More Approximation Algorithms

What it means for an algorithm to be an α-approximation

Minimization Problems: **OPT** ≤ **ALG** ≤ α · **OPT**, **α > 1** (smaller α is better)

Maximization Problems: **OPT** ≥ **ALG** ≥ α · **OPT**, **α < 1** (larger α is better)

ALG = value returned by our algorithm

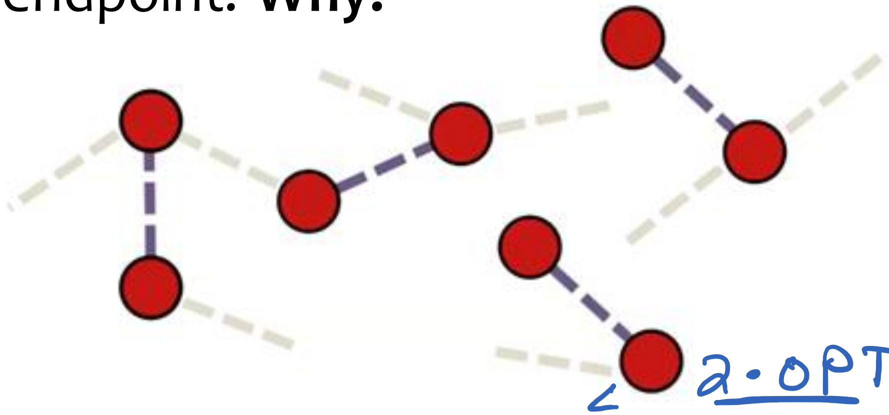
OPT = Optimal value

α is the *approximation ratio*

Approximating Minimum VC

A seemingly-terrible-but-actually-good idea: Choose an arbitrary edge, add *both* endpoints to the VC, and delete endpoints (and incident edges).

Observation about double-cover algorithm: None of the edges we choose share an endpoint. **Why?**



Observation: ALG $\overset{\le}{=}$ **2** · (#edges chosen) $\le 2 \cdot OPT$

Upper bound on ALG

Observation: OPT must circle at least one endpoint of each of our chosen edges. **Why?**

⇒ (# edges chosen) ≤ **OPT**

Lower bound on OPT

Two ingredients in approximation analysis

Minimization Problems:

- **Upper bound** on **ALG**
- **Lower bound** on **OPT**

Maximization Problems:

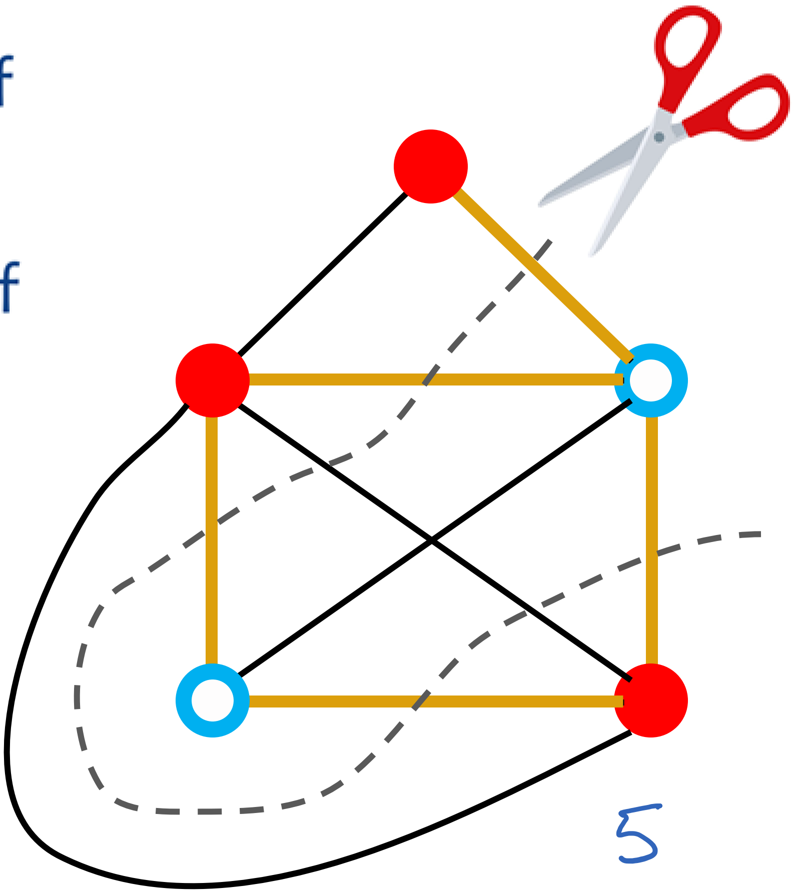
- **Lower bound** on **ALG**
- **Upper bound** on **OPT**

