Recall: Graph Mining is...

Link Prediction

Centrality

Clustering/CD

Vertex classification

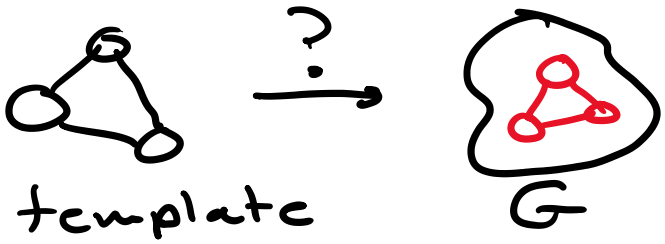
Now: subgraph mining

you can tell it's relevant to graph mining

Many different problem fit under this umbrella

- Triangle counting (clustering/cc, triadic closure etc)
- Template matching
  aka subgraph isomorphism

aka subgraph isomorphism



also: subgraph counting/enumeration
↳ How many? ↳ where?

- Motif/antimotif detection

  what subgraphs show up more/less frequently than expected

  aka H3p2

- subgraph-based comparative analytics

  compare two or more graph via constituent subgraphs (usually normalized)

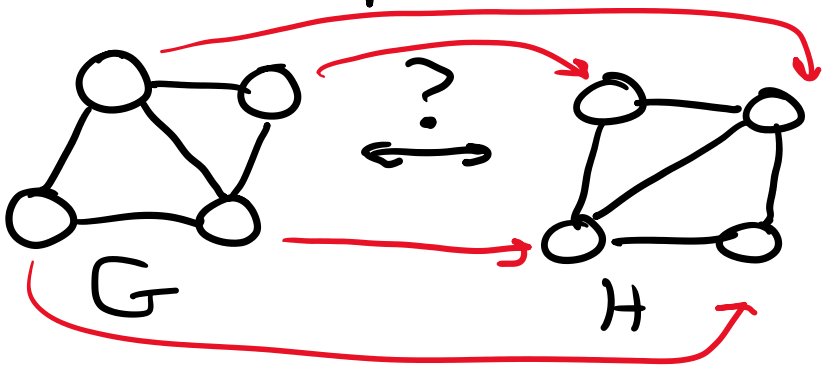
- graph alignment (next class)

  approximate (sub)graph isomorphism

  we align networks based on some cost function

Subgraph isomorphism

Graph isomorphism



Are G and H equivalent?

↳ Tougher problem than you might think

(complexity unknown)

Usually: algorithms turn G and H into strings and compare them

Useful in (bio)chemistry

What we care more about:

Subgraph isomorphism subgraph

Subgraph isomorphism

subgraph → |V(T)|

Good

ton of applications

broadly useful

computationally feasible (in certain cases)

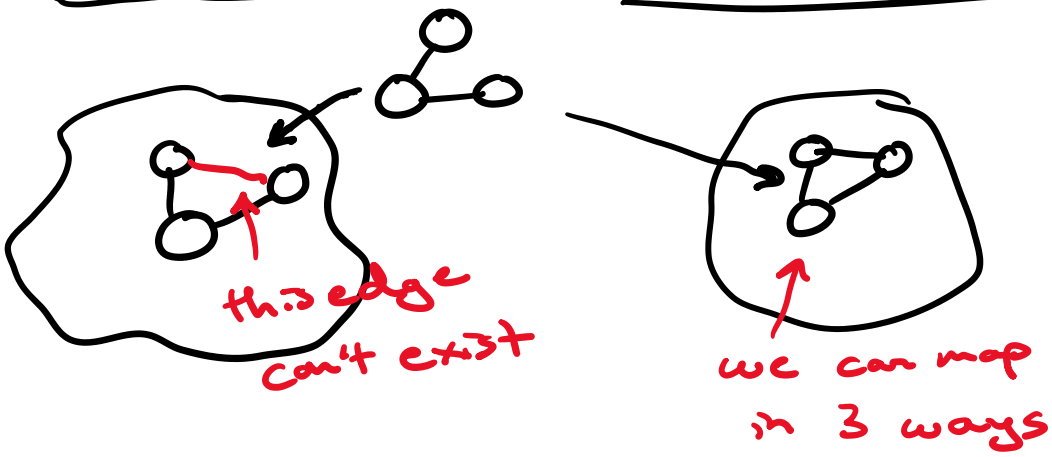
Bad

NP-complete $O(n^{|V(T)|})$ ← $|V(G)|$

↳ we need to solve for all subgraph-based analytics

However: can use approximating counts

Induced vs. non-induced subgraphs



Triangle Counting

Triangle are particularly useful

→ Why? very useful for clustering analytics, prediction, etc,

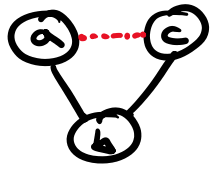
analytics, prediction, etc,

Also: most real, interesting, large networks exhibit some form of clustering

Can be easier to count/enumerat than larger templates

$O(n^\omega)$ ← exponent of fast matrix multiplication complexity

↳ essentially computing closure of all wedges



Sparse graphs: $m << n^2$

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

↑ operations on edge list itself

Basic Algorithm:

Count = 0

For all $v \in V(G)$:

For all u, w in $N(v)$:

if $(u,w) \in E(G)$:

if $(u,w) \in E(G)$:
count += 1

count /= 6
(or 3)

Most general-purpose subgraph counting algorithms use some variant of the above

---

Template matching

↳ general term for "subgraph isomorphism"

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

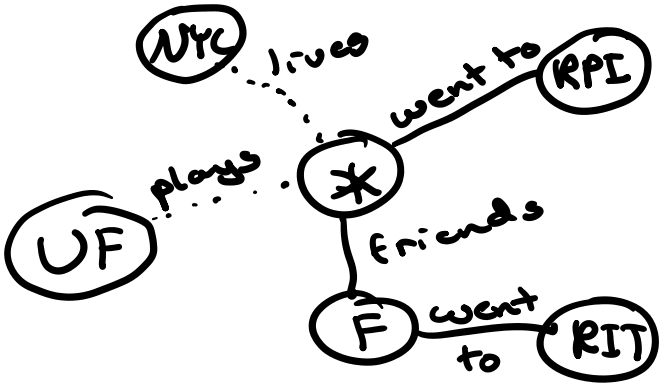
Note: we can also consider labels on our template

↳ Luckily, it makes the problem a lot 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 NYC OR currently play ultimate frisbee"



⇒ greatly restricts search space

Unfortunately: immediately abused and removed

stalkers, int. agency hackers, etc.

Also widely applicable:

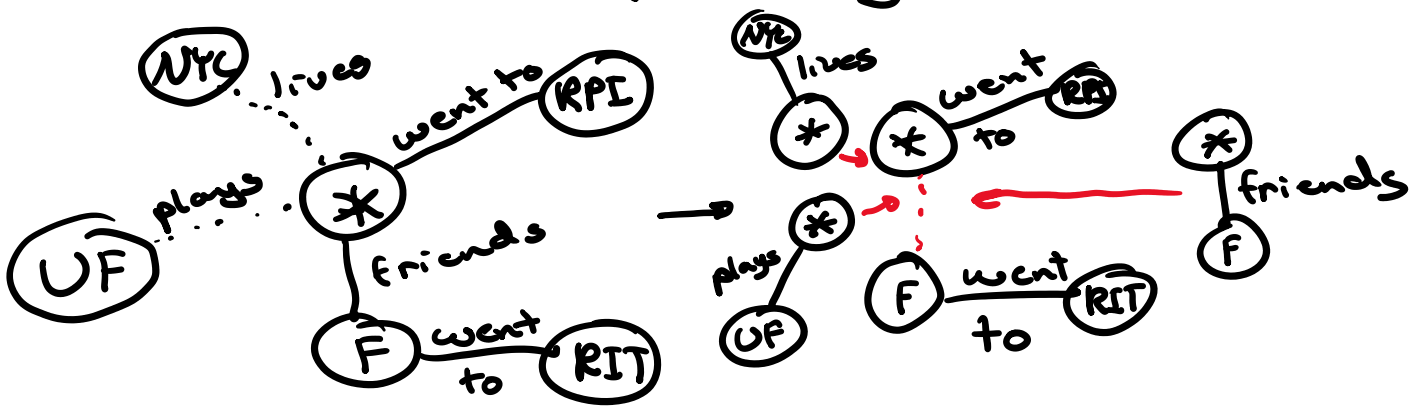
- Social networks, financial networks
  Information networks, ↑ flow of money
  ETC.

Algorithms for template search:

Naive brute force → $O(n^k)$

Dynamic Programming

→ Better, especially for labeled data



However: worst-case complexity still the same

Approximate Algorithms:

We can actually improve our bound

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 the

↑ we color G randomly with at least k colors,

portion of the network

with at least k colors, and only search for "colorful" embedding

↑ all vertices have a different color

One more thing: Parallelization

↓

applies to all the above

=> generally, parallelizing where in the graph we're searching