Optimizing cascades & submodular optimization

Rik Sarkar

Today

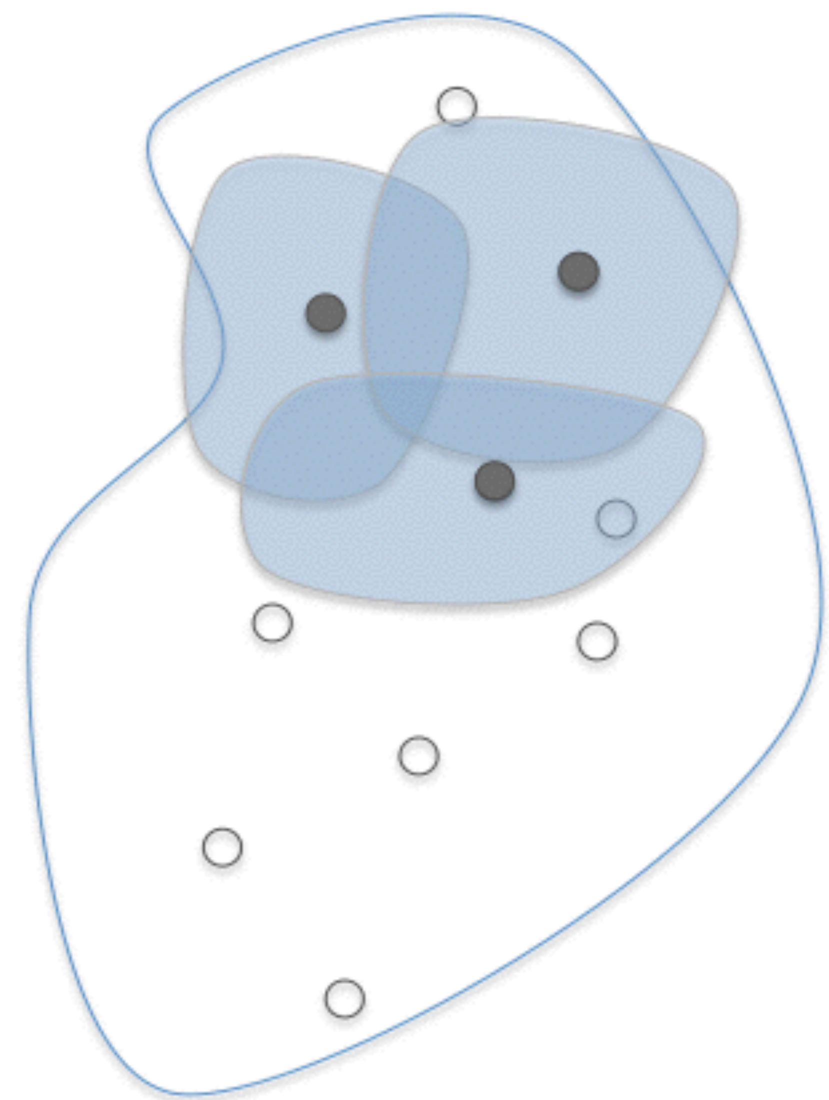
- Maximizing cascades
- Other applications of submodularity
- Network flows
- NP completeness

Recap: Selecting nodes to activate

- We have a network of n nodes
- And a budget to activate k nodes
- Which k nodes should we activate to get the largest cascade?
- Hard problem, we want approximate solutions

Recap: Submodular maximization

- Submodular function f:
  - Value added by an item decreases with bigger sets
- Find the set S of size k that maximizes f(S)



$$S \subseteq T \implies$$

$$f(S \cup \{x\}) - f(S) \geq f(T \cup \{x\}) - f(T)$$

Recap: Approximation

- A simple greedy algorithm:
  - In next round, pick the item that gives the largest increase in value
- For monotone submodular maximization, the greedy algorithm gives $\left(1 - \frac{1}{e}\right)$ approximation

Cascade

- Cascade function f(S):
  - Given set S of initial adopters, f(S) is the number of final adopters
- We want to show: f(S) is submodular
- Idea: Given initial adopters S, let us consider the set H that will be the corresponding final adopters
  - H is “covered” by S

Cascade in independent activation model

- If node u activates to use A, then u causes neighbor v to activate and use A with probability
    - $p_{u,v}$
- Now suppose u has been activated
    - Neighbor v will be activated with prob. $p_{u,v}$
    - Neighbor w will be activated with prob. $p_{u,w}$ etc..
    - Instead of waiting for u to be activated before making the random choices, we can make the random choices beforehand
    - ie. if u is activated, then v will be activated, but w will not be activated… etc

Cascade in independent activation model

- We can make the random choices for u activation beforehand.
- Tells us which edges of u are “effective” when u is “on”
- Similarly for other nodes v, x, y ….
- We know exactly which nodes will be activated as a consequence of u being activated
- Exactly the same as “coverage” of a sensor network
- Say, c(u) is the set of nodes covered by u.