Written in terms of cleverly chosen 3rd quantity

Next: an approximation algorithm for the problem of Maximum Cut…

Graph Cuts

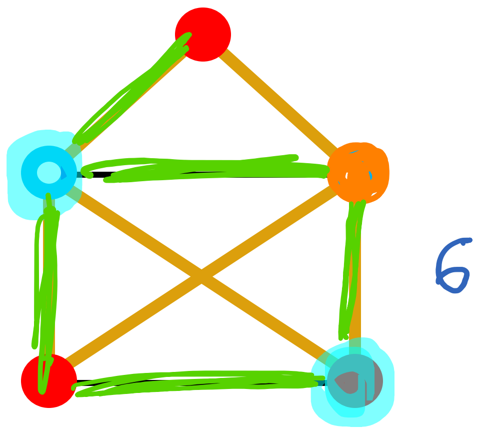
- A ***cut*** of a graph is a *partition* of its vertices $(S, \overline{S})$.
- An edge ***crosses*** the cut $(S, \overline{S})$ if one of its endpoints is in S and the other is in $\overline{S}$.
- The ***size*** of a cut $(S, \overline{S})$ is the number of edges crossing it.



Maximum Cut Problem

Max-Cut Problem: Given a graph, find a cut of maximum size.

- decision version is NP-complete (we won't prove)

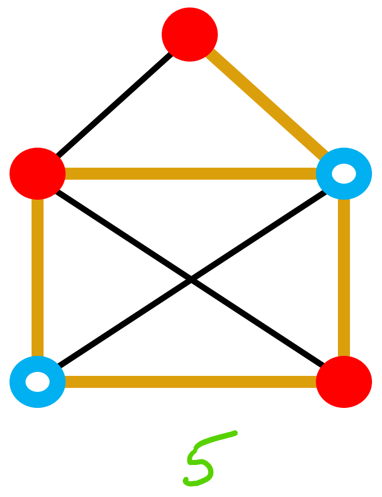


Has applications in network/circuit design, physics, and more…

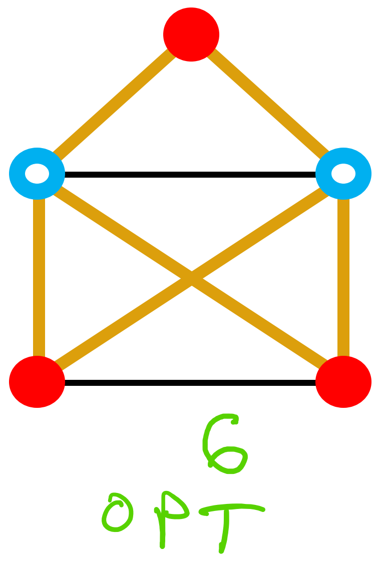
Approximate Maximum Cut

We will show a poly-time ½-approximation

(i.e. the cut returned by our algorithm is at least ½ the size of a max cut)



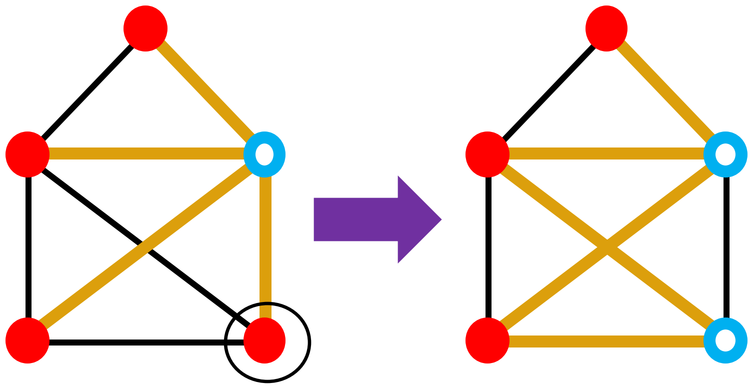
is a 1/2-approx. of optimum:



Approximate Maximum Cut

Technique: Local Search

Idea: Start with an arbitrary cut. Repeatedly pick a vertex, look at its neighborhood, decide whether to move it to the other side of the cut.



If we move a vertex to the other side of the cut, how does the size of the cut change?

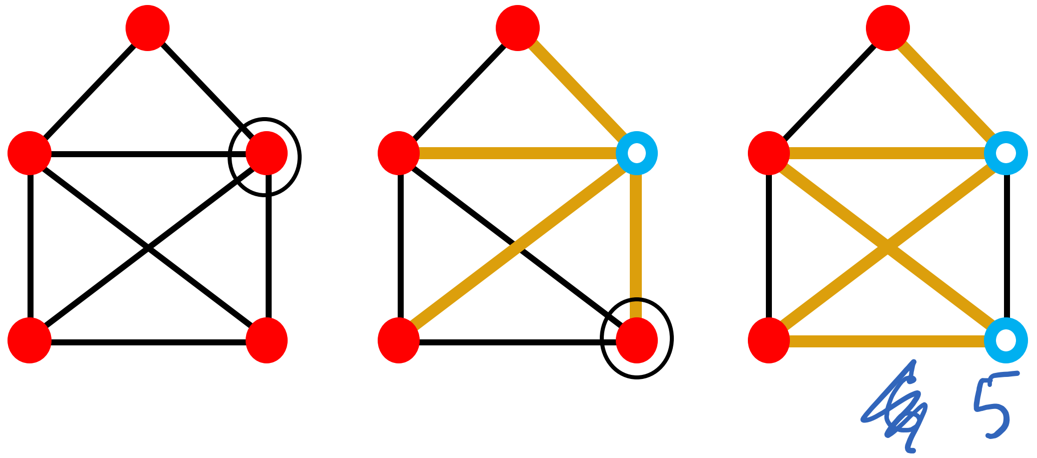
~~+ #neighbors on old side of cut − #neighbors on new side of cut~~

