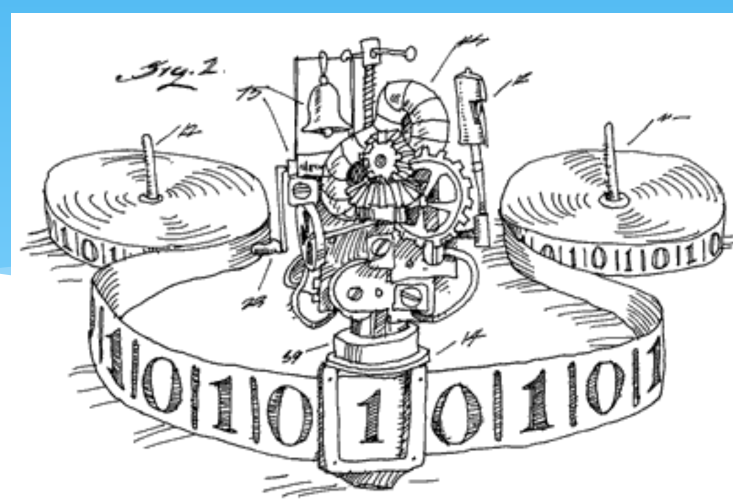
EECS 376: Foundations of Computer Science

Chris Peikert

22 March 2023

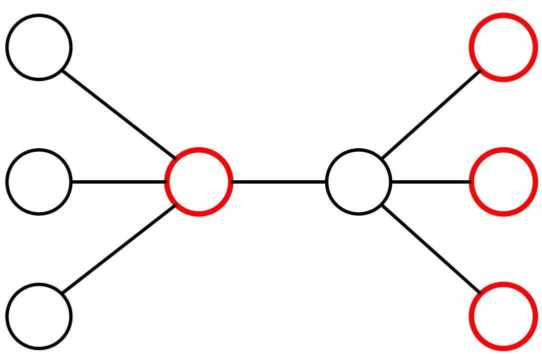


Agenda

- Approximation algorithms for NP-hard problems
  - Cute, clever, surprising!
- Analysis strategy: bound value of optimum solution

Approximating Min Vertex-Cover

* **Starbucks Executive:** "I'm ok with building *at most twice* as many stores as is optimal."
* A vertex cover S is an ***α-approximation*** if S contains *at most* **α** times as many vertices as a smallest one: $|S| \leq \alpha \cdot |C|$ for any VC C.
  * α is called the ***approximation ratio*** (smaller is better here)



A 2-approximate min-VC
(optimum = 2)

Attempt #3: Double Cover

* **Weird Idea:** Choose *edges* and delete *both* endpoints!

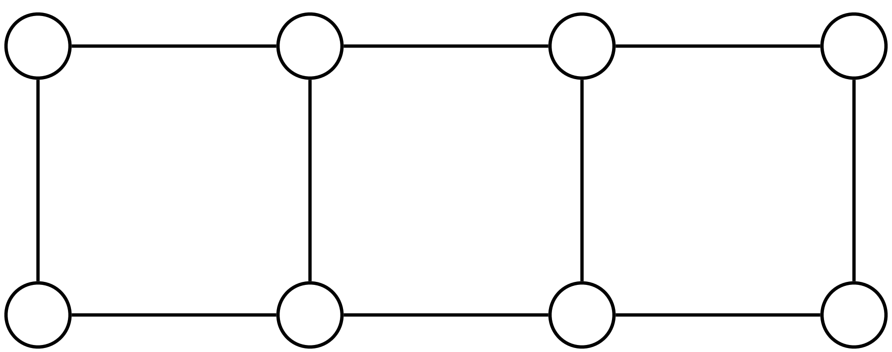
double-cover(G):
1. $C \leftarrow \emptyset$
2. **while** G has an edge:
3. choose any edge $e = (u, v)$
4. $G \leftarrow G - \{u, v\}$; $C \leftarrow C \cup \{u, v\}$ // delete/add **both** endpoints
5. **return** C

Theorem: double-cover obtains a 2-approx to min-vertex-cover.

