Triangle Counting

Triangles are particularly useful
→ why: clustering, link prediction, vertex classification, prediction

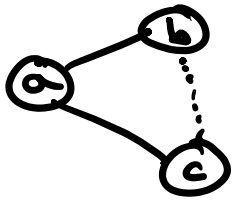
Also: most real networks have substantial clustering

Can be easier to count/enumerate than larger templates/subgraphs

$O(n^w)$ ← exponent of fast matrix mult.

$O(n^k)$ for subgraphs in general

↳ essentially computing closure of all wedges



← closure

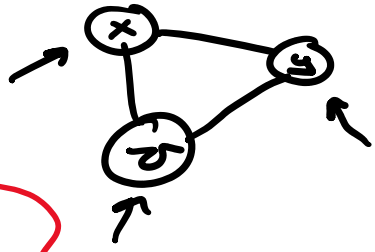
Sparse graphs: $m \ll n^2$

we have $O(m^{3/2})$

↑ working on edge list itself

Basic Algorithm:

```
Count = 0
For all v ∈ V(G):
    For all u, w ∈ N(w)
        if (u, w) ∈ E(G)
            Count ++
Count /= 3
        (or 6)
```



How these lines are implemented in practice differs between 1000s of implementations

---

Template Matching

/

↳ generic term for subgraphs w.r.t. subgraph isomorphism

We consider some template T and search for isomorphic subgraphs in G

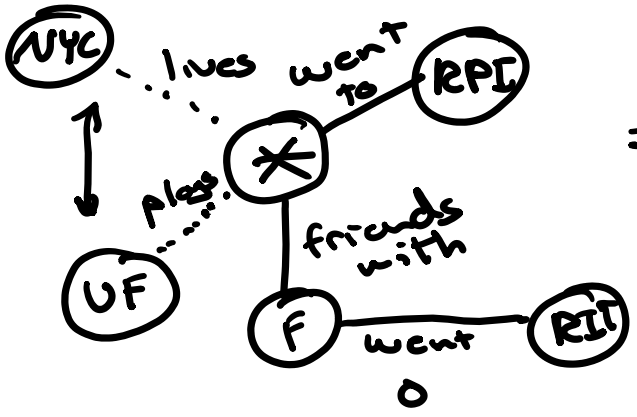
Note: we also can consider T as labeled

→ Luckily, it makes the problem much easier

Template search applications

Facebook Graph Search

↳ "Find all people who went to RPI and are friends with someone who went to RIT and currently live in NYS OR currently play ultimate frisbee"



=> greatly reduces search

stalkers, int. agencies, hackers, etc.

Unfortunately: immediately abused and removed

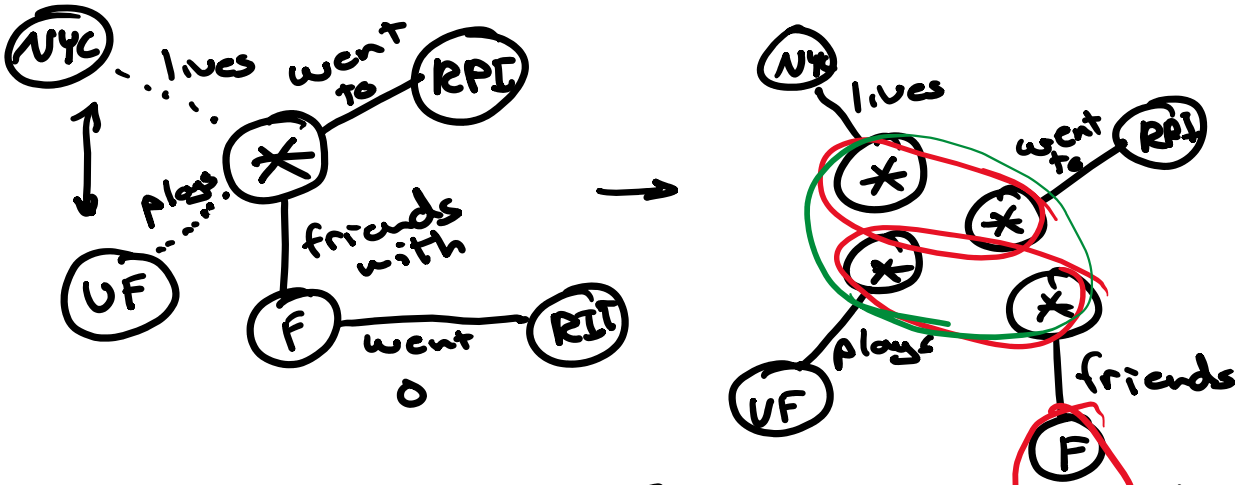
Also widely applicable:

