LECTURE 25 (November 19^{th})

NP-hardness & Approximation Algorithms

We have seen polynomial time algorithms for different kinds of problems
But often in real life, the problems we want to solve seem to be hard

- How do we recognize it?
- How do we get around it?

The complexity classes P & NP

A way to recognize that your problem is hard is via the theory of NP-hardness, which allows us to categorize the problems according to their difficulty in a way.

To simplify our life, for the moment we will only look at decision problems

$$\text{input} \longrightarrow \boxed{} \longrightarrow \text{output} \in \{\text{YES}, \text{NO}\}$$

where the output is a decision or a boolean value

The complexity class P is the class of (decision) problems for which there are polynomial time algorithms.

The complexity class NP is the class of (decision) problems for which we can verify the answer is correct in polynomial time given a proof of this fact.

For example, given a 3-SAT instance

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_4 \vee x_5) \wedge \ldots$$

one can verify that the formula is satisfiable or not in polynomial time given an assignment.

Thus, 3-SAT $\in$ NP

In general, $P \subseteq NP$ but it is believed that $P \subsetneq NP$ since it seems difficult to find proofs that would verify the solution in polynomial time. For instance, for 3-SAT it is believed that the best algorithm to find a satisfying assignment must take exponential time.

NP-hard problems are problems that are harder than any problem in NP, e.g. harder than 3-SAT.

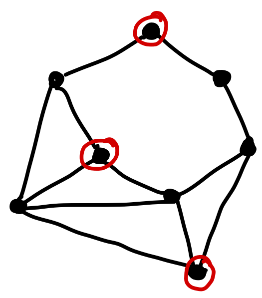
Thus, if you can show that your problem is NP-hard, you have shown that one can not expect a polynomial time algorithm

How to prove NP hardness? Via Reductions

Show that if you are given an algorithm for your problem, you can use it to solve some known NP-hard problem, e.g. 3-SAT, with polynomially many calls to the subroutine. It is easiest to explain it via an example

Maximum Independent Set

Let G be an undirected graph. An independent set in G is a subset of vertices with no edge between any two vertices in the set.



The red vertices form an independent set

The maximum independent set problem asks if given a graph G and an integer K, whether the maximum independent set in the graph has size at least k.

We will show that this problem is NP-hard by showing that if there is an algorithm for this problem, then using it as a subroutine polynomially many times, we can solve 3-SAT.

This would mean that it is unlikely that this problem would have a poly-time algorithm otherwise you would find a poly-time algorithm for 3-SAT which is not believed to exist.

Suppose we are given a 3-SAT instance

$$(x_1 \vee x_2 \vee x_3) \wedge (x_i \vee x_j \vee \overline{x}_k) \wedge \cdots\cdots$$

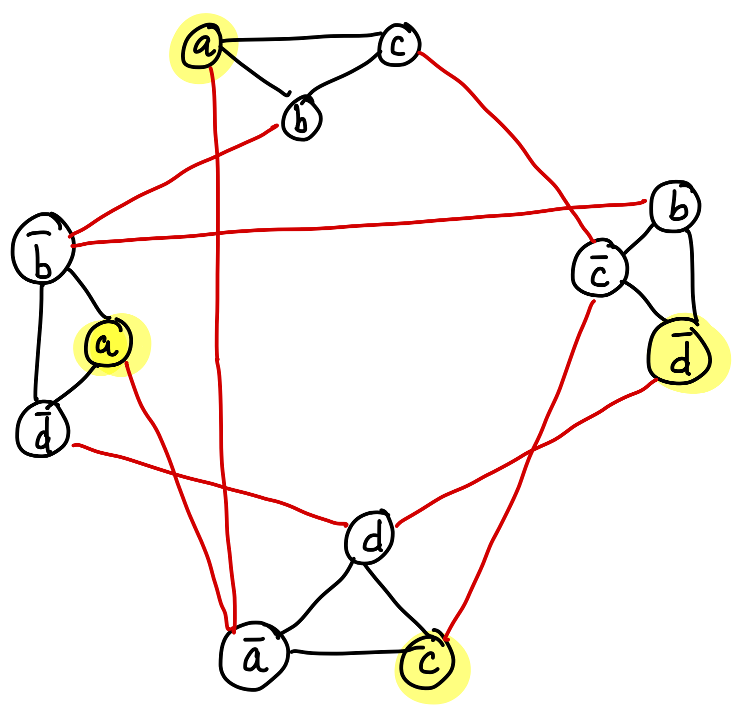
with n variables & m clauses.

We will construct a graph G as follows.

G will have $3m$ vertices, one for each literal in the clauses.
Two vertices will have an edge iff
either (1) they correspond to literals in the same clause
(2) they correspond to a variable and its negation

For example,

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d}) \text{ gives}$$



This graph can be constructed in linear time.

Now, to decide if the 3-SAT instance is satisfiable or not, we need to call our algorithm that solves the independent set problem

We will show that

- if G has an independent set of size m $\Rightarrow$ 3-SAT formula is satisfiable
  (i.e. max independent set has size $\geq m$)

- if G has no independent set of size m $\Rightarrow$ 3-SAT formula is not satisfiable
  (i.e. max independent set has size $< m$)

$\boxed{\text{Proof}}$ Largest independent set in G has size $\leq m$ since one can only choose one vertex from each clause in the independent set

Claim: Max. independent set size $= m \Leftrightarrow$ 3-SAT formula is satisfiable

If 3-SAT formula is satisfiable, then a satisfying assignment gives us an independent set of size m — just pick one literal in each clause that are true under this assignment, which exists since the formula is satisfiable. This must be an independent set (Why?)