+ #neighbors on my side of cut (originally)
− #neighbors on other side of cut (originally)

Approximate Maximum Cut

Algorithm:

- Start with an arbitrary cut.
- Repeat: Find a vertex v such that the **majority of v's neighbors are on v's side of the cut**. Move v to the other side of the cut.
- If no such v exists, return.



As a result, the size of the cut increases!

Approximate Maximum Cut

Approximation Ratio Analysis:
When the algorithm terminates, at least half of the edges in the graph are in the cut. **Why?**

When alg terminates, every vertex has $\geq \frac{1}{2}$ of its incident edges in the cut.

$$\underline{\#\text{ edges in cut}} = \frac{\sum_v \overbrace{\#\text{ v's incident edges in cut}}^{\geq \frac{\text{degree}(v)}{2}}}{2} \geq \frac{\overbrace{\sum_v \text{degree}(v)}^{2m}}{4} = \frac{m}{2}$$

Thus, ALG ≥ m/2.

Lower bound on ALG

OPT ≤ m.

Upper bound on OPT

⇒ ALG ≥ ½ · OPT

Approximate Maximum Cut

Running Time Analysis:

Why does the algorithm terminate?

At every step, cut size increases by ≥ 1. Can't exceed m.

Potential function: $m -$ size cut

Why is it polynomial time?

$$O\Big(\underset{\substack{\uparrow \\ \text{\# iterations}}}{m} \cdot \big(m + \underset{\substack{\uparrow \\ \text{time per} \\ \text{iteration}}}{n}\big)\Big)$$

Can we do better than 1/2?

Yes!

A breakthrough by Goemans and Williamson gave a 0.878..-approximation.

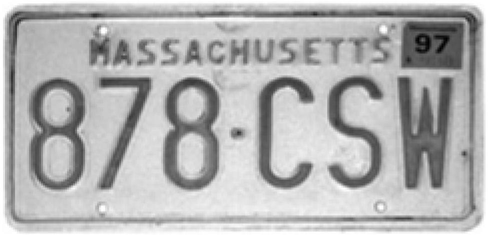
It is NP-hard to do any better under a complexity hypothesis (Unique Games Conjecture).

Mathematical Vanity Plates

DONALD E. KNUTH

This column is a place for those bits of contagious mathematics that travel from person to person in the community, because they are so elegant, surprising, or appealing that one has an urge to pass them on.

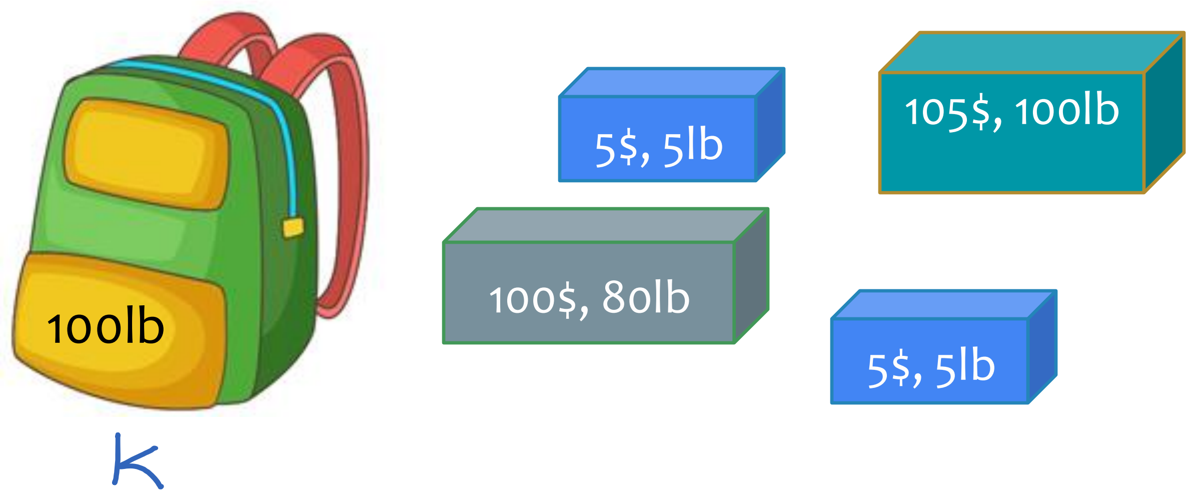
Sometimes people obtain mathematically significant license plates purely by accident, without making a personal selection. A striking example of this phenomenon is the case of Michel Goemans, who received the following innocuous-looking plate from the Massachusetts Registry of Motor Vehicles when he and his wife purchased a Subaru at the beginning of September 1993:



Two weeks later, Michel got together with his former student David Williamson, and they suddenly realized how to solve a problem that they had been working on for some years: to get good approximations for maximum cut and satisfiability problems by exploiting semidefinite programming. Lo and behold, their new method—which led to a famous, award-winning paper [15]—yielded the approximation factor .878! There it was, right on the license

Knapsack Problem

Given a backpack with weight capacity ~~W~~ K, and a set of **n** items each with an integer value $\mathbf{v_i} \leq \mathbf{V}$ and weight $\mathbf{w_i} \leq$ ~~W~~ K, what is the largest total value of a set of items that fit in the backpack (i.e. total weight of set $\leq$ ~~W~~)?
K

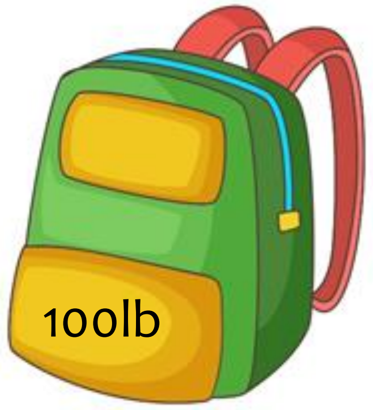


On the HW: **Knapsack is NP-hard**

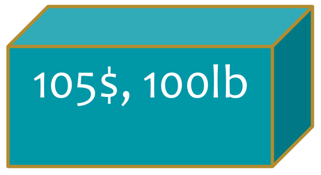
Approximate Knapsack

We will show a poly-time ½-approximation
i.e. the total value of the items chosen by our algorithm is at least ½ the optimal value.

(we will start it in class and you will finish it for HW)

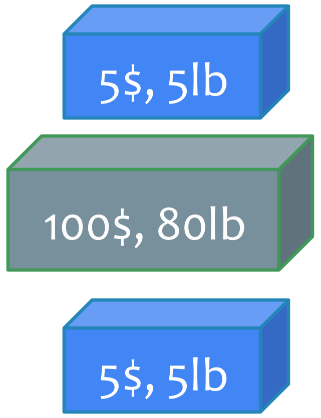


~0.95 · OPT



105

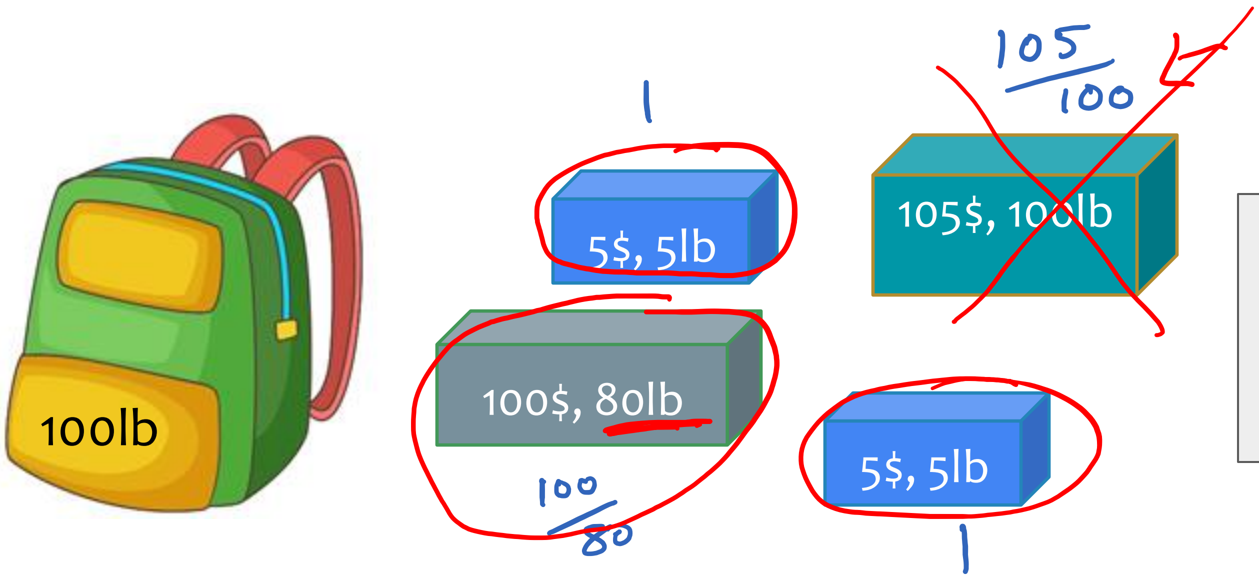
OPT



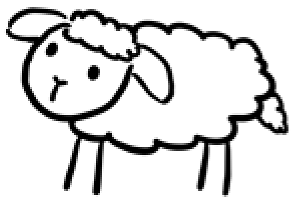
110

Relatively-Greedy Algorithm:

- Consider items in decreasing order by *relative* value (breaking ties arbitrarily) i.e. the ratio **value/weight**
- Greedily select item if it fits in remaining capacity.