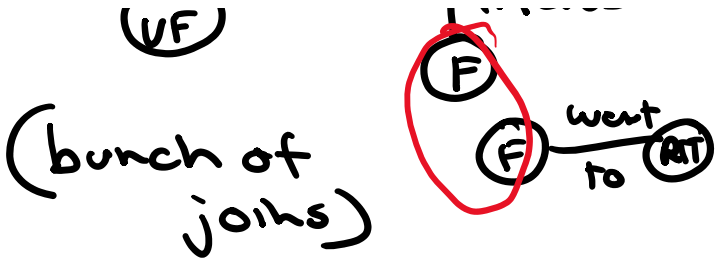
- social network, financial networks, info nets, etc

Algorithms for template search:

Naive brute force: $O(n^k)$

Dynamic Programming:





(bunch of joins)

Worst case still there

↳ but can be fast in practice

Approximation Algorithms

We can improve computational cost

But: we either aren't getting an exact count OR we aren't enumerating all subgraphs

How: sampling, color-coding

↗ only search a portion of G

↓

Necessary to be careful with how sampling is done

↗ color G with at least k colors, find "colorful" embeddings

↗ all vertices have the same color

sampling is done
(skewed D.D., etc.)

Motif Finding

↳ a subgraph that occurs more frequently than expected
↳ based on a null model

Why would it be more frequent:

Biological networks: functional reason → why/how proteins interact

Social networks: the way the network grows → triangles

Financial networks: common transactional patterns

(Local bank) → (global bank) → (IRA account)

Note: we can have both motifs and antimotifs

↳ anomaly detection

↳ Finding patterns/subgraphs that occur less frequently than expected

E.g., an atypical financial transaction pattern might indicate money laundering/fraud/etc.

Interesting Open Problem:

cortical conjecture.

↳ Consider a brain graphs

structure: how neurons connect

functional: higher-level connections based on observed

based on observed functional interactions

→ connectomes

Cortical conjecture: there's repeating computational substructures of the connectome that explains "intelligence" or higher-order thought

(open problem)

Generally, how we solve/find motifs in networks

→ count occurences of template T in network A ← one we care about

→ count occurences of template T in network(s) B ← null model or other relevant network

(Repeat for various T)

$C(T, A) >>>>> C(T, B)$ → motif in A

$C(T, A) <<<<<< C(T, B)$ → antimotif in A

Note: we can compare graph similarity base on various subgraph counts

Local: between regions of a graph

Global: comparing full graphs

Similarity measures we've seen:

→ spectral methods

→ embeddings (GCNs) (vector of values)

Using subgraph counts as a similarity measure:

→ consider same # of templates

→ count em up

→ use these counts as a measure

→ use these counts as a measure of similarity (normalized or not)

Explicitly: subgraph frequency distance

(Globally) - Given G and H

- define $N_i(G)$ as counts of template i in G
- define $T(G)$ as total count over all i
- define $F_i(G) = \log\left(\frac{N_i(G)}{T(G)}\right)$

we have: $D(G,H) = \sum_i |F_i(G) - F_i(H)|$

similarity measure: $S(G,H) = 1 - D(G,H)$ (or 2)

Uses: graph classification
null model selection

Q: (what template(s) ...)

