Learning Goals

- Define class P and NP.
- Understand the relationship between P and NP.
- Define what is an NP-complete problem.
- State the decision problems SAT and 3-SAT.
- State Cook-Levin Theorem.
- Master the procedure to prove a problem is NP-complete.
- Understand the reduction from 3-SAT to INDEPENDENT SET.

The classes P

Note: In this lecture we cannot get into the nuts and bolts of some definitions or theorems, because we haven't defined a computation model (e.g. Turing machine). Nevertheless, everything stated can be rigorously proved.

Definition

The class P is the set of all decision problems that can be solved in polynomial time.

The classes P

Note: In this lecture we cannot get into the nuts and bolts of some definitions or theorems, because we haven't defined a computation model (e.g. Turing machine). Nevertheless, everything stated can be rigorously proved.

Definition

The class P is the set of all decision problems that can be solved in polynomial time.
Recall from 221/320: by polynomial time, we mean polynomial in the length of the input.

The classes P

Note: In this lecture we cannot get into the nuts and bolts of some definitions or theorems, because we haven't defined a computation model (e.g. Turing machine). Nevertheless, everything stated can be rigorously proved.

Definition

The class P is the set of all decision problems that can be solved in polynomial time.
Recall from 221/320: by polynomial time, we mean polynomial in the length of the input.

Example: decision versions of Shortest Path, Minimum Spanning Tree, Max Flow, Min Cut, Bipartite Matching, Baseball Elimination...

The class NP

Definition

A decision problem A is in class NP (*nondeterministic polynomial time*) if there exists a *polynomial-time verifier* algorithm V for the following task:

1. if the answer to an instance a of A is YES, then there exists a *polynomial-length certificate* $c(a)$, such that V, when provided with both the instance a and the certificate, will return yes;
2. if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.

The class NP

Definition

A decision problem A is in class NP (*nondeterministic polynomial time*) if there exists a *polynomial-time verifier* algorithm V for the following task:

1. if the answer to an instance a of A is YES, then there exists a *polynomial-length certificate* $c(a)$, such that V, when provided with both the instance a and the certificate, will return yes;
2. if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.

Example: INDEPENDENT SET

The class NP

Definition

A decision problem A is in class NP (*nondeterministic polynomial time*) if there exists a *polynomial-time verifier* algorithm V for the following task:

1. if the answer to an instance a of A is YES, then there exists a *polynomial-length certificate* $c(a)$, such that V, when provided with both the instance a and the certificate, will return yes;
2. if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.

Example: INDEPENDENT SET

- Input: graph G, integer k

The class NP

Definition

A decision problem A is in class NP (*nondeterministic polynomial time*) if there exists a *polynomial-time verifier* algorithm V for the following task:

1. if the answer to an instance a of A is YES, then there exists a *polynomial-length certificate* $c(a)$, such that V, when provided with both the instance a and the certificate, will return yes;
2. if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.

Example: INDEPENDENT SET

- Input: graph G, integer k
- Certificate: a set S of nodes in G

The class NP

Definition

A decision problem A is in class NP (*nondeterministic polynomial time*) if there exists a *polynomial-time verifier* algorithm V for the following task:

1. if the answer to an instance a of A is YES, then there exists a *polynomial-length certificate* $c(a)$, such that V, when provided with both the instance a and the certificate, will return yes;
2. if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.

Example: INDEPENDENT SET

- Input: graph G, integer k
- Certificate: a set S of nodes in G
- Verifier: check whether S is an independent set, and whether $|S| \geq k$. If so, return YES; if not, return NO.

Relationship between P and NP

Proposition

P $\subseteq$ NP.

Relationship between P and NP

Proposition

P $\subseteq$ NP.

Proof.

Given any problem in P, let the verifier V be a polynomial-time algorithm that solves the problem . Let the certificate be $\emptyset$. $\square$

Relationship between P and NP

Proposition

P $\subseteq$ NP.

Proof.

Given any problem in P, let the verifier V be a polynomial-time algorithm that solves the problem . Let the certificate be $\emptyset$. $\square$

Question

NP $\subseteq$ P?

Relationship between P and NP

Proposition

P $\subseteq$ NP.

Proof.

Given any problem in P, let the verifier V be a polynomial-time algorithm that solves the problem . Let the certificate be $\emptyset$. $\square$

Question

NP $\subseteq$ P?

One of the most famous questions in (theoretical) computer science.

Relationship between P and NP

Proposition

P $\subseteq$ NP.

Proof.

Given any problem in P, let the verifier V be a polynomial-time algorithm that solves the problem . Let the certificate be $\emptyset$. $\square$

Question

NP $\subseteq$ P?

One of the most famous questions in (theoretical) computer science.
Some philosophical discussion.

NP Completeness

- A problem A in a class $\mathcal{C}$ of problems is said to be *$\mathcal{C}$-complete* if all problems in $\mathcal{C}$ can be reduced to A by a meaningful reduction.

NP Completeness

- A problem A in a class $\mathcal{C}$ of problems is said to be *$\mathcal{C}$-complete* if all problems in $\mathcal{C}$ can be reduced to A by a meaningful reduction.
  - Intuitively, a problem complete in $\mathcal{C}$ is a hardest problem in $\mathcal{C}$.

NP Completeness

- A problem A in a class $\mathcal{C}$ of problems is said to be *$\mathcal{C}$-complete* if all problems in $\mathcal{C}$ can be reduced to A by a meaningful reduction.
  - Intuitively, a problem complete in $\mathcal{C}$ is a hardest problem in $\mathcal{C}$.
- For the class NP, the "meaningful" reductions are polynomial-time reductions.

NP Completeness

- A problem A in a class $\mathcal{C}$ of problems is said to be *$\mathcal{C}$-complete* if all problems in $\mathcal{C}$ can be reduced to A by a meaningful reduction.
  - Intuitively, a problem complete in $\mathcal{C}$ is a hardest problem in $\mathcal{C}$.
- For the class NP, the "meaningful" reductions are polynomial-time reductions.

Definition

A problem is NP-*complete* if it is in NP and if all other problems in NP can be polynomial-time reduced to it.

NP Completeness

- A problem A in a class $\mathcal{C}$ of problems is said to be *$\mathcal{C}$-complete* if all problems in $\mathcal{C}$ can be reduced to A by a meaningful reduction.
  - Intuitively, a problem complete in $\mathcal{C}$ is a hardest problem in $\mathcal{C}$.
- For the class NP, the "meaningful" reductions are polynomial-time reductions.

Definition

A problem is NP-*complete* if it is in NP and if all other problems in NP can be polynomial-time reduced to it.
Formally, a problem A is NP-complete if $A \in \mathrm{NP}$ and, $\forall B \in \mathrm{NP}$, $B \leq_{\mathrm{P}} A$.

SAT and Cook-Levin Theorem

Definition

In a *Boolean satisfiability (SAT)* problem, we are given a Boolean formula in *conjunctive normal form (CNF)*; that is, the formula is the AND of (many) OR clauses. We must decide whether there is a way of assigning TRUE and FALSE to each variable so that the formula evaluates to TRUE.

SAT and Cook-Levin Theorem

Definition

In a *Boolean satisfiability (SAT)* problem, we are given a Boolean formula in *conjunctive normal form (CNF)*; that is, the formula is the AND of (many) OR clauses. We must decide whether there is a way of assigning TRUE and FALSE to each variable so that the formula evaluates to TRUE.

Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$

SAT and Cook-Levin Theorem

Definition

In a *Boolean satisfiability (SAT)* problem, we are given a Boolean formula in *conjunctive normal form (CNF)*; that is, the formula is the AND of (many) OR clauses. We must decide whether there is a way of assigning TRUE and FALSE to each variable so that the formula evaluates to TRUE.

Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$

Theorem (Cook-Levin)

SAT is NP*-complete.*

SAT and Cook-Levin Theorem

Definition

In a *Boolean satisfiability (SAT)* problem, we are given a Boolean formula in *conjunctive normal form (CNF)*; that is, the formula is the AND of (many) OR clauses. We must decide whether there is a way of assigning TRUE and FALSE to each variable so that the formula evaluates to TRUE.

Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$

Theorem (Cook-Levin)

SAT is NP*-complete.*

A meaningful proof needs a rigorous definition of Turing machines.

Review of last lecture

Definition

A decision problem A is in class NP (*nondeterministic polynomial time*) if there exists a *polynomial-time verifier* algorithm V for the following task:

1. if the answer to an instance a of A is YES, then there exists a *polynomial-length certificate* $c(a)$, such that V, when provided with both the instance a and the certificate, will return yes;
2. if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.

Definition

A problem A is NP-complete if $A \in \mathrm{NP}$ and, $\forall B \in \mathrm{NP}$, $B \leq_{\mathrm{P}} A$.

SAT Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$

Review of last lecture

Definition

A decision problem A is in class NP (*nondeterministic polynomial time*) if there exists a *polynomial-time verifier* algorithm V for the following task:

1. if the answer to an instance a of A is YES, then there exists a *polynomial-length certificate* $c(a)$, such that V, when provided with both the instance a and the certificate, will return yes;
2. if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.

Definition

A problem A is NP-complete if $A \in \mathrm{NP}$ and, $\forall B \in \mathrm{NP}$, $B \leq_{\mathrm{P}} A$.

SAT Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$

Theorem (Cook-Levin)

SAT is NP*-complete.*

3-SAT

Definition

A 3-SAT problem is a SAT problem wherein each clause contains three literals.

3-SAT

Definition

A 3-SAT problem is a SAT problem wherein each clause contains three literals.

Theorem

3-*SAT is* NP-*complete.*

3-SAT

Definition

A 3-SAT problem is a SAT problem wherein each clause contains three literals.

Theorem

3-SAT is NP*-complete.*

How do we show a new problem is NP complete?

Proposition

Polynomial reduction is transitive, i.e., if $A \leq_P B$, $B \leq_P C$, then $A \leq_P C$.

3-SAT

Definition

A 3-SAT problem is a SAT problem wherein each clause contains three literals.

Theorem

3-SAT is NP*-complete.*

How do we show a new problem is NP complete?

Proposition

Polynomial reduction is transitive, i.e., if $A \leq_P B$, $B \leq_P C$, then $A \leq_P C$.

Proof sketch: If the polynomial-time reduction from A to B runs in time $p_1(\cdot)$, and the reduction from B to C runs in time $p_2(\cdot)$, then A can be solved by concatenating the reductions, with oracle access to C, and running time $O(p_1(\cdot)p_2(\cdot))$, which is still polynomial.

Corollary

If a problem A is in NP*, and if there is any* NP*-complete problem B such that $B \leq_{\mathrm{P}} A$, then A is* NP*-complete.*

Corollary

If a problem A is in NP*, and if there is any* NP*-complete problem B such that $B \leq_{\mathrm{P}} A$, then A is* NP*-complete.*

Proof.

We need to show that, for any $C \in \mathrm{NP}$, $C \leq_{\mathrm{P}} A$.

Corollary

If a problem A is in NP*, and if there is any* NP*-complete problem B such that $B \leq_{\mathrm{P}} A$, then A is* NP*-complete.*

Proof.

We need to show that, for any $C \in \mathrm{NP}$, $C \leq_{\mathrm{P}} A$.

- Since B is NP complete, $C \leq_{\mathrm{P}} B$;

Corollary

If a problem A is in NP*, and if there is any* NP*-complete problem B such that $B \leq_{\mathrm{P}} A$, then A is* NP*-complete.*

Proof.

We need to show that, for any $C \in \mathrm{NP}$, $C \leq_{\mathrm{P}} A$.

- Since B is NP complete, $C \leq_{\mathrm{P}} B$;
- But $B \leq_{\mathrm{P}} A$, therefore $C \leq_{\mathrm{P}} A$ by proposition.

$\square$

Procedure to show NP completeness

Given a problem A, to show it is NP-complete, we show that

1. A is in NP. We show a polynomial-time verifier: for TRUE instances, show polynomial-length certificates that makes the verifier accept, and for FALSE instances, show the verifier never accepts;
2. Take an NP-complete problem B, and show $B \leq_{\mathrm{P}} A$. To do this, we
   - Give a *polynomial-time* algorithm φ which takes as input an instance of B and outputs an instance of A;
   - Show that an instance b of B has answer TRUE if and only if the instance $\varphi(b)$ has answer TRUE.

First example of NP-completeness reduction

Theorem

INDEPENDENT SET is NP *complete.*

First example of NP-completeness reduction

Theorem

INDEPENDENT SET is NP *complete.*

Proof.

1. INDEPENDENT SET is in NP. A certificate for a YES instance is an independent set of size k. The verifier checks its validity.

First example of NP-completeness reduction

Theorem

INDEPENDENT SET is NP *complete.*

Proof.

1. INDEPENDENT SET is in NP. A certificate for a YES instance is an independent set of size k. The verifier checks its validity.
2. We show 3-SAT $\leq_{\mathrm{P}}$ INDEPENDENT SET. Given a 3-SAT formula with m clauses, construct an undirected graph G:

First example of NP-completeness reduction

Theorem

INDEPENDENT SET is NP *complete.*

Proof.

1. INDEPENDENT SET is in NP. A certificate for a YES instance is an independent set of size k. The verifier checks its validity.
2. We show 3-SAT $\leq_P$ INDEPENDENT SET. Given a 3-SAT formula with m clauses, construct an undirected graph G:
    1. For each clause, construct three nodes representing the three literals;

First example of NP-completeness reduction

Theorem

INDEPENDENT SET is NP *complete.*

Proof.

1. INDEPENDENT SET is in NP. A certificate for a YES instance is an independent set of size k. The verifier checks its validity.
2. We show 3-SAT $\leq_P$ INDEPENDENT SET. Given a 3-SAT formula with m clauses, construct an undirected graph G:
   1. For each clause, construct three nodes representing the three literals;
   2. Connect any two nodes that represent, respectively, a variable and its negation (e.g. x_2 and $\neg x_2$);

First example of NP-completeness reduction

Theorem

INDEPENDENT SET is NP *complete.*

Proof.

1. INDEPENDENT SET is in NP. A certificate for a YES instance is an independent set of size k. The verifier checks its validity.
2. We show 3-SAT $\leq_{\mathrm{P}}$ INDEPENDENT SET. Given a 3-SAT formula with m clauses, construct an undirected graph G:
   1. For each clause, construct three nodes representing the three literals;
   2. Connect any two nodes that represent, respectively, a variable and its negation (e.g. x_2 and $\neg x_2$);
   3. For each clause, connect the three nodes representing its literals.

First example of NP-completeness reduction

Theorem

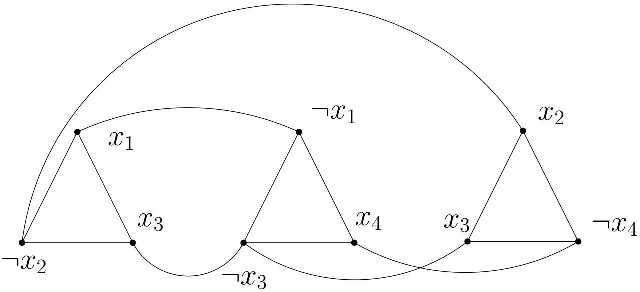
INDEPENDENT SET is NP *complete.*

Proof.

1. INDEPENDENT SET is in NP. A certificate for a YES instance is an independent set of size k. The verifier checks its validity.
2. We show 3-SAT $\leq_P$ INDEPENDENT SET. Given a 3-SAT formula with m clauses, construct an undirected graph G:
    1. For each clause, construct three nodes representing the three literals;
    2. Connect any two nodes that represent, respectively, a variable and its negation (e.g. x_2 and $\neg x_2$);
    3. For each clause, connect the three nodes representing its literals.

   Now show that the 3-SAT formula is satisfiable if and only if G has an independent set of size at least m.

Example instance



Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$

Proof cont.

Claim

The 3-SAT formula is satisfiable if and only if G has an independent set of size at least m.

Proof.

3-SAT satisfiable $\Rightarrow$ G having independent set S of size m:

Proof cont.

Claim

The 3-SAT formula is satisfiable if and only if G has an independent set of size at least m.

Proof.

3-SAT satisfiable $\Rightarrow$ G having independent set S of size m: Given a satisfying truth assignment, each clause has a literal that is true.

Proof cont.

Claim

The 3-SAT formula is satisfiable if and only if G has an independent set of size at least m.

Proof.

3-SAT satisfiable $\Rightarrow$ G having independent set S of size m: Given a satisfying truth assignment, each clause has a literal that is true. Include in S the corresponding node in the 3-cycle. Then $|S| = m$.

Proof cont.

Claim

The 3-SAT formula is satisfiable if and only if G has an independent set of size at least m.

Proof.

3-SAT satisfiable $\Rightarrow$ G having independent set S of size m: Given a satisfying truth assignment, each clause has a literal that is true. Include in S the corresponding node in the 3-cycle. Then $|S| = m$.
S is an independent set:

1. No edge in any triangle is in $E(S)$;
2. No edge connecting a variable and its negate is in $E(S)$.

Proof cont.

Claim

The 3-SAT formula is satisfiable if and only if G has an independent set of size at least m.

Proof.

G having an independent set S of size $m \Rightarrow$ 3-SAT formula satisfiable:

Proof cont.

Claim

The 3-SAT formula is satisfiable if and only if G has an independent set of size at least m.

Proof.

G having an independent set S of size $m \Rightarrow$ 3-SAT formula satisfiable