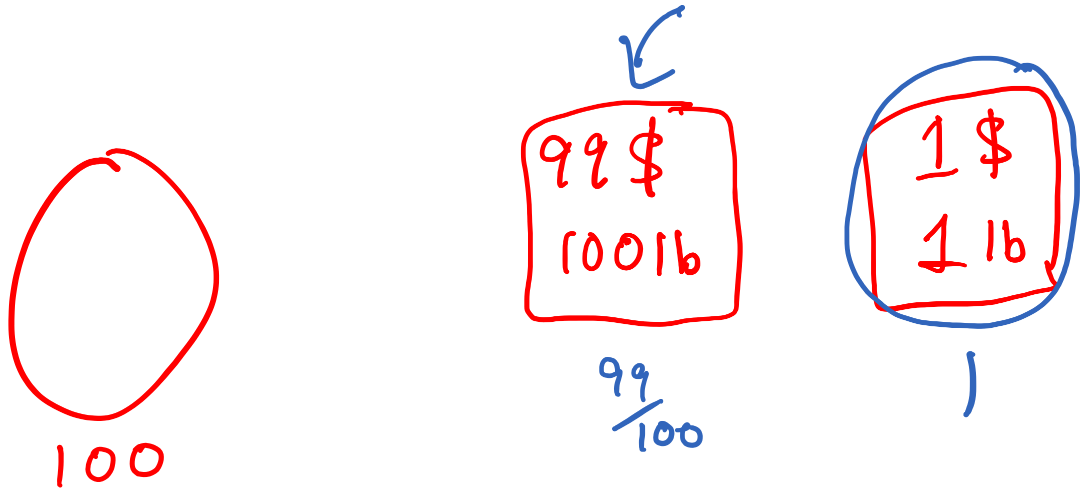
This is similar to the algorithm from the HW5 that solves the *fractional* knapsack problem



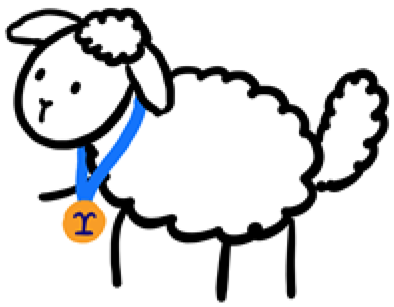
Your task: How bad is the approximation ratio of the Relatively-Greedy algorithm?

Construct an example to support your claim.

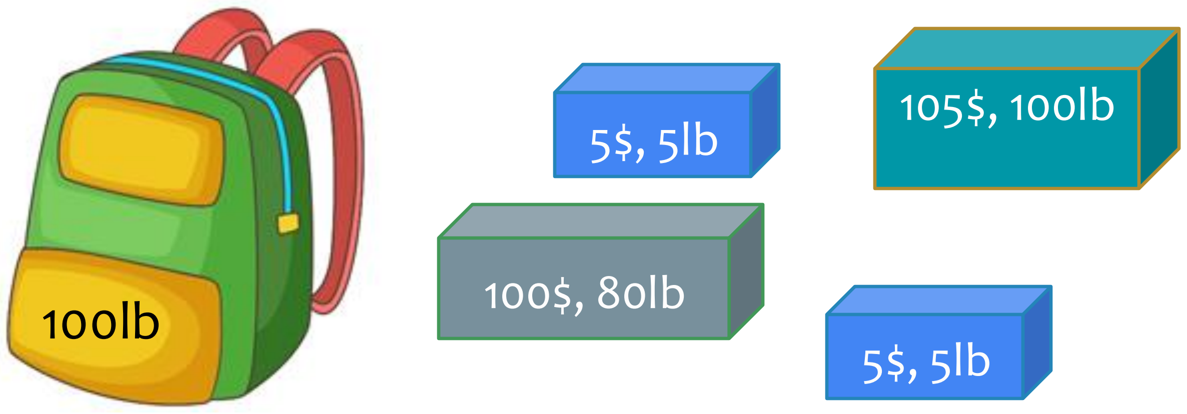
(An example consists of: weight of backpack, and weight/value of each item)



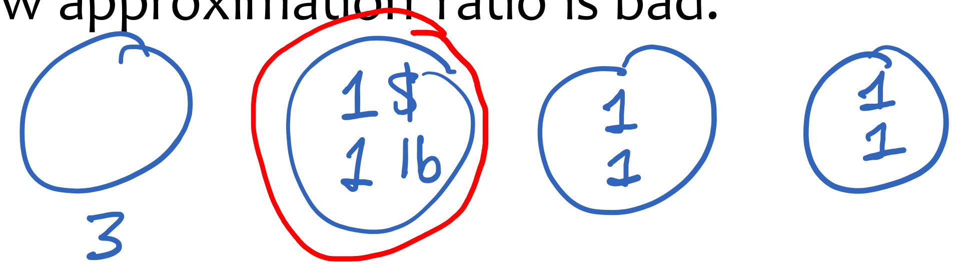
Just take the one single item of largest value. Done!