Q: what templates / subgraphs

A: graphlets and others

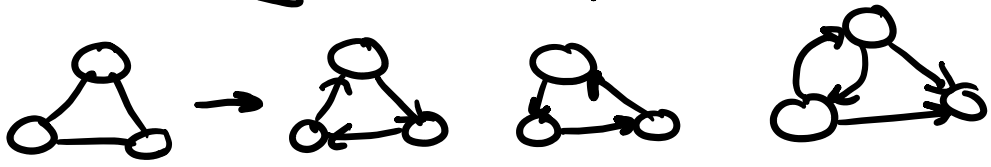
↳ undirected 3-5 vertex connected induced subgraphs

→ small enough to compute

→ still varied topology

→ useful enough for practical purposes

Directed: many more options



etc.

Another option: treelets

(beyond graphlets) ↳ (Slota special) tree templates from 3-9 vertices

Bigger → captures large neighborhood

neighborhood

Faster → much faster to count

Getting Local

↳ vertex similarity measures using subgraph

Really: just using our global measure <u>rooted</u> at a vertex v

How: we <u>root</u> the subgraphs and count embedding at v for that root → repeat for all roots

Consider 

→ we need to count for all automorphically distinct roots at v

Subgraph Degree Signature:

- Consider vertices $u, v \in V(G)$
- Consider all possible orbits

- Consider all possible orbits across all templates considered
- For orbit i with counts u_i, v_i

$$D_i = w_i \frac{|\log(u_i+1) - \log(v_i+1)|}{\log(\max(u_i, v_i)+2)}$$

w_i: weighting based on # of automorphism

Overall:

$$D(u,v) = \sum_i \frac{D_i(u,v)}{\sum_i w_i}$$

$D(u,v)$: from 0...1

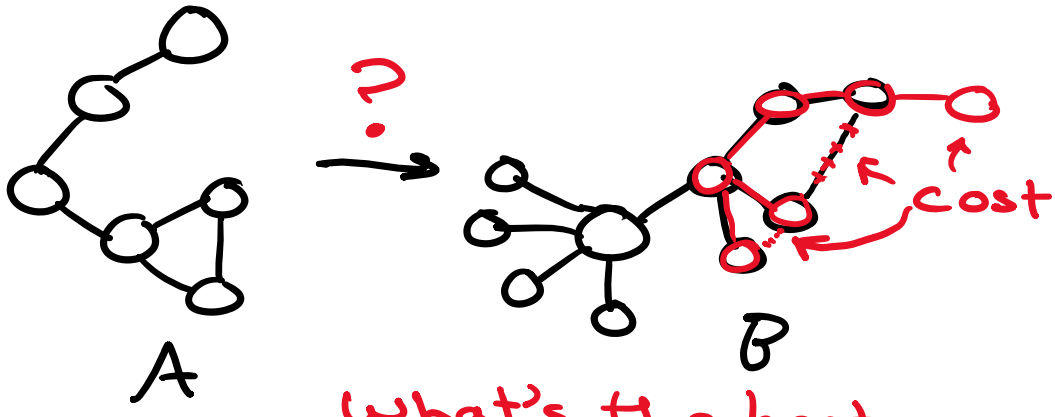
$$S(u,v) = 1 - D(u,v)$$

$S(u,v)$: similarity measure

Uses: vertex classification
vertex embedding
regional analysis within a network
Graph Alignment

Graph Alignment

↳ Think of it as approximate (sub) graph isomorphism



What's the best mapping?