Example and Key Fact

double-cover(G):
1. $C \leftarrow \emptyset$
2. **while** G has an edge:
3. choose any edge $e = (u, v)$
4. $G \leftarrow G - \{u, v\}$; $C \leftarrow C \cup \{u, v\}$ // delete/add both endpoints
5. **return** C

- **Key Fact:** chosen edges are (vertex-)*disjoint*; output cover has 2 • **(# chosen edges)** vertices.
- **Q:** How many vertices are needed to cover a set of *disjoint* edges?
- **Observe:** *Any* cover C^* has *at least* **(# chosen edges)** vertices.



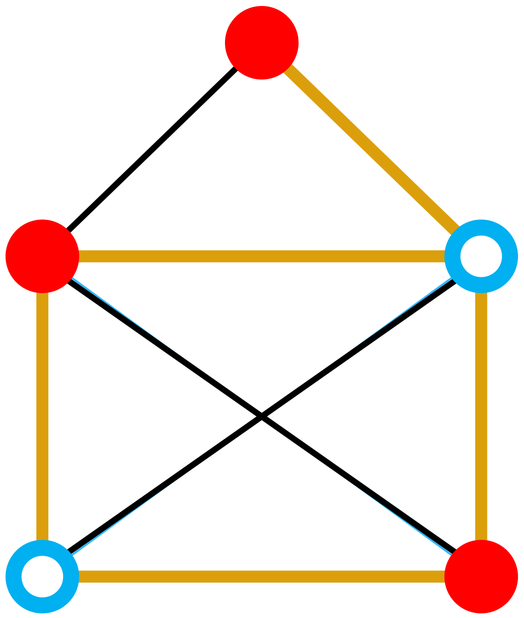
Proof of 2-Approx

double-cover(G):
1. $C \leftarrow \varnothing$
2. **while** G has an edge:
3. choose any edge $e = (u, v)$
4. $G \leftarrow G - \{u, v\}$; $C \leftarrow C \cup \{u, v\}$ // delete/add both endpoints
5. **return** C

* **Theorem:** **double-cover** outputs a 2-approx of min-vertex-cover.
  * Let M be the set of chosen edges and C be the set of vertices of M (i.e., output cover): $|C| = 2|M|$.
  * Consider any *arbitrary* vertex cover C^*.
  * Since M is disjoint and C^* covers it, $|M| \leq |C^*|$.
  * Therefore, $|C| = 2|M| \leq 2|C^*|$.
* **Observe:** we **lower-bounded** the size of any cover by the number of selected edges

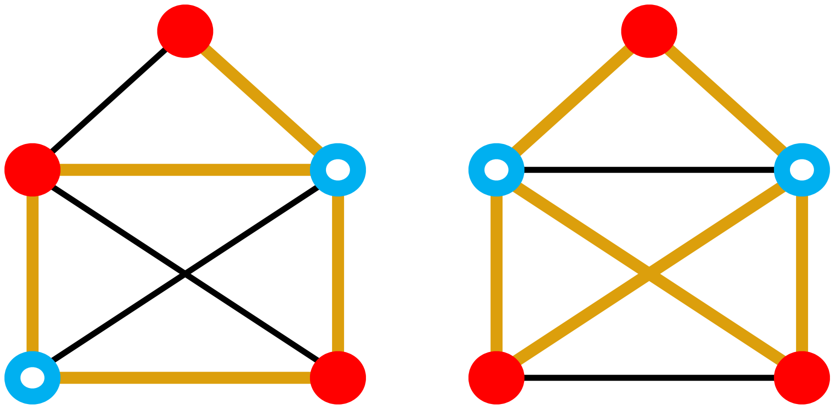
Cuts

- A ***cut*** of a graph is a *partition* $(S, \overline{S})$ of its vertices.
- 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.