- We know exactly which nodes will be activated as a consequence of u being activated
- Exactly the same as “coverage” of a sensor network
- Say, $c(u)$ is the set of nodes covered by u.
  - $c(S)$ is the set of nodes covered by a set S
- $f(S) = |c(S)|$ is submodular

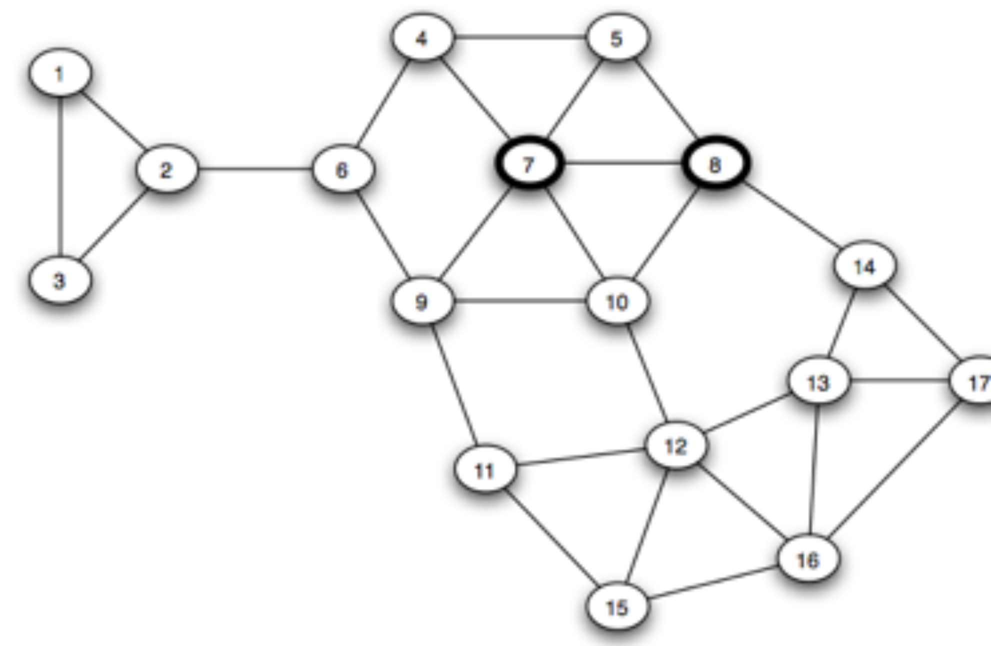
- Remember that we had made the probabilistic choices for each edge uv:
  - With probability $p_{u,v}$ we set the edge to be "active": if u is activated, v will be activated
  - Let us represent the choices for all edges in the entire network be x
- We showed that given x, the function is submodular
- Now let X be the space of possibilities of all such choices
  - Each element x in X contains choices for all edges
  - In making the random choices beforehand, we had basically fixed x
- Now, we can sum over all possible x, weighted by their probability.

- Now, we can sum over all possible x, weighted by their probability.
    - Since non-negative linear combinations of submodular functions are submodular, the sum is submodular
- The approximation algorithm for submodular maximization is an approximation for the cascade in independent activation model with same factor