Similarly, if there is an independent set of size m, then this would also give us a satisfying assignment.

Thus, max independent set is NP-hard

So, how do we deal with hard problems ?

(1) Assume our input has more structure or its random
(2) Be satisfied with approximate solutions

Approximation Algorithms

Suppose you are trying to solve an optimization problem (e.g. finding the size of the maximum independent set) on an input x

Let $OPT(x)$ denote the true optimal value & $A(x)$ be the value given by the algorithm

We say A gives an $\alpha(n)$ approximation if for every input $x \in \{0,1\}^n$, we have

$$\frac{OPT(x)}{\alpha(n)} \leq A(x) \leq OPT(x) \qquad \text{(for maximization problems)}$$

$$OPT(x) \leq A(x) \leq \alpha(n) \cdot OPT(x) \qquad \text{(for minimization problems)}$$

For many NP-hard problems, we can still find good approximation algorithms

Vertex Cover

Given an undirected graph $G = (V,E)$, find the size of the smallest vertex cover.

> Recall that a vertex cover is a set of vertices s.t. every edge touches some vertex in the cover

This problem is NP-hard, but it has a simple 2-approximation algorithm.

```
C ← ∅
while G has at least one edge
    {u,v} ← any edge in G
    G ← G \ {u,v}
    C ← C ∪ {u,v}
return C
```

The minimum vertex cover — in fact every vertex cover contains at least one of the two vertices u and v chosen inside the while loop. Thus, this is a 2 approximation algorithm. It is believed that $2-\varepsilon$ approximation is hard, so this dumb algorithm is the best one can hope for.

To design approximation algorithms for other problems, we need more powerful tools, such as linear programming. Let's see an example.

Lightest Vertex Cover

Given a graph $G = (V,E)$ where every vertex v has a non-negative weight $w(v)$

Goal : Compute a vertex cover C with the minimum total weight $w(C) = \sum_{v \in C} w(c)$

The dumb approximation algorithm can be very bad here.

But one can get a 2-approximation algorithm by casting the problem as an integer linear program — this is just a linear program where some variables are constrained to be integers. In general, integer linear programming is also NP-hard, so this by itself is not useful without additional work.

For the lightest vertex cover, we can write the following integer linear program

$$\begin{aligned} \min \quad & \sum_v w(v)\, x_v \\ \text{subject to} \quad & x_u + x_v \geq 1 \qquad \forall \text{ edges } \{u,v\} \\ & x_u \in \{0,1\} \qquad \forall \text{ vertex } v \end{aligned}$$

Let OPT = weight of lightest vertex cover

Then, objective value of the integer linear program is OPT.

As we said before, we can't solve this integer linear program but what if we relax the integer constraints.

$$\begin{aligned} \min \quad & \sum_v w(v)\, x_v \\ \text{subject to} \quad & x_u + x_v \geq 1 \qquad \forall \text{ edges } \{u,v\} \\ & 0 \leq x_u \leq 1 \qquad \forall \text{ vertex } v \end{aligned}$$

This becomes a linear program which we can solve in poly-time, but is it useful ?

This linear program is called the LP relaxation of the integer linear program.

Let x^* denote the optimal solution of the LP relaxation and $OPT^* = \sum_v w(v) x_v^*$ its optimal value.

Unfortunately, x^* may not be integral, since $x_v^* \in [0,1]$ may be fractional and it does not correspond to a vertex cover directly. x^* is called the optimal fractional solution. Nevertheless, this is still useful for two reasons

1. Every feasible integral solution to the original integer linear program is also a feasible solution to the LP relaxation.
   Thus,

   $$OPT \gtrsim OPT^* \quad \text{as the LP value can be even smaller}$$

2. We can derive a good approximation to the lightest vertex cover by rounding the optimal fractional solution. Specifically,

   $$\text{if } x_v^* \geq \tfrac{1}{2} \implies \text{round to } x_v = 1$$

   $$\text{if } x_v^* < 1/2 \implies \text{round to } x_v = 0$$

   For every edge $\{u,v\}$, the constraint $x_u^* + x_v^* \geq 1$

   $$\implies \max(x_u^*, x_v^*) \geq 1/2$$

   Thus, either $x_u = 1$ or $x_v = 1$
   so, x is the indicator vector for a vertex cover.

   On the other hand, $x_v \leq 2x_v^* \quad \forall$ vertex v.

   Therefore, $$\sum_v w(v) x_v \leq 2 \sum_v w(v) x_v^* = 2 \cdot \underbrace{OPT^*}_{\text{LP relaxation objective value}} \leq 2 \cdot \underbrace{OPT}_{\text{Integer LP objective value}}$$

Thus, we obtain a simple 2-approximation algorithm.

General recipe : 1. Write down an integer linear program (ILP) for your problem

2. Relax it to obtain an LP & solve the relaxation to obtain a fractional solution

3. Round the fractional solution to obtain an approximate solution. $\Rightarrow$ Next time : other techniques for rounding