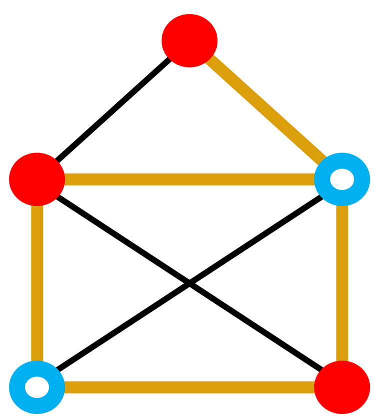
Max-Cut Problem

- **Problem:** Given a graph G, find a cut of G with the *largest* possible size, i.e., a ***max-cut***.
- **Fact:** The max-cut problem is $\mathbf{NP}$-Hard.
- **Applications:** network/circuit design, physics, …

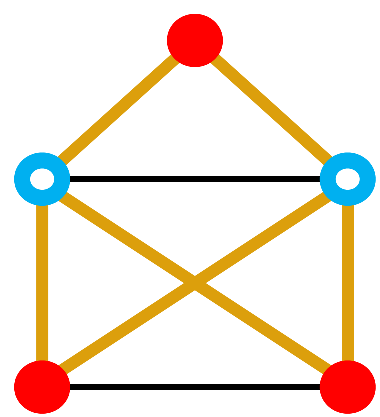


Approximate Max-Cut

- A cut of a graph G is an ***α-approximation*** $(\alpha \leq 1)$ of a max-cut if its size is *at least* α times the size of any (optimal) cut of G.
  - α is the ***approximation ratio*** (larger is better here)

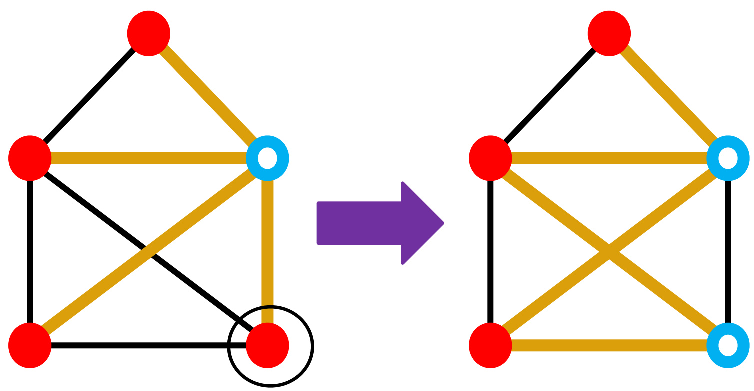


is a $\frac{5}{6}$-approx. of optimum:



Local-Move Heuristic

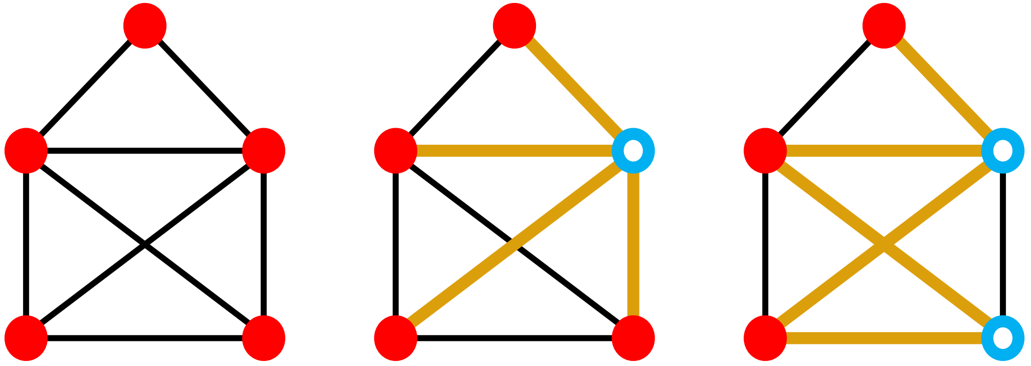
- **Q:** What happens to the cut size if we *flip* the color of the circled vertex in the example below?
- **Q:** Are there other vertices like this?



Observation: If we flip the color of a vertex with *majority same-color* neighbors, the cut size *increases*.