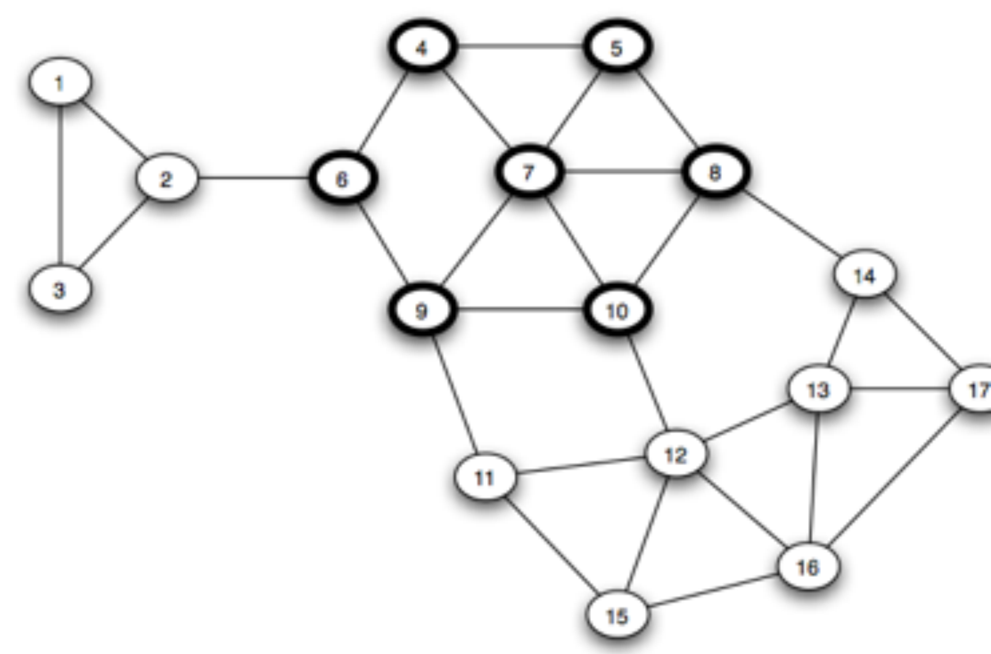
- The linear threshold model

- Node compares the fraction of its neighbors activated to a threshold q

- Generalization: Each edge has a weight $p_{u,v}$ and total weight for activated items must exceed q



(a) *Two nodes are the initial adopters*



(b) *The process ends after three steps*

- Modified model (for the proof):
- Node u picks 1 neighbor v and turns on directed edge vu (meaning v influences u)
  - Edge vu is turned on with probability proportional to $p_{u,v}$
  - All other edges are turned off (not used)

Theorem

- Any subset $H \subseteq V$ has the same probability of being covered in
  - Original linear threshold model, and
  - Modified model
- Proof: Omitted

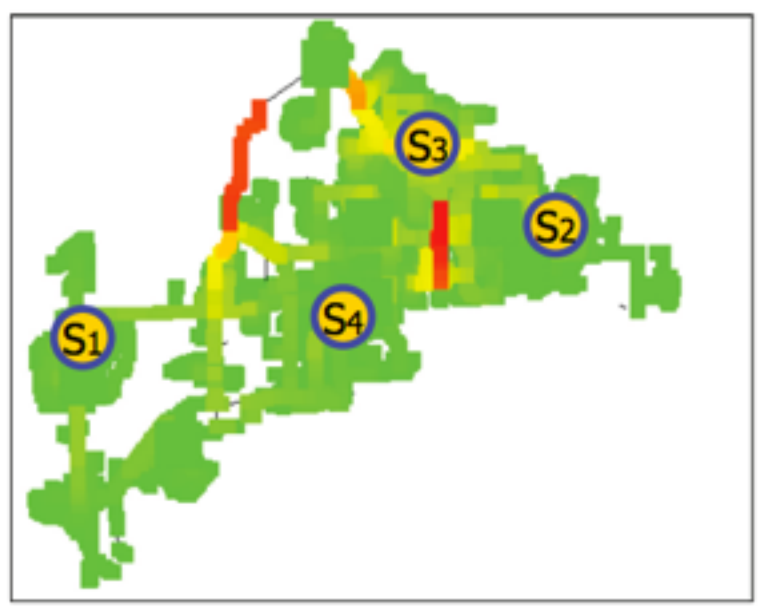
- Ref: Kempe, Kleinberg, Tardos; Maximizing the spread of infleunce through a social network, SIGKDD 03.

Applications of submodular optimization

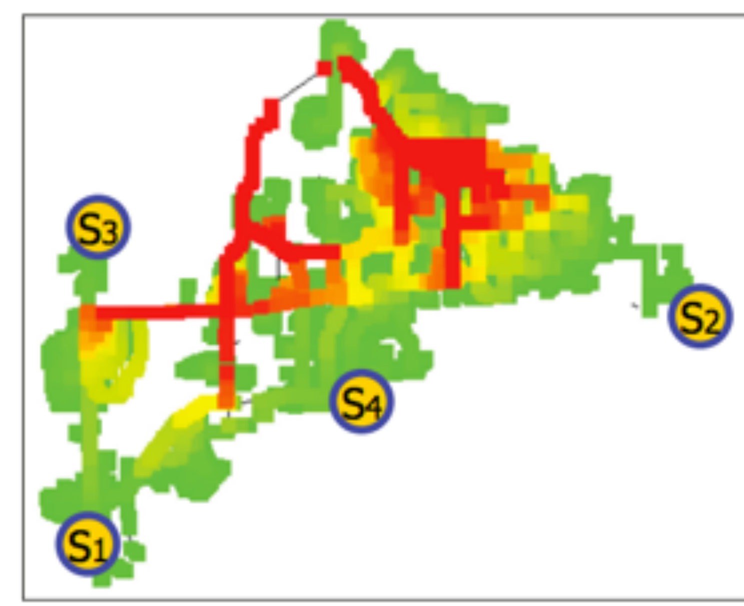
- Sensing the contagion
  - Place sensors to detect the spread
- Find “representative elements”: Which blogs cover all topics?
- Machine learning
  - Exemplar based clustering (eg: what are good seeds?)
  - Image segmentation