We generally define some <u>cost</u> and try to minimize it

→ many differing cost metrics across fields/data (e.g. structural)

→ Functional metrics → explicit cost measures of functional similarity

→ Structural metrics

↳ edge/vertex deletion/addition or contraction

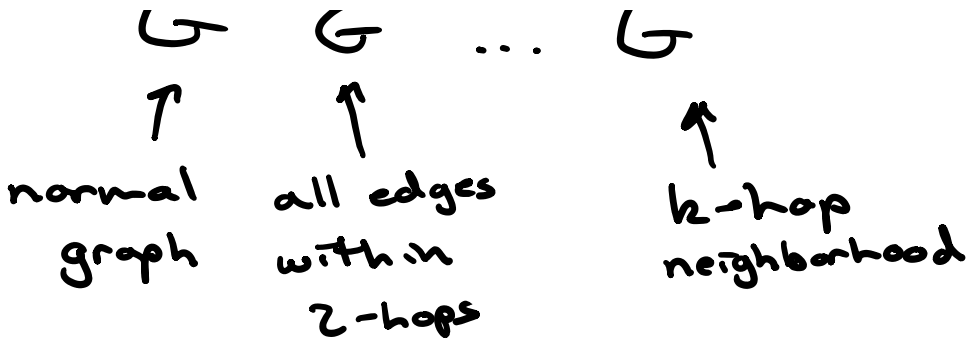
Note: graph alignment is generally NP-hard as an optimization problem

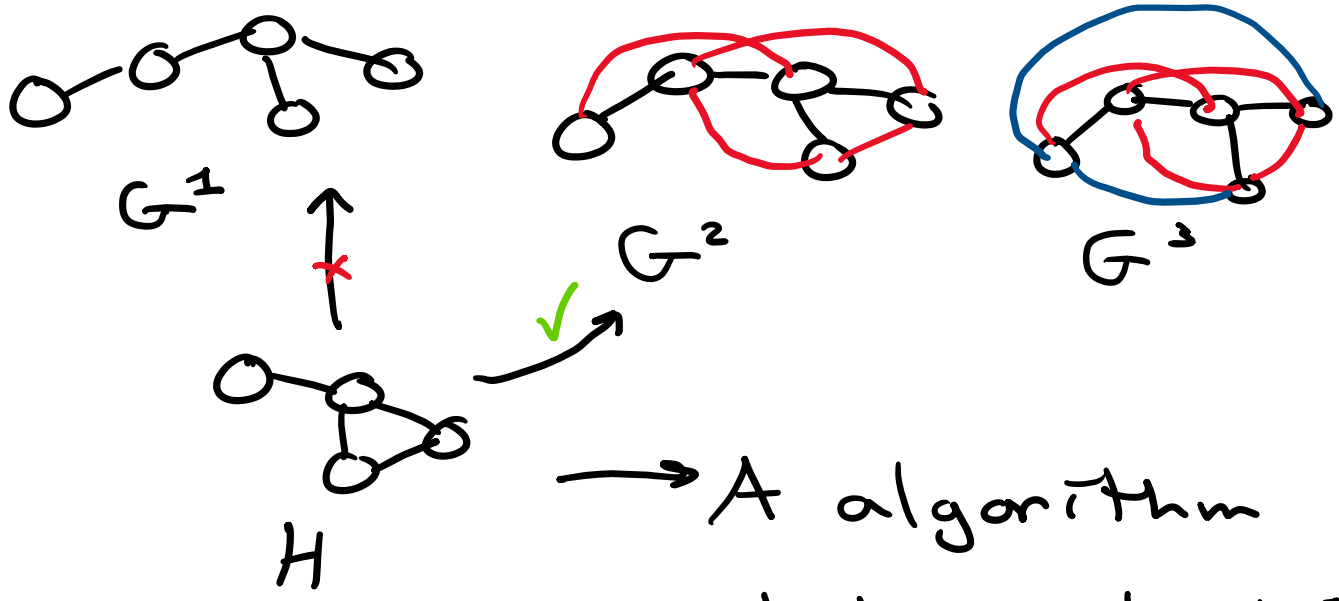
How can we use subgraph similarities for the graph alignment problem:

- We want to align $G \to H$
- we count all orbits and compute pairwise similarity score between all $v \in V(G)$ and $u \in V(H)$
- we then heuristically align using highest similarity vertex pairs

Note: accounting for missing vertices or edges

G^1 G^2 ... G^k





→ A algorithm

→ select highest similarity pair of vertices (u, v)

→ greedily align G, H from u, v

→ If not complete, then try on G^2, G^3 etc.