Local Search: Algorithm

- Initially color each vertex red ($S = \emptyset$, $\overline{S} = V$)
- **Repeat:** find a vertex v with *majority same-color* neighbors and *flip* its color.
  (I.e., if v is red, $S \leftarrow S \cup \{v\}$, $\overline{S} \leftarrow \overline{S} - \{v\}$;
  else, $S \leftarrow S - \{v\}$, $\overline{S} \leftarrow \overline{S} \cup \{v\}$.)
  - If none found, return current cut



EECS ECE CSE

Local Search: Efficiency

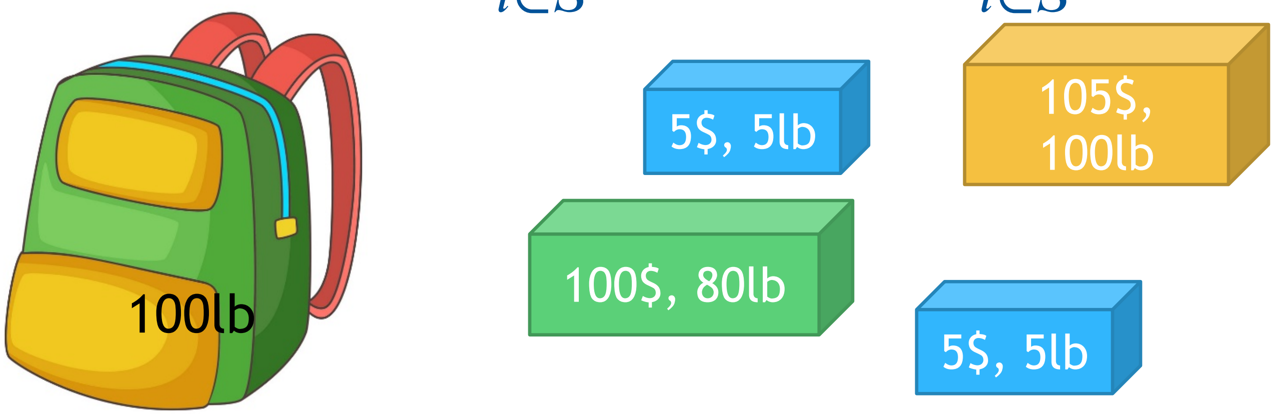
- Initially color each vertex red ($S = \emptyset$, $\overline{S} = V$)
- **Repeat:** find a vertex v with *majority same-color* neighbors and *flip* its color.
  - If none found, return current cut
- **Claim:** The algorithm is efficient.
- **Q:** How many flips can occur?
  - At most $|E|$.
  - **Potential** argument (blast from the past!): each flip increases the cut size, which cannot exceed $|E|$.

Local Search: Approximation

- Initially color each vertex red ($S = \emptyset$, $\overline{S} = V$)
- **Repeat:** find a vertex v with *majority same-color* neighbors and *flip* its color.
  - If none found, return current cut
- **Claim:** a ½-approximation of a max-cut is output.
- **Q:** How many *same-color* neighbors can each vertex end up with?
    - At least *half* the edges touching each vertex cross the cut, so the total number of cut edges is at least $\frac{1}{2}|E|$.
  - No cut can have more than $|E|$ edges, so the algorithm produces a ½-approximation of a max cut.