Sensing the contagion

- Consider a different problem:
- A water distribution system may get contaminated
- We want to place sensors such that contamination is detected



(c) effective placement



(d) poor placement

Social sensing

- Which blogs should I read? Which twitter accounts should I follow?
  - Catch big breaking stories early
- Detect cascades
  - Detect large cascades
  - Detect them early…
  - With few sensors
- Can be seen as submodular optimization problem:
  - Maximize the “quality” of sensing
- Ref: Krause, Guestrin; Submodularity and its application in optimized information gathering, TIST 2011

Representative elements

- Take a set of Big data
- Most of these may be redundant and not so useful
- What are some useful “representative elements”?
  - Good enough sample to understand the dataset
  - Cluster representatives
  - Representative images
  - Few blogs that cover main areas…

Problem with submodular maximization

- Too expensive!
- Each iteration costs O(n): have to check each element to find the best
- Problem in large datasets
- Mapreduce style distributed computation can help
  - Split data into multiple computers
  - Compute and merge back results: Works for many types of problems
- Ref: Mirzasoleiman, Karbasi, Sarkar, Krause; Distributed submodular maximization: Finding representative elements in massive data. NIPS 2013.

Projects

- Office hours
- Wednesday 11 nov (tomorrow), 10:00-12:00
- Monday 16 nov, 10:00 - 12:00

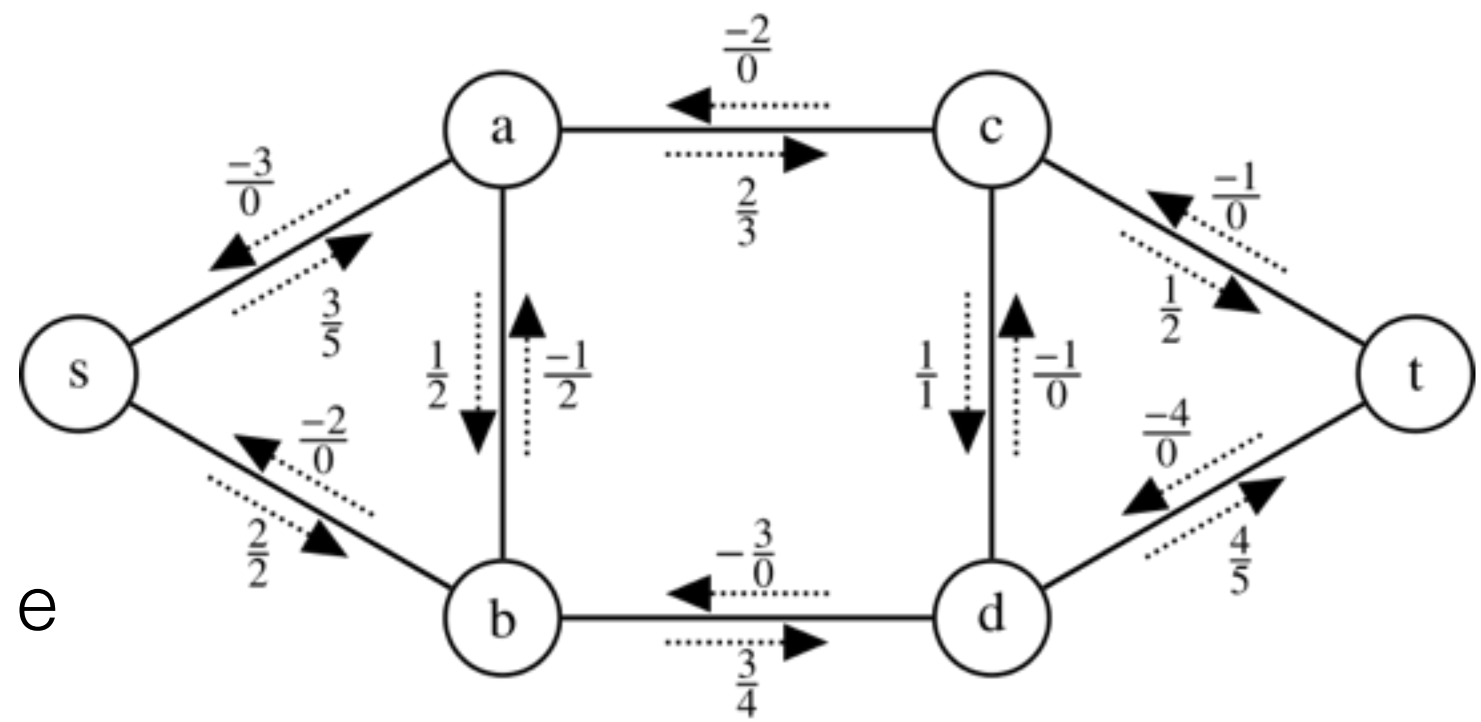
- Submission guidelines to be given today (I hope..)

