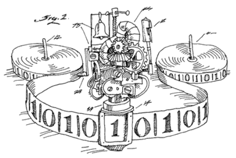
EECS 376: Foundations of Computer Science

Lecture 18 - Search to Decision and Dealing with NP-Completeness



NP-Completeness Retrospective

Skills learned:

- Recognizing provably "hard" problems (save time by not trying to find a fast algorithm)
- Converting a problem into a different problem (useful not only for hardness proofs, but also algorithm design)

Search-to-decision Reductions

TODAY

See lecture 14: *Intro to Complexity* (Used binary search)

Search Problems — Find a max clique

vs.

"Size" Problems — What is the size of a max clique?

vs.

Decision Problems — Is there a clique of size ≥k?

For all NP-complete problems,
if the decision version is in time T(n),
then the search version is in poly(T(n)) time.
(we won't prove, but we'll see examples)

Goal:

- Given an algorithm **size-clique** that returns the **size** of a max clique **in time T(n),**
- Show an algorithm **find-clique** that returns a max clique in **poly(T(n)) time.**

Common Strategy: Go through each vertex and consider whether removing it changes the size of the solution.

Idea of **find-clique(G)**:

1. Call **size-clique(G)**
2. Pick an arbitrary vertex **v** and remove it (and its incident edges) to get G-v.
3. Call **size-clique(G-v)**
   a. If the answer stayed the same:
      There exists a max clique without v ⇒ **don't include v** in our clique
   a. If the answer decreased by 1:
      Every max clique contains v ⇒ **include v** in our clique

Running time: $O(n \cdot T(n+m)) = poly(T(n))$

Goal:

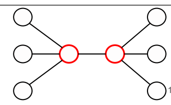
- Given an algorithm **size-VC** that returns the **size** of a min VC in time **T(n),**
- Show an algorithm **find-VC** that returns a min VC in **poly(T(n)) time.**

Common Strategy: Go through each vertex and consider whether removing it changes the size of the solution.

Idea of **find-VC(G)**:

1. Call **size-VC(G)**
2. Pick an arbitrary vertex **v** and remove it (and its incident edges) to get G-v.
3. Call **size-VC(G-v)**
   a. If the answer stayed the same:
      All min-VC exclude v ⇒ ignore v
   a. If the answer decreased by 1:
      Some min-VC includes v ⇒ Add v to the solution. Delete v.

Reminder of VC: set S of vertices so that every edge has at least one endpoint in S



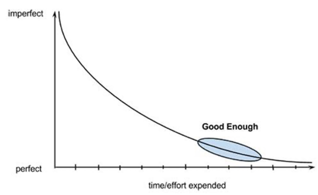
Goal:

- Given an algorithm **decide-SAT** that decides if a formula is satisfiable in time **T(n),**
- Show an algorithm **find-SAT** that returns a satisfying assignment in **poly(T(n)) time.**

find-SAT(φ):
  If decide-SAT(φ) = no: return ⊥ (no satisfying assignment)
  for each variable x_i
    if decide-SAT($\varphi_{x_i \leftarrow T}$) = yes:
      $\varphi \leftarrow \varphi_{x_i \leftarrow T}$
      $x_i \leftarrow T$
    if decide-SAT($\varphi_{x_i \leftarrow F}$) = yes:
      $\varphi \leftarrow \varphi_{x_i \leftarrow F}$
      $x_i \leftarrow F$

Example:
$\varphi = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_3) \wedge (x_2 \vee x_3)$
$\varphi_{x_1 \leftarrow T} = (T \vee x_2) \wedge (F \vee x_3) \wedge (x_2 \vee x_3)$
$= (x_3) \wedge (x_2 \vee x_3)$

Now on to **Approximation Algorithms**



Approximating Minimum VC

An algorithm is an **α-approximation** for the VC problem if it returns a VC that contains *at most* α times as many vertices as a min VC.

OPT ≤ ALG ≤ α · OPT,   α > 1

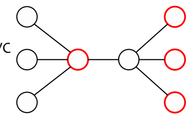
Optimal solution size

Solution size returned by our algorithm

α is called the ***approximation ratio*** (smaller is better here).

We will show that VC has a polynomial-time 2-approximation.

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

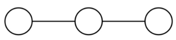


Coffee Shop CEO said: "I'm ok with building at most twice as many stores as is optimal."

Approximating Minimum VC

Check out my algorithm! Pick an arbitrary vertex covering at least one edge, delete it, and repeat!

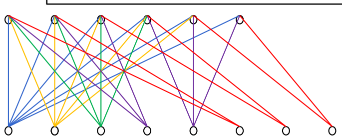
cover-and-remove(graph G):
1. $C \leftarrow \emptyset$
2. **while** G has an edge:
3. pick a vertex v *covering at least one edge*
4. $G \leftarrow G - v$; $C \leftarrow C \cup \{v\}$ // delete/add it to cover
5. **return** C



Approximating Minimum VC

I have another idea! Pick the vertex covering the most edges!

greedy-cover-and-remove(graph G):
1. $C \leftarrow \emptyset$
2. **while** G has an edge:
3. pick a vertex v covering *the most edges*
4. $G \leftarrow G - v$; $C \leftarrow C \cup \{v\}$ // delete/add it to cover
5. **return** C

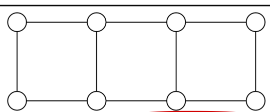


An extension of this idea shows that the approximation ratio is α = Ω(log n) (we won't prove)

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).

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

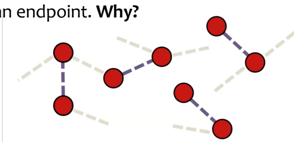


Theorem: double-cover obtains a 2-approx!

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 = **2** · (#edges chosen)

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

⇒ (# edges chosen) ≤ **OPT**

So **ALG** ≤ **2** · **OPT**

Doesn't this mean every NP-complete problem has a 2-approximation since they're all reducible to each other?

No!

2 reasons:

1. Some problems are minimization, some are maximization, and some are neither.

Minimization: OPT	≤ ALG ≤ α · OPT,	α > 1	α is the *approximation ratio*
Maximization: OPT.	≥ ALG ≥ α · OPT,	α < 1	

2. Reductions don't necessarily imply **anything** about approximation

Consider the following example…

Last time we showed that an n-vertex graph G has a **VC of size ≤k** if and only if G has an **IS of size ≥n-k**.

(This can show both VC ≤p IS and IS ≤p VC. Most reductions cannot be immediately reversed, but this one can since the graph doesn't change.)

E.g. Consider a graph G with **max-IS size n/2** and **min-VC size n/2**.

Running our **2-approx for VC** on G gives a **VC of size ≤n**, which translates to an **IS of size ≥n-n=0**.

So **IS-OPT = n/2**, **IS-ALG ≥ 0**, and the approximation ratio **α is zero**.

Conclusion: Even though a poly-time mapping reduction shows IS ≤p VC, the 2-approximation algorithm for VC doesn't imply <u>anything</u> about approximation algorithms for IS.

≤p 不代表 approximation algo也有关系

NP-complete problems come in many types:

- Some can be approximated to within a constant factor
  - e.g. VC
- Some can only be approximated to within a larger factor
  - e.g. Set Cover has an O(log n)-approximation and there's no better approximation ratio unless P = NP
- Some have no non-trivial approximation at all unless P = NP
  - e.g. Clique and Independent Set
- Some can be approximated *arbitrarily well*
  - e.g. Knapsack