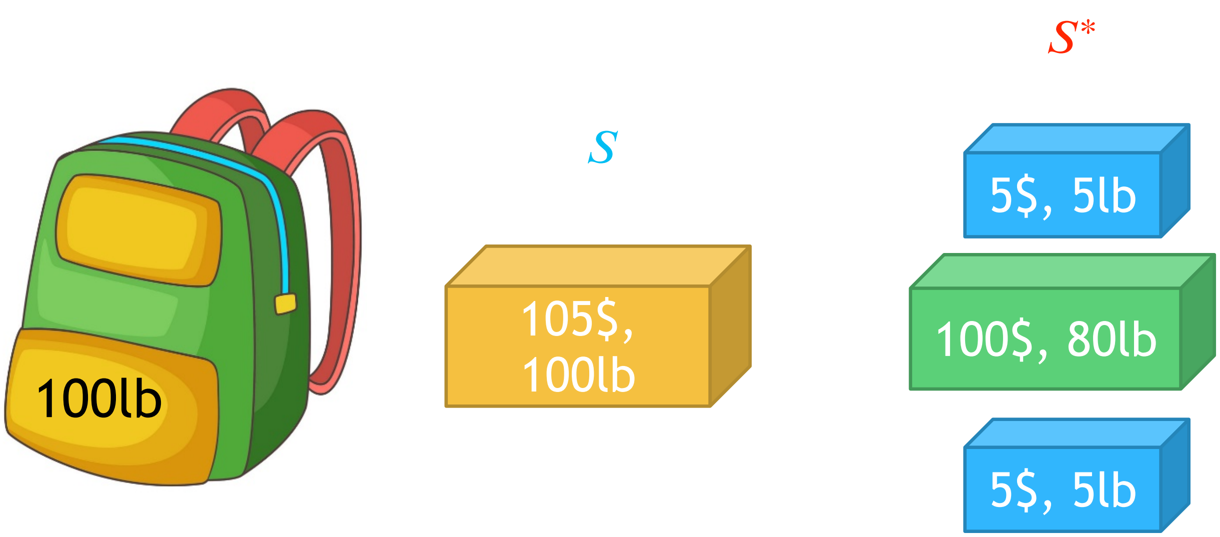
Knapsack

* n items; item i is worth $\$v_i$ and weighs w_i lbs
* Your knapsack can hold at most W lbs.
* **Problem:** Find a subset S of items having maximum value $\sum_{i \in S} v_i$ such that $\sum_{i \in S} w_i \leq W$.



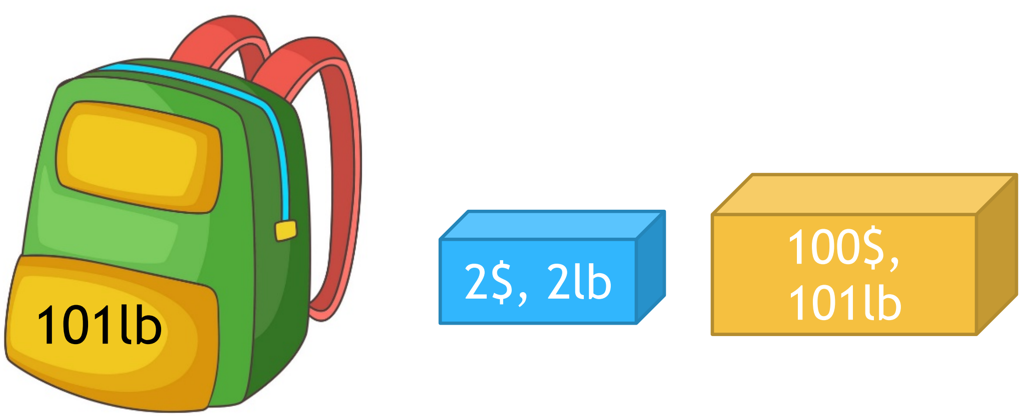
Approximate Knapsack

- Exact knapsack is NP-hard.
- A set of items is an ***α-approximation*** ($\alpha \leq 1$) if its value is at least α times that of an optimal set.