PhD at Edinburgh

- If you are finding the project interesting…
- CDT in datascience:
  - http://datascience.inf.ed.ac.uk/
- CDT in parallelism/systems:
  - http://pervasiveparallelism.inf.ed.ac.uk/
- Other PhD options:
  - http://www.ed.ac.uk/informatics/postgraduate/research-degrees/phd
- For general procedure for applying, see a guideline at
  - http://homepages.inf.ed.ac.uk/rsarkar/positions.html
  - Ask any questions..

Network Flows and Cuts

- Network flow problem
- Give an graph (imagine pipes/ roads)
  - Nodes s, t
  - Capacity c(e) on each edge e
  - What is the maximum rate of flow from s to t ?
- Solution consists of a flow value on each edge that attains max flow from s to t

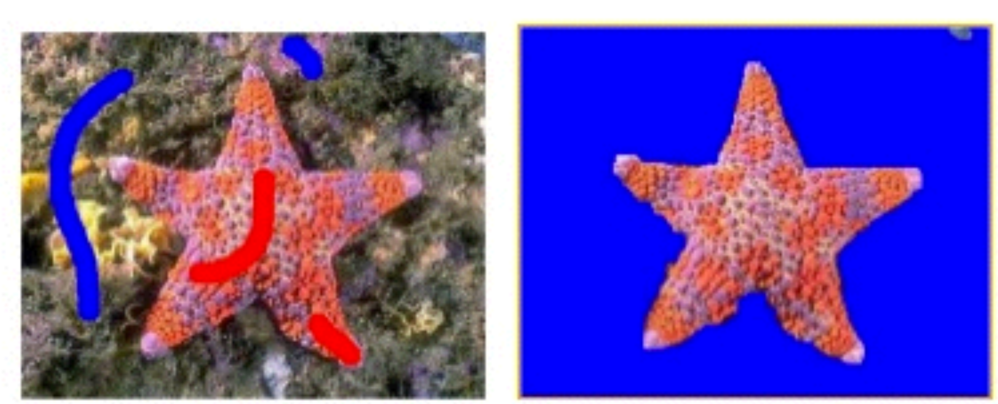


Network flows

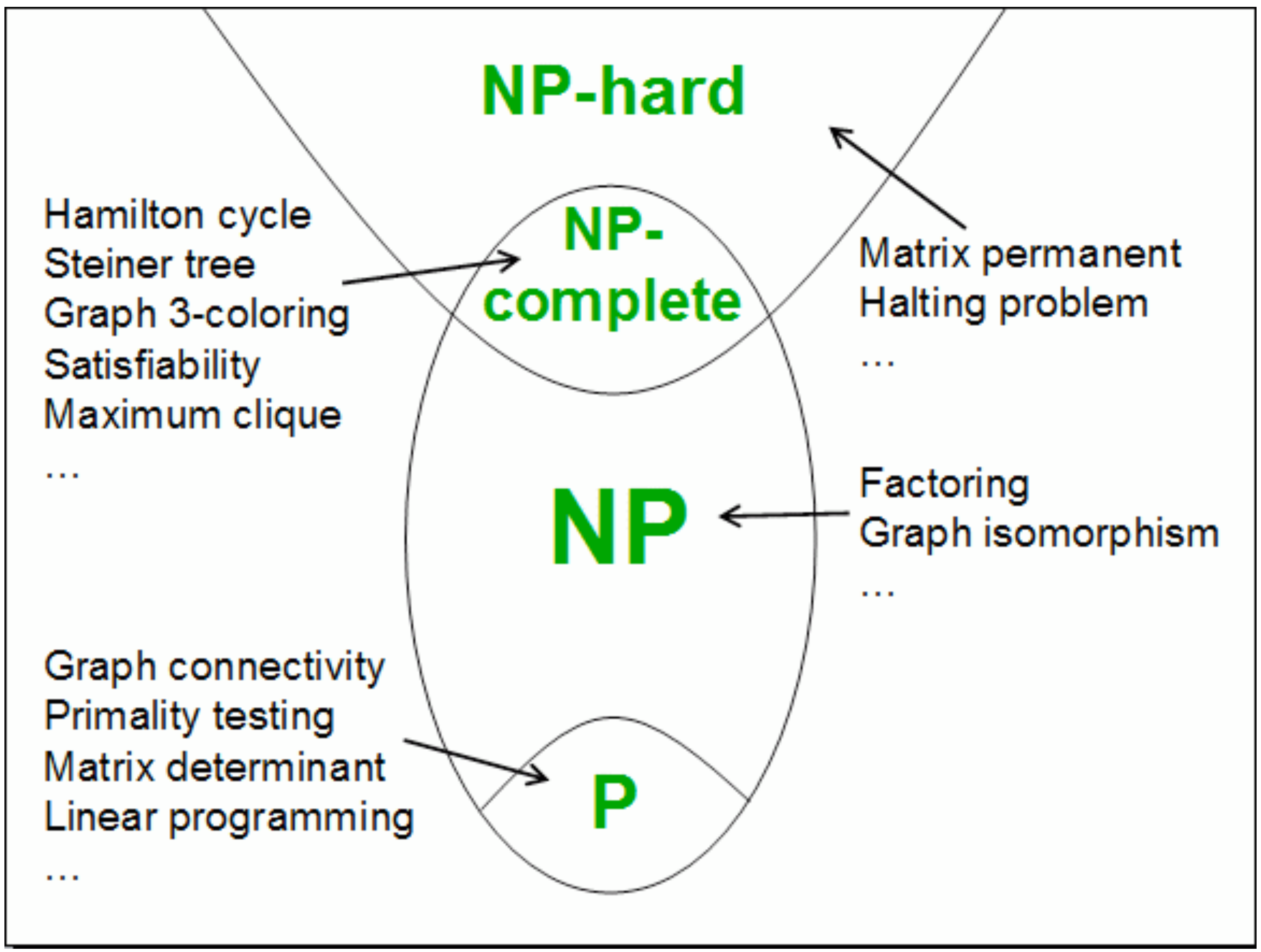
- Solved using Ford-Fulkerson or similar algorithms
- Complexity ~ O(nm) [ie. O(|V| * |E|)]
    - or similar, depending on exact requirements etc
    - Too large in large networks

