Algorithm Design, Analysis & Complexity

Lecture 9 - $\mathcal{NP}$ Completeness & Computational Intractability

Koushik Pal

University of Toronto

July 20, 2021

Decision Problems

Definition

A **decision problem** is one which has a “yes/no” answer.

Example

- Given graph G and vertices s, t, is there a path from s to t?
- Given weighted graph G and bound b, is there a spanning tree of G with total weight $\leq b$?

Why decision problems?

Question

Why limit to decision problems?

Answer

We want to prove negative results. Intuitively, decision problems are "easier" than corresponding optimization/search problems. So, if we can show that a decision problem is hard (has no known efficient solution), this would imply that the more general problem is also hard.

It turns out that, in most cases, an optimization/search problem can be solved with the help of an algorithm for a corresponding decision problem.

Formalization

Input to a computational problem is a finite binary string s, for example, $s = 011100\ldots001$. Thus, $s \in \{0,1\}^n$, where $n = |s| =$ length of s.

We denote the set of all **finite binary strings** by $\{0,1\}^*$. In other words,

$$\{0,1\}^* = \bigcup_{n\in\mathbb{N}} \{0,1\}^n.$$

We denote a **decision problem** X as

$$X = \{s \in \{0,1\}^* \mid X(s) = \text{“\textit{yes}”}\}.$$

Polynomial Time Reducibility

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is called **polynomial-time computable** if there exists a polynomial-time algorithm A that, given any input $x \in \{0,1\}^*$, produces as output $f(x)$.

Let X and Y be two problems. We say Y is **polynomial-time reducible** to X (or, X is at least as hard as Y w.r.t. polynomial time) if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that

$$x \in Y \iff f(x) \in X \quad \forall x \in \{0,1\}^*.$$

Notation: $Y \leq_p X$.

Properties of $\leq_p$

Fact

1. *Suppose* $Y \leq_p X$*. If* X *can be solved in polynomial time, then* Y *can be solved in polynomial time.*
2. *Suppose* $Y \leq_p X$*. If* Y *cannot be solved in polynomial time, then* X *cannot be solved in polynomial time.*
3. *If* $Z \leq_p Y$ *and* $Y \leq_p X$*, then* $Z \leq_p X$*.*

Proof.

Let f *and* g *be polytime functions that witness* $Z \leq_p Y$ *and* $Y \leq_p X$ *respectively. i.e.,*

$$\begin{aligned} z \in Z &\iff f(z) \in Y \\ y \in Y &\iff g(y) \in X. \end{aligned}$$

It follows that

$$z \in Z \iff f(z) \in Y \iff g(f(z)) \in X \iff (g \circ f)(z) \in X.$$

Since $g \circ f$ *is a polytime function, we obtain* $Z \leq_p X$*.* $\square$

IndependentSet

Definition

A set $S \subseteq V$ of nodes in a graph $G = (V, E)$ is called **independent** if no two nodes in S are connected by an edge.

Note

A challenging problem is to find a largest independent set.

Although optimization problems seem harder than decision problems in general, they are equivalent to a certain extent.

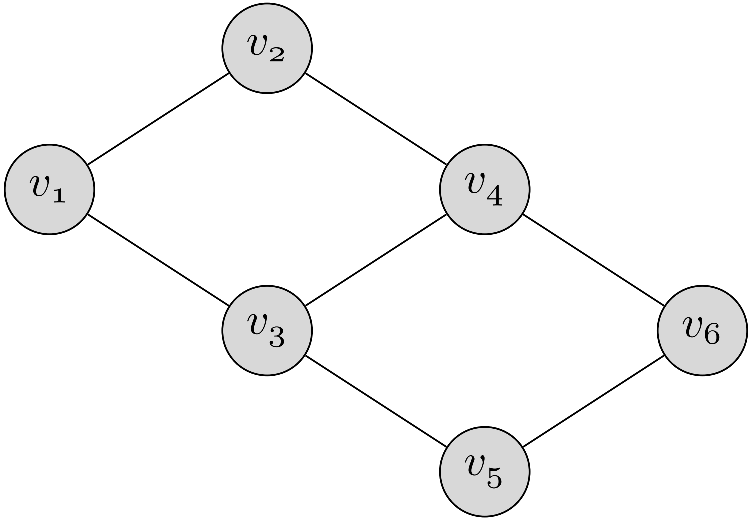
For example, define the following decision problem:

Definition (IndependentSet)

Given a graph G and a number k, does G contain an independent set of size at least k?

Then, it it easy to see that solving the optimization problem is equivalent to solving $\mathcal{O}(\lg n)$ many decision problems.