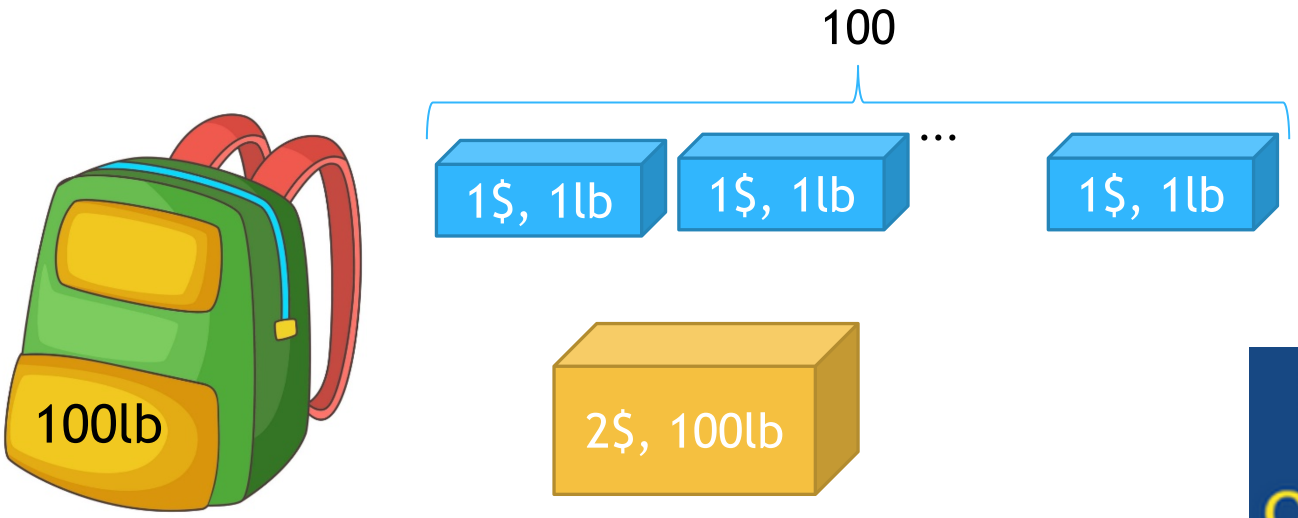
Relatively Greedy Algorithm

- **Approach I: Relatively-Greedy Algorithm**
- Consider items in non-increasing order by ***relative* value** v_i/w_i :
  - Greedily select the item *if* it fits w/in remaining capacity.
- **Example:** What's the approximation ratio here?



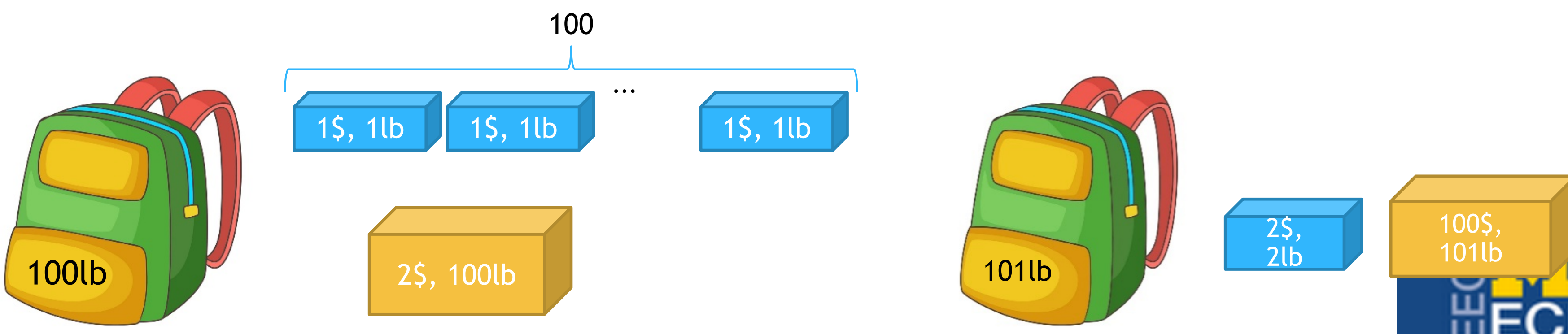
Dumb Greedy Algorithm

- **Approach II: Dumb-Greedy Algorithm**
- Take a ***single*** item of largest value v_i
- **Example:** What's the approximation ratio here?



Smart-Greedy Algorithm

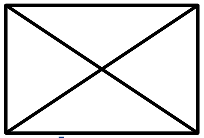
- **Approach III: Smart-Greedy Algorithm**
- Run **Relatively-Greedy** and **Dumb-Greedy**
- Take the best of the two solutions
- **Homework: Smart-Greedy** ½-approximates knapsack!

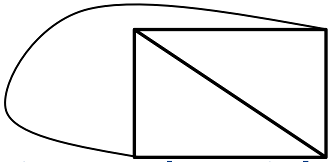


Smart-Greedy is optimal on these two instances!

