calls bounded by number of small leaf-free subgraphs.

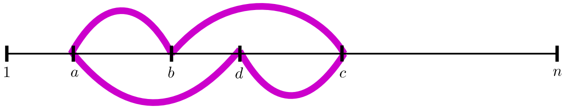


Bounding Number of Leaf-Free Subgraphs

expectation of $\# \square$ in G_m^n

Bounding Number of Leaf-Free Subgraphs

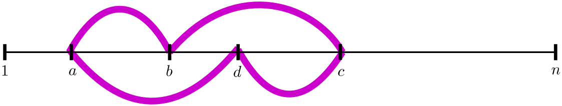
expectation of $\# \square$ in G_m^n

$= \Pr[$  $]$

Bounding Number of Leaf-Free Subgraphs

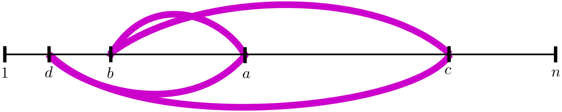
expectation of $\# \square$ in G_m^n

$= \Pr[$



$]$

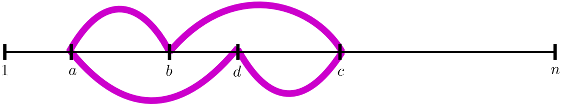
$+ \Pr[$

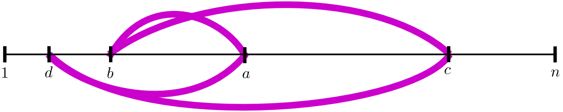


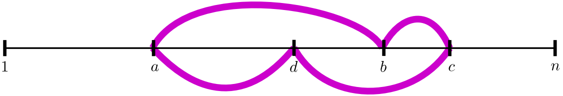
$]$

Bounding Number of Leaf-Free Subgraphs

expectation of $\# \square$ in G_m^n

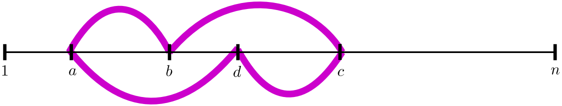
$= \Pr[$  $]$

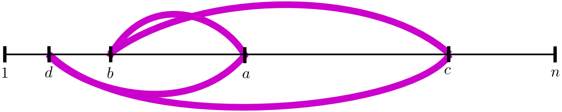
$+ \Pr[$  $]$

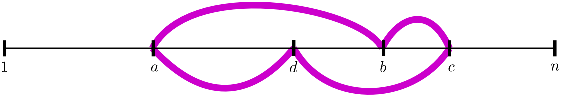
$+ \Pr[$  $]$

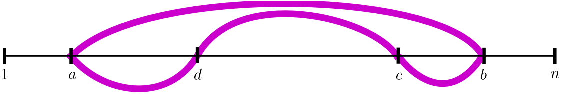
Bounding Number of Leaf-Free Subgraphs

expectation of $\# \square$ in G_m^n

$= \Pr[$  $]$

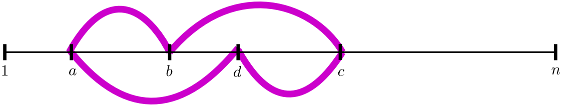