Example



$\{v_1, v_4, v_5\}$ and $\{v_2, v_3, v_6\}$ are the largest independent sets in G.

VertexCover

Definition

A set $S \subseteq V$ of nodes in a graph $G = (V, E)$ is called a **vertex cover** if every edge $e \in E$ has at least one end in S.

Note

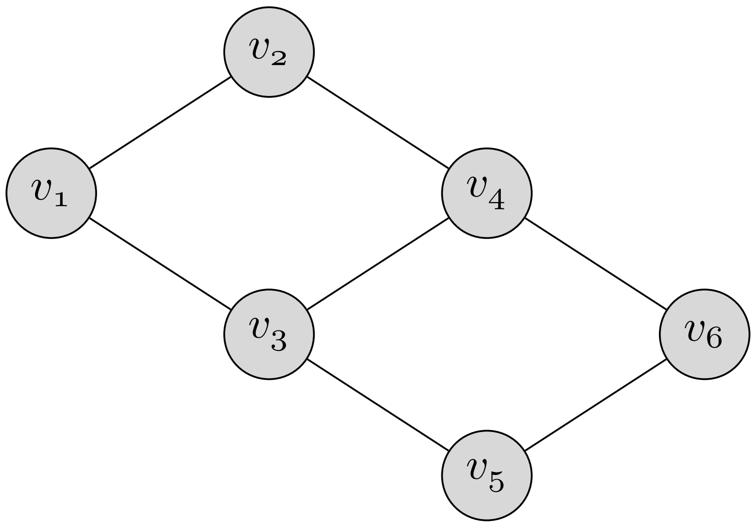
A challenging problem is to find a vertex cover of the least size.

Once again, define the corresponding decision problem:

Definition (VertexCover)

Given a graph G and a number k, does G contain a vertex cover of size at most k?

Example



$\{v_1, v_4, v_5\}$ and $\{v_2, v_3, v_6\}$ are the smallest vertex covers in G.

Relation between independent set and vertex cover

Lemma

Let $G = (V, E)$ be a graph. Then $S \subseteq V$ is an independent set iff its complement $V \setminus S$ is a vertex cover.

Proof.

($\Longrightarrow$) Suppose S is an independent set.
Consider an arbitrary edge $e = (u, v) \in E$.
Since S is independent, it cannot be the case that both u and v are in S; so one of them must be in $V \setminus S$.
Since e is arbitrary, it follows that $V \setminus S$ is a vertex cover.

($\Longleftarrow$) Suppose $V \setminus S$ is a vertex cover.
Consider two arbitrary nodes $u, v \in S$.
If they are joined by an edge e, then neither end of e would lie in $V \setminus S$, contradicting our assumption that $V \setminus S$ is a vertex cover.
Since u, v are arbitrary, it follows that S is an independent set.

$\square$

Relation between $\textsc{IndependentSet}$ & $\textsc{VertexCover}$

Corollary

$\textsc{IndependentSet} \leq_p \textsc{VertexCover}$.

Proof.

Define a function $f : \mathcal{P}(V) \to \mathcal{P}(V)$ as follows:

$$f(S) = V \setminus S.$$

Clearly, f is polytime. Moreover,

$$S \in \textsc{IndependentSet} \iff V \setminus S \in \textsc{VertexCover}.$$

Hence, $\textsc{IndependentSet} \leq_p \textsc{VertexCover}$. $\square$

Corollary

$\textsc{VertexCover} \leq_p \textsc{IndependentSet}$.

Proof.

Similar argument. $\square$

SetCover

Definition (SetCover)

Given a set U of n elements, a collection $S_1, \ldots, S_m$ of subsets of U, and a number k, does there exist a collection of at most k of these sets whose union is equal to all of U?

Lemma

$\textsc{VertexCover} \leq_p \textsc{SetCover}$

Proof.

Consider an arbitrary instance of VertexCover, specified by a graph $G = (V, E)$, and a number k.

Define a function $f : V \to \mathcal{P}(E)$ as follows: for each $i \in V$, define

$$f(i) = S_i = \text{set of edges } e \in E \text{ incident to } i.$$

Clearly, f is polytime.

Proof

If $\{S_{i_1}, \ldots, S_{i_\ell}\}$ are $\ell \leq k$ sets that cover E, then every edge in G is incident to one of the vertices $i_1, \ldots, i_\ell$, and so the set $\{i_1, \ldots, i_\ell\}$ is a vertex cover in G of size $\ell \leq k$.

Conversely, if $\{i_1, \ldots, i_\ell\}$ is a vertex cover in G of size $\ell \leq k$, then the sets $S_{i_1}, \ldots, S_{i_\ell}$ cover E.