More Ways to Cope (w/NP-hardness)

- **Idea:** Concentrate on an "interesting" subset of inputs.
- A graphs is ***planar*** if it can be drawn on the plane in such a way that no two edges cross each other.
  - **Example:**

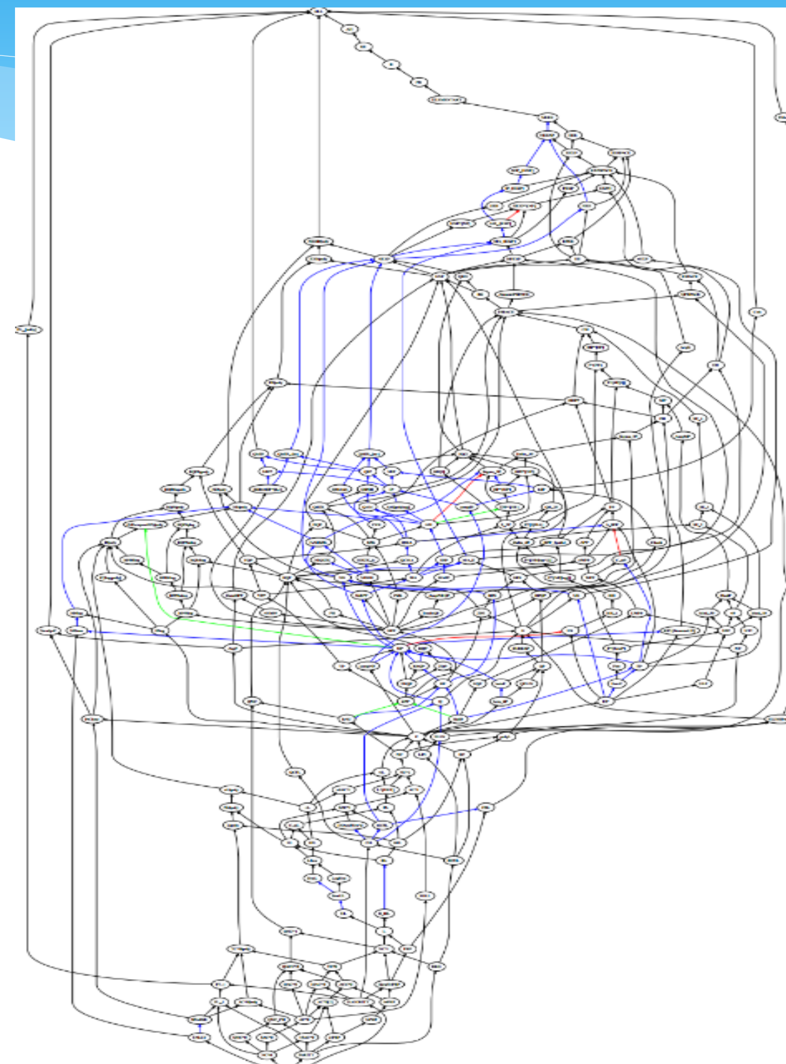




- **Fact:** Max-Cut has an efficient algorithm for planar graphs.
  - However, no efficient algorithm is known for vertex-cover on such graphs!
- **Fact:** Knapsack has an efficient (dynamic programming) algorithm for instances with "small" numbers.

Goodbye Complexity…

- It's a jungle out there
  (495 complexity classes and counting)
  - See "Complexity Zoo"
- **EECS 574**
- **Open problems:**
  - Nearly everything
  - We prove things like:
    "If pigs can fly, then horses can whistle."



Next Up: Randomness

- Next we will begin studying ***randomized algorithms***.
  - Often simple and efficient, but analysis can be tricky.
- It is possible that randomization yields <u>*strictly*</u> faster algorithms than any deterministic ones.
- Most experts *believe* that any efficient randomized algorithm with one-sided error can be "<u>*derandomized*</u>" to an efficient deterministic algorithm (w/ worse running time).
  - **Example:** primality testing, max-cut, max 3SAT