# Algorithms for counting triangles and other graphlets

## Motivation

- Counting triangles is a fundamental problem in graph theory
- Some problems require exact algorithms
- Exact algorithms are needed to evaluate approximate algorithms
  - Exact counting is slow. What are some possible euristic optimizations?
- Counting graphlets is a more general problem with many applications:
  - Discover interesting properties of biological graphs
  - Useful feature for machine learning at the node/graph level
  - many more!
- Can we extendend triangle couting algorithms to other graphlets?
  - Again, what are some possible euristic optimizations?

## Methods

- Start by implementing algortihms seen in class
- Look for euristic optimizations, still giving exact results
  - Known limitation: we'll need to load all edges in memory
- Compare running times across graphs of different sizes
  - What parameters influence running time? `n`, `m`, `delta`, ...
- Explore possible generalizations of triangles counting algorithms to other graphlets
  - Counting 2-pointeed-stars shoud be similar to counting triangles
  - Can we count graphlets of size `k=4` using results from `k=3`?
- Possibly compare the results with approximate algorithms

## Intended experiments

- Use graphs from `https://networkrepository.com/`
  - Choose graphs spanning different orders of magnitude in size
  - Nomralize the input format
- Implement algorithms from scratch (C++)
  - Use pseudocode from class as a guide for known algorithms
  - Look for possible heuristic optimizations
  - Search the literature for possilbe other optimizations
- Measure algorithm efficiency
  - Measure running time
  - Measure memory usage
  - Compare running times across graphs of different sizes
  - Compare results with theoretical expectations
- Extend the implementation to other graphlets
  - Try to generalize the algorithms already implemented for triangles
- Machine for experiments: our personal laptop

Idea for an heuristic optimization: sort the nodes by degree, start counting from the nodes of lowest degree, delete nodes and incident edges as they are processed. Initial results are promising:

| Graph | \|V\| | \|E\| | Triangles | Time Naive | Time Heuristic | Naive/Opt |
|---|---|---|---|---|---|---|
| soc-wiki-Vote | 882 | 2914 | 2119 | 0.026s | 0.003s | 8.66 |
| fb-pages-politician | 5908 | 41729 | 174632 | 0.827s | 0.153s | 5.40 |
| soc-twitter-follows-mun | 465017 | 835423 | 38389 | 76.415s | 2.245s | 34.04 |
| soc-youtube-snap | 1134890 | 2987624 | 3056386 | 657.263s | 11.962s | 54.94 |
| soc-flixster | 2523386 | 7918801 | 7897122 | 752.680s | 36.171s | 20.80 |

Here, *Time Naive* refers to the time taken by the `O(m*delta)` algorithm seen in class, and *Time Heuristic* to the time taken by our heuristic optimization.

## References


- Slides from class
- *Graphlet decomposition: framework, algorithms, and applications* (2016) offers a great review of algorithms and applications for graphlets