In other words,

$$(G = (V, E), k) \in \textsc{VertexCover}$$
$$\iff (E, S_1, \ldots, S_n, k) \in \textsc{SetCover}.$$

$\square$

SetPacking

Definition (SetPacking)

Given a set U of n elements, a collection $S_1, \ldots, S_m$ of subsets of U, and a number k, does there exist a collection of at least k of these sets with the property that no two of them intersect?

Lemma

IndependentSet $\leq_p$ SetPacking

Proof.

Exercise! $\square$

Finding vs checking a solution

Contrast between "finding" a solution and "checking" a solution.

Checking a proposed solution for an independent set or a vertex cover problem is easy!

But think about what "evidence" could we show to convince someone in polynomial time that a graph G does not have any independent set of size k?

Class $\mathcal{P}$

Definition

An **algorithm** A for a decision problem X receives an input string s and returns "yes" or "no". The output is denoted by $A(s)$.

We say A **solves** the problem X if for all strings s, we have

$$A(s) = \text{"}yes\text{"} \iff s \in X.$$

We say A has a **polynomial running time** if there is a polynomial function $p(\cdot)$ so that for every input string s, the algorithm A terminates on s in at most $\mathcal{O}(p(|s|))$ steps.

Definition ($\mathcal{P}$)

$$\mathcal{P} := \{X \mid \exists \text{ an algorithm } A \text{ with a polynomial running time that solves the decision problem } X\}.$$

Class $\mathcal{NP}$ - Informal Definition

A "checking algorithm" for a problem X has a different structure: in order to "check" whether an input string s is a solution, we need the input string s, as well as a separate "certificate" string t that contains the evidence that s is a "yes" instance of X.

Definition (Informal definition of $\mathcal{NP}$)

$$\mathcal{NP} := \{X \quad | \quad \exists \text{ a "generate-and-verify" algorithm as follows that solves the decision problem } X\}.$$

```
procedure GenerateAndVerify(s)
    generate all certificates t
    for each certificate t do
        if verify(s, t) then
            return True
    return False
```

where the verification phase (the call to verify(s, t)) runs in worst-case polynomial time, as a function of size(s).

$\mathcal{NP}$ Example

$\textsc{Composite}$: Given positive integer s, does s have any factor?

Fact

$\textsc{Composite} \in \mathcal{NP}$ because it is solved by the following generate-and-verify algorithm:

```
procedure COMPOSITE(s)
    for all integers t = 2, ..., s−1 do
        if t divides s then
            return True
    return False
```

Note that this is an exponential algo since the complexity is

$$\Theta(s) = \Theta(2^{\lg s}) = \Theta(2^{\mathsf{size}(s)}).$$

Class $\mathcal{NP}$ - Formal Definition

Definition (Efficient Certifier)

B is an **efficient certifier** for a problem X if the following properties hold:

- B is a polynomial-time algorithm that takes two input arguments s and t.
- There is a polynomial function p so that for every string s, we have $s \in X \iff$ there exists a string t such that $|t| \leq p(|s|)$ and $B(s, t) =$ "yes".

(B says that an input $s \in X \iff$ there exists a proposed proof t that is not too long and that will convince that $s \in X$.)

Definition (Formal definition of $\mathcal{NP}$)

$$\mathcal{NP} := \{X \mid \exists \text{ an efficient certifier for decision problem } X\}.$$

Relationship between $\mathcal{P}$ and $\mathcal{NP}$

Theorem
$\mathcal{P} \subseteq \mathcal{NP}$.

Proof.
Consider a problem $X \in \mathcal{P}$. By definition, there is a polynomial time algorithm A that solves X.

Design an efficient certifier B for X as follows:

$$B(s,t) := A(s) \qquad (B \text{ simply ignores } t)$$

1. Clearly B is a polynomial time algorithm since A is.
2. If $s \in X$, then for every t of length at most $p(|s|)$, we have $B(s,t) =$ "yes".
3. If $s \notin X$, then for every t of length at most $p(|s|)$, we have $B(s,t) =$ "no".

Thus, B is an efficient certifier for X. Hence, $X \in \mathcal{NP}$. $\square$

More $\mathcal{NP}$ examples

Example

1. $\textsc{IndependentSet} \in \mathcal{NP}$: the certificate t is a set of at least k vertices, and the certifier B checks that, for these vertices, no edge joins any pair of them.
2. $\textsc{SetCover} \in \mathcal{NP}$: the certificate t is a list of k sets from the given collection, and the certifier B checks that the union of these sets is equal to the underlying set U.
3. $\textsc{VertexCover} \in \mathcal{NP}$: exercise.
4. $\textsc{SetPacking} \in \mathcal{NP}$: exercise.