Minimum cuts

- Find the set of edges with smallest capacity that separates s and t
    - Max flow min cut Theorem: The total capacity of this smallest cut is the max flow from s to t.
- The cut capacity function f: flow across a cut
    - Is submodular
- Min cut: Submodular minimization
    - Application: Image segmentation



Complexity classes P, NP, NP-hard



Class P

- Decision problems: A yes or no answer
- Problems that can be solved in polynomial time
- eg:
  - Searching: Does element x exist in array A?
  - Graph connectivity: Is G connected…

Class NP

- Some decision problems do not have known polynomial time solutions
- But given a “yes” answer, the solution can be checked in polynomial time
- Eg.
  - Vertex cover: Is there a subset S of size k in V such that every edge has at least one end point in S?
  - Does the graph contain a clique of size k ?
  - Set cover: Suppose X = {S1, S2, …} is a collection of subsets of U
    - is there are collection of size k that covers all elements of U?

Succinct certificates

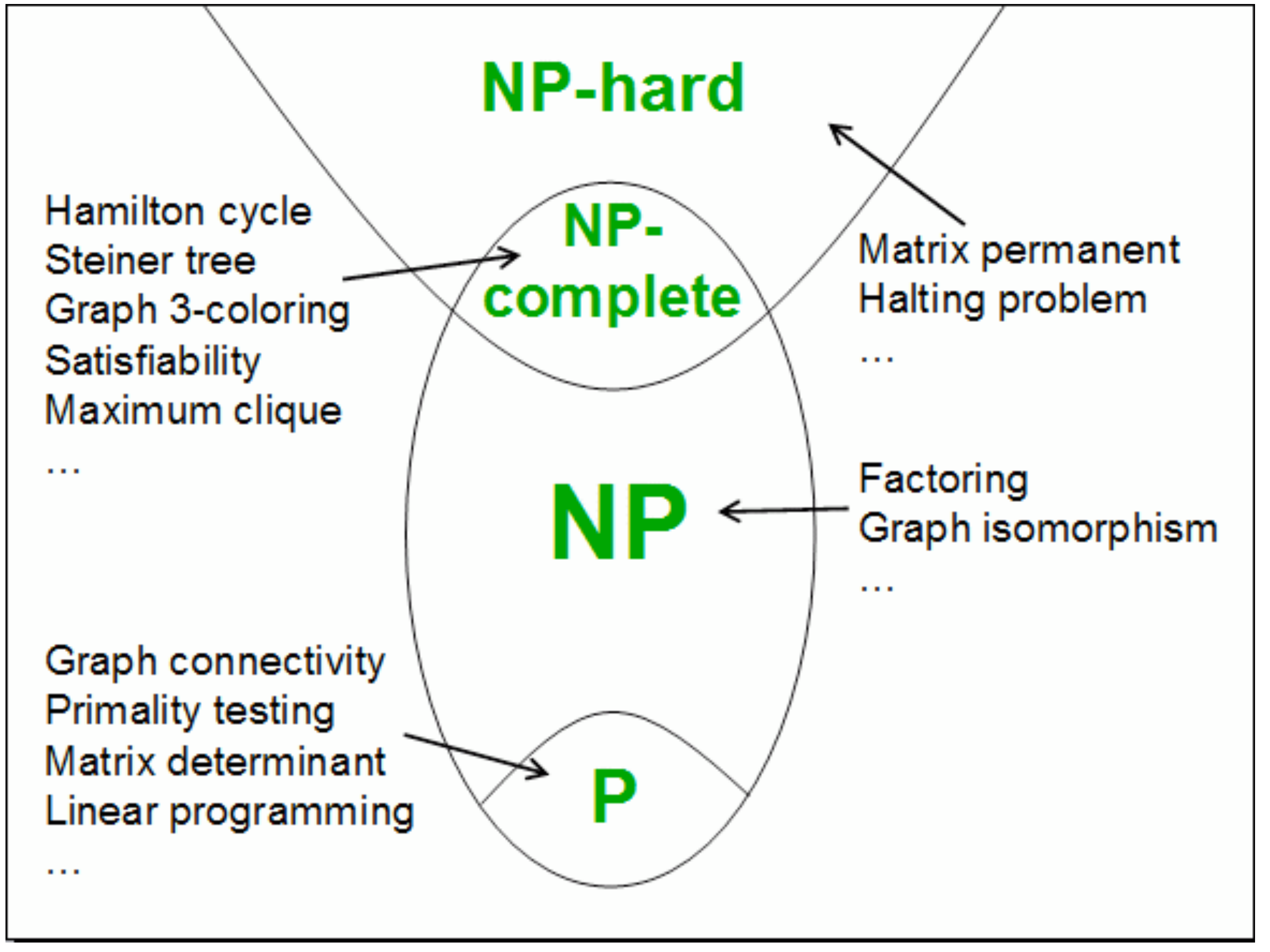
- NP problems have *succinct certificates* — that can be used to check the answer in polynomial time
- E.g.
  - Vertex cover: The solution set S of size k
  - Clique: The clique of size k
  - Set cover the collection of size k that covers V

Problem reduction

- Convert problem 1 to a version of problem 2
- E.g. Vertex cover to set cover
    - Elements U = E
    - Collection of subsets: S_v = Edges on vertex v
- U can be covered by a collection of size k iff E can be covered by a set Y in V
- Note:
    - If we have a solution to Set cover, we can use it to solve vertex cover
    - The conversion from problem 1 to problem 2 is polynomial time