Single-Greedy Algorithm: Take the one single item of largest value that fits in the backpack. (Don't take any more items.)



Example to show approximation ratio is bad:



Combined-Greedy Algorithm:

- Run **Relatively-Greedy** and **Single-Greedy**
- Take the best of the two solutions

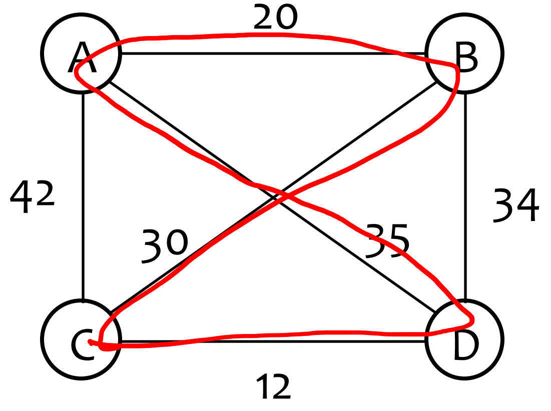
On the HW you will show: **Combined-Greedy** is a ½-approximation!

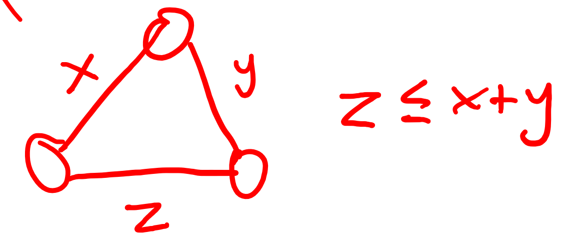
Example of combined algorithms in practice: ***The Netflix Challenge* (2009)**
"The most interesting aspect is how the 'competition' turned into a cooperation"
"[The winning team] simply ran hundreds of algorithms from their 30-plus members and combined their results into a single set, using a variation of weighted averaging that favored the more accurate algorithms."

Approximate Metric-TSP

Input: n vertices, positive distances between each pair of vertices that obey the **triangle inequality**

Output: What is the minimum total length of a cycle ("tour") containing every vertex?





$z \leq x+y$

The decision version of Metric-TSP is NP-complete.

We will show a poly-time 2-approximation.

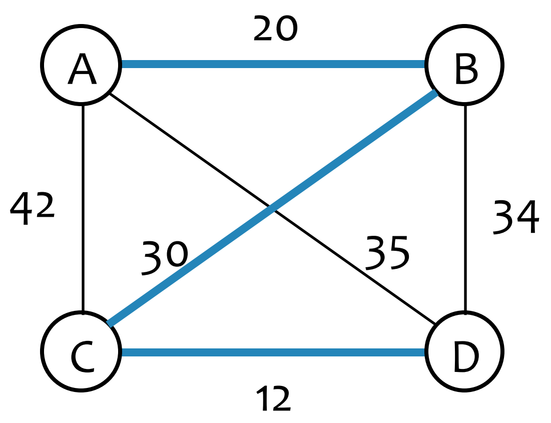
(returns a tour of length at most 2 times the optimal tour)

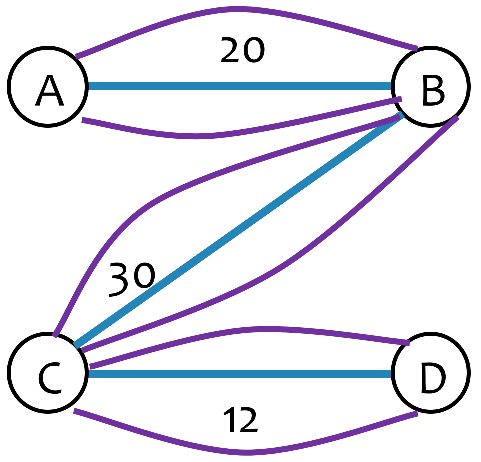
Approximate Metric-TSP

Step 1: Find an **MST** (in polynomial time)

Step 2: Walk around the perimeter of the **MST** to form **“tree-tour”**

tree-tour is not a legitimate TSP tour!





tree-tour = **2·MST**

Approximate Metric-TSP

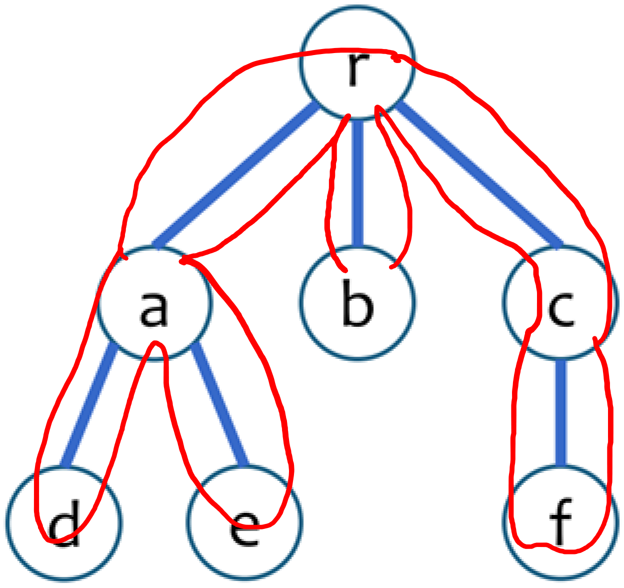
Step 1: Find an **MST** (in polynomial time)

Step 2: Walk around the perimeter of the **MST** to form **"tree-tour"**

tree-tour is not a legitimate TSP tour!