$+ \Pr[$  $]$

$+ \Pr[$  $]$

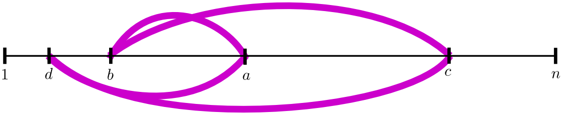
$+ \Pr[$  $]$

Bounding Number of Leaf-Free Subgraphs

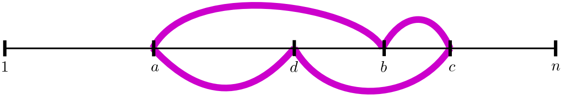
expectation of $\# \square$ in G_m^n

$= \Pr[$ 

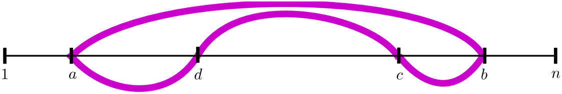
$]$

$+ \Pr[$ 

$]$

$+ \Pr[$ 

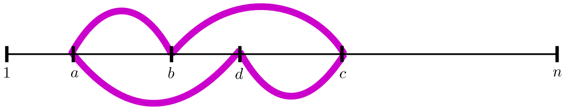
$]$

$+ \Pr[$ 

$]$

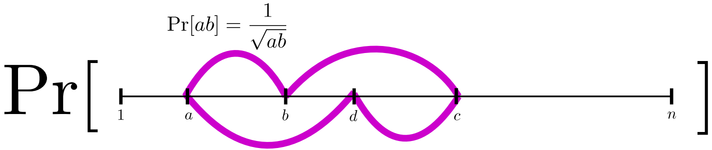
$+ \cdots$

Bounding Number of Leaf-Free Subgraphs

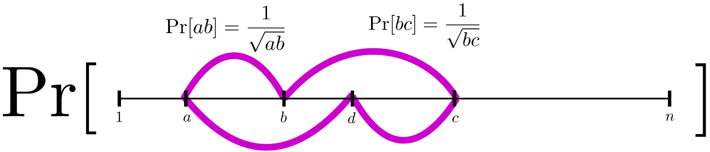
$\Pr[$ 

$]$

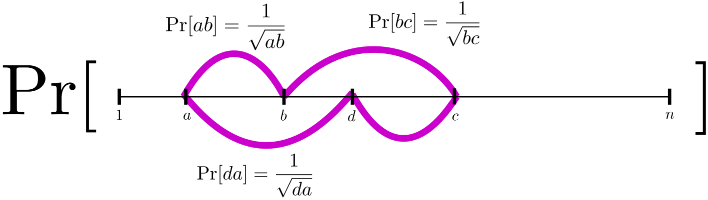
Bounding Number of Leaf-Free Subgraphs



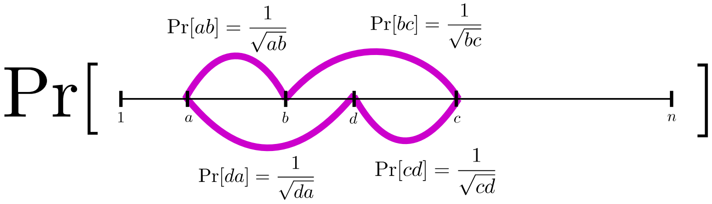
Bounding Number of Leaf-Free Subgraphs



Bounding Number of Leaf-Free Subgraphs

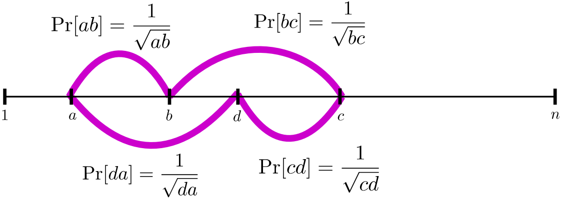


Bounding Number of Leaf-Free Subgraphs



Bounding Number of Leaf-Free Subgraphs

Pr[



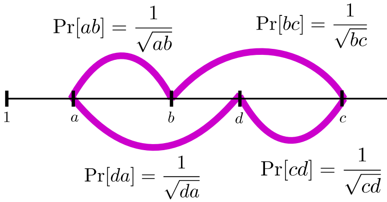
]

$$= \Pr[ab] \cdot \Pr[bc] \cdot \Pr[cd] \cdot \Pr[da]$$

$$= \frac{1}{abcd}$$

Bounding Number of Leaf-Free Subgraphs

Pr[



]