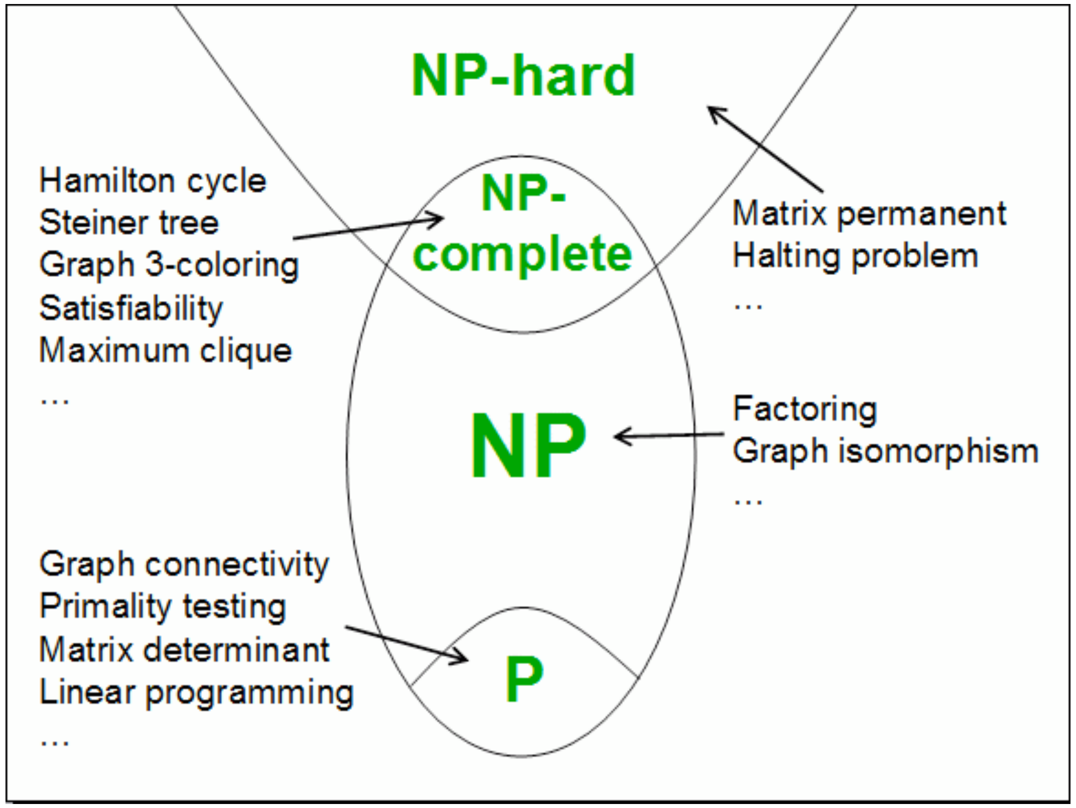
Classes NP-Hard and NP-complete

- A problem X is NP hard, if *any* NP problem can be reduced to X in polynomial time
- A problem is NP-complete if it is both:
  - In NP
  - and NP-hard



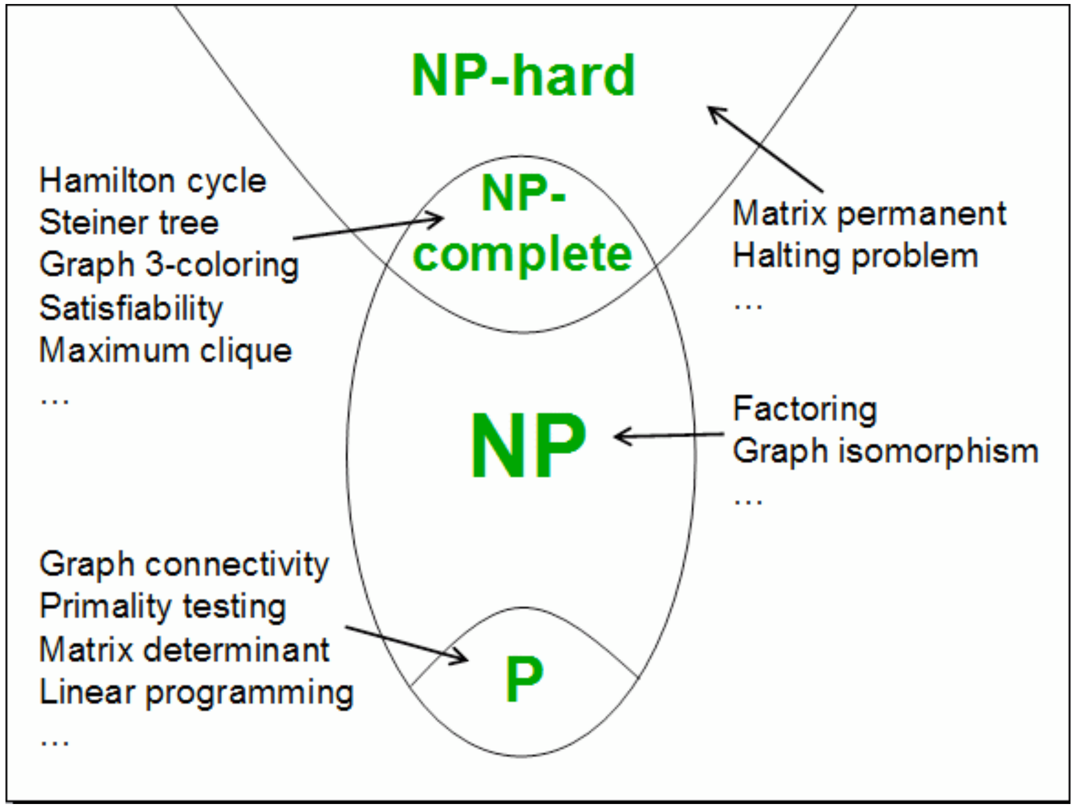
Showing that a problem X is NP-complete

- Show X is in NP
    - Usually easy: Show a succinct certificate
- Showing NP-hardness
- Idea: All NP-complete problems are reducible to each-other!
- So, show that one known NP-complete problem can be reduced to X



Showing that a problem X is NP-complete

- Take Y which is NP-complete
- Show that an instance of Y can be reduced to an instance of X in polynomial time
- And the solution of X can be converted back to a solution of Y in Polynomial time
- Thus, if X has an easy (Polynomial) solution, that can be used to solve NP-hard problem Y
- Implies that X cannot have easy (polynomial) solution!



NP-hardness

- Note that an NP-hard problem need not be a decision problem it can be an optimization problem
- E.g.
  - Find largest clique
  - Find smallest set cover
  - Find longest path…
- Proving the NP-hardness part is anyway the difficult issue