If you wanted to code it:

```
Find-Tour(u)
    Let $v_1, \ldots, v_k$ be u's children
    For i = 1, ..., k
        $T = T + (u, v_i)$
        Find-Tour($v_i$)
        $T = T + (v_i, u)$
```



tree-tour = **2·MST**

Approximate Metric-TSP

Step 1: Find an **MST** (in polynomial time)

Step 2: Walk around the perimeter of the **MST** to form **"tree-tour"**

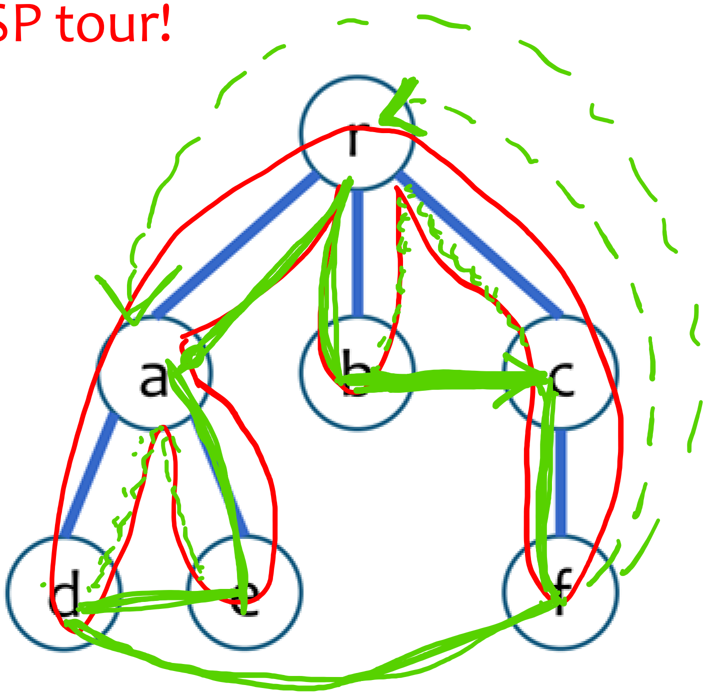
tree-tour is not a legitimate TSP tour!

Step 3: **"Shortcut"** tree-tour

Now we need a lower bound on OPT!

Claim: MST ≤ OPT. Why?

MST ≤ spanning tree in OPT cycle ≤ OPT



ALG ≤ tree-tour = 2·MST ≤ 2·OPT

Can we do better than a 2-approximation?

Yes!

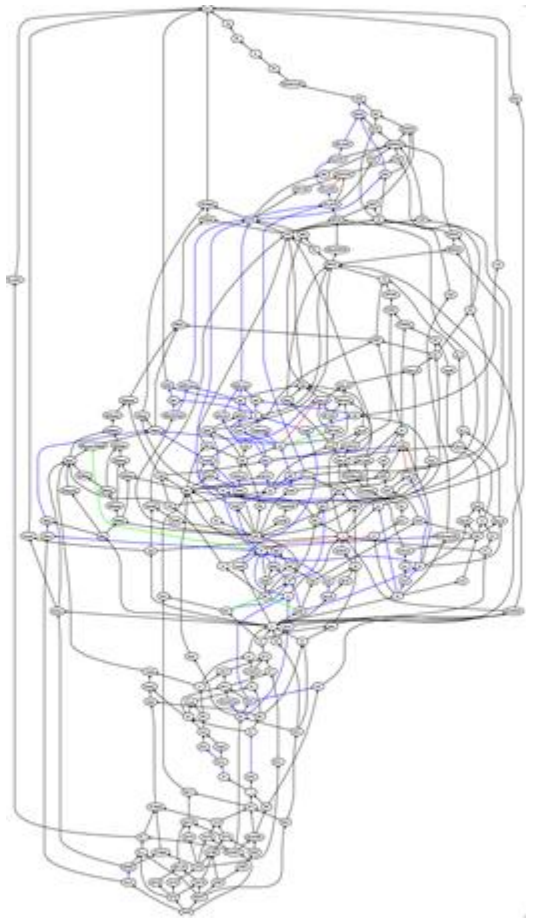
- [Christofides 1976] 1.5-approximation
- [Karlin-Klein-Oveis Gharan 2021] $(1.5 - 10^{-36})$-approximation.
- [Karpinski-Lampis-Schmied 2013] No 1.008-approximation unless P = NP.