$$= \log(n)^{O(1)} \cdot \Pr[ab] \cdot \Pr[bc] \cdot \Pr[cd] \cdot \Pr[da]$$

$$= \frac{\log(n)^{O(1)}}{abcd}$$

Bounding Number of Leaf-Free Subgraphs

$$\text{expectation of } \# \square \text{ in } G_m^n$$

Bounding Number of Leaf-Free Subgraphs

$$\text{expectation of } \# \square \text{ in } G_m^n$$

$$= \sum_{a=1}^{n}$$

Bounding Number of Leaf-Free Subgraphs

$$\text{expectation of } \# \square \text{ in } G_m^n$$

$$= \sum_{a=1}^{n} \sum_{b=1}^{n}$$

Bounding Number of Leaf-Free Subgraphs

$$\text{expectation of } \# \square \text{ in } G_m^n$$

$$= \sum_{a=1}^{n} \sum_{b=1}^{n} \sum_{c=1}^{n}$$

Bounding Number of Leaf-Free Subgraphs

expectation of $\# \square$ in G_m^n

$$= \sum_{a=1}^{n} \sum_{b=1}^{n} \sum_{c=1}^{n} \sum_{d=1}^{n}$$

Bounding Number of Leaf-Free Subgraphs

expectation of $\# \square$ in G_m^n

$$= \sum_{a=1}^{n} \sum_{b=1}^{n} \sum_{c=1}^{n} \sum_{d=1}^{n} \frac{\log(n)^{O(1)}}{abcd}$$

Bounding Number of Leaf-Free Subgraphs

$$\text{expectation of } \#\square \text{ in } G_m^n$$

$$= \sum_{a=1}^{n} \sum_{b=1}^{n} \sum_{c=1}^{n} \sum_{d=1}^{n} \frac{\log(n)^{O(1)}}{abcd}$$

$$= \log(n)^{O(1)} \sum_{a=1}^{n} \frac{1}{a} \sum_{b=1}^{n} \frac{1}{b} \sum_{c=1}^{n} \frac{1}{c} \sum_{d=1}^{n} \frac{1}{d}$$

Bounding Number of Leaf-Free Subgraphs

expectation of $\# \square$ in G_m^n

$$= \sum_{a=1}^{n} \sum_{b=1}^{n} \sum_{c=1}^{n} \sum_{d=1}^{n} \frac{\log(n)^{O(1)}}{abcd}$$

$$= \log(n)^{O(1)} \sum_{a=1}^{n} \frac{1}{a} \sum_{b=1}^{n} \frac{1}{b} \sum_{c=1}^{n} \frac{1}{c} \sum_{d=1}^{n} \frac{1}{d}$$

$$= \log(n)^{O(1)}$$

(similarly for every leaf-free graph)

Total Runtime

Analysis of run time:

Total Runtime

Analysis of run time:

- Number of vertices: $O(n)$

Total Runtime

Analysis of run time:

- Number of vertices: $O(n)$
- Subroutine calls per vertex: $\log(n)^{O(1)}$

Total Runtime

Analysis of run time:

- Number of vertices: $O(n)$
- Subroutine calls per vertex: $\log(n)^{O(1)}$
- Time for each call: $O(n)$

Total Runtime

Analysis of run time:

- Number of vertices: $O(n)$
- Subroutine calls per vertex: $\log(n)^{O(1)}$
- Time for each call: $O(n)$

Total run time: $\log(n)^{O(1)} n^2$

Total Runtime

Analysis of run time:

- Number of vertices: $O(n)$
- Subroutine calls per vertex: $\log(n)^{O(1)}$
- Time for each call: $O(n)$

Total run time: $\log(n)^{O(1)} n^2$

With a more careful analysis: $\log(n)^{O(1)} n$

Related: FPT First-Order Model Checking

First-Order Model Checking

- $\exists$ and $\forall$ quantification over vertices, equality, adjacency
- cannot be solved in time $f(|\varphi|)n^{o(|\varphi|)}$ under ETH
- *Grohe, Kreutzer, Siebertz:* for fixed formula decidable in time $O(n^{1+\varepsilon})$ on nowhere-dense graph classes

Theorem

Let $\varepsilon > 0$, $m \in \mathbf{N}$ be fixed.
One can decide every property expressible in FO logic on G^n_m in expected time $O(n^{1+\varepsilon})$.

Thank you!