Challenging Fundamental Question

Question

Is $\mathcal{P} = \mathcal{NP}$?

Class $co\text{–}\mathcal{NP}$

Definition

$co\text{–}\mathcal{NP}$:= complements of problems in $\mathcal{NP}$, i.e., problems whose no-instances can be verified in polytime, but for which we have no information about yes-instances.

$D \in co\text{–}\mathcal{NP}$ means there is a verifier $B(s,t)$ running in polynomial time such that

$$\begin{array}{lll} B(s,t) = \textit{False} & \text{for some } t & \text{whenever } s \text{ is a no-instance} \\ B(s,t) = \textit{True} & \text{for all } t & \text{whenever } s \text{ is a yes-instance.} \end{array}$$

Remark

Recall that $D \in \mathcal{NP}$ means there is a verifier $B(s,t)$ running in polynomial time such that

$$\begin{array}{lll} B(s,t) = \textit{True} & \textit{for some } t & \textit{whenever } s \textit{ is a yes-instance} \\ B(s,t) = \textit{False} & \textit{for all } t & \textit{whenever } s \textit{ is a no-instance.} \end{array}$$

$co\text{–}\mathcal{NP}$ example

Example

PRIME: Given positive integer x, is x prime?

PRIME $\in co\text{–}\mathcal{NP}$ since COMPOSITE $\in \mathcal{NP}$.

Alternative Proof:

```
procedure PRIME(s)
    for all integers t = 2, ..., s−1 do
        if t divides s and s ≠ 2 then
            return False
    return True
```

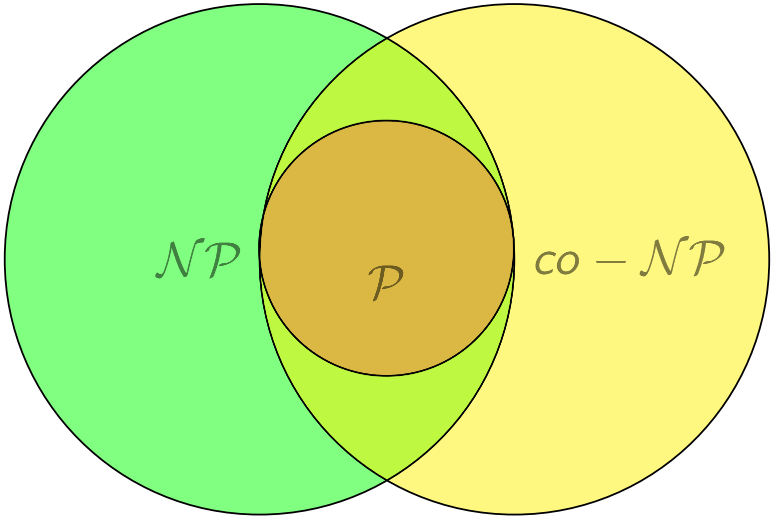
Fact

$\mathcal{P} \subseteq co\text{–}\mathcal{NP}$.

Proof.

Exercise! $\square$

Class Relationships



Believed (but not known yet): $\mathcal{P} \neq \mathcal{NP}$ and $\mathcal{NP} \neq co\text{–}\mathcal{NP}$.

Unknown: $\mathcal{P} = \mathcal{NP} \bigcap co\text{–}\mathcal{NP}$?

$\mathcal{NP}$-complete problems

We use $\leq_p$ to identify "hardest problems" in $\mathcal{NP}$.

Definition

Decision problem D is called **$\mathcal{NP}$-complete** if

1. $D \in \mathcal{NP}$
2. D is **$\mathcal{NP}$-hard**: for all $D' \in \mathcal{NP}$, we have $D' \leq_p D$.

Power of $\mathcal{NP}$-completeness

Theorem

If D is $\mathcal{NP}$-complete, then $D \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$.

Proof.

($\Longleftarrow$) Since D is $\mathcal{NP}$-complete, we have $D \in \mathcal{NP}$. If $\mathcal{P} = \mathcal{NP}$, it follows that $D \in \mathcal{P}$.

($\Longrightarrow$) Conversely, since D is $\mathcal{NP}$-complete, we have $D' \leq_p D$ for all $D' \in \mathcal{NP}$.
Since $D \in \mathcal{P}$, it follows that $D' \in \mathcal{P}$ for all $D' \in \mathcal{NP}$.
Consequently, $\mathcal{NP} \subseteq \mathcal{P}$.
Since we already know $\mathcal{P} \subseteq \mathcal{NP}$, it follows that $\mathcal{P} = \mathcal{NP}$.

$\square$

Relationship between $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complete & $\mathcal{NP}$-hard