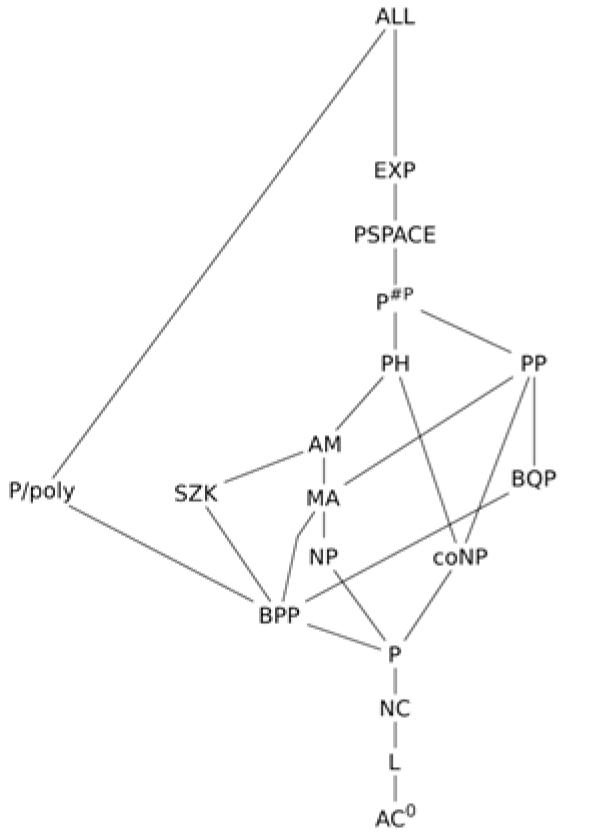
Ways to deal with NP-Hardness

1. **Approximation algorithms**
2. Restrict to **special classes of inputs**
   - E.g. randomly-generated inputs, planar graphs, …
3. **Heuristics:** algorithms without provable guarantees that seem to work well in practice
   - SAT solvers sometimes do well in practice
4. If your **input is small**, sometimes you can afford to run an exponential-time algorithm

Goodbye Complexity…

The Complexity Zoo
(a database of 547 complexity classes)





The Petting Zoo

Scott Aaronson, zookeeper

Goodbye Complexity…

EECS 574: Computational Complexity

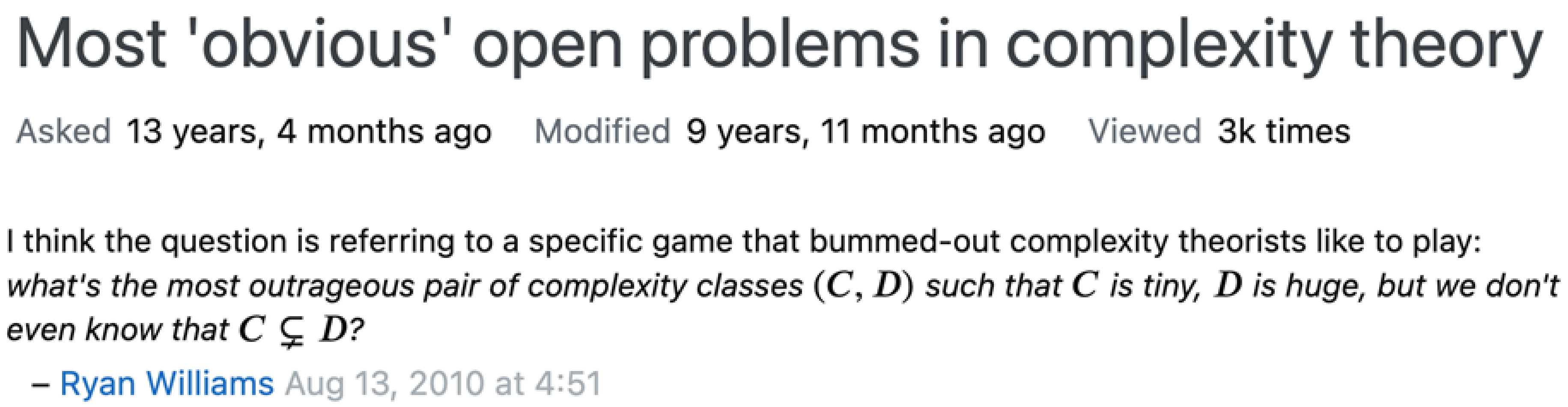
- more complexity classes
- hardness of approximation

EECS 477: Introduction to Algorithms

- more NP-hardness proofs
- more approximation algorithms

Open problems: Nearly everything

mathoverflow



Most 'obvious' open problems in complexity theory

Asked 13 years, 4 months ago Modified 9 years, 11 months ago Viewed 3k times

I think the question is referring to a specific game that bummed-out complexity theorists like to play: *what's the most outrageous pair of complexity classes* (C, D) *such that* C *is tiny,* D *is huge, but we don't even know that* $C \subsetneq D$*?*

– Ryan Williams Aug 13, 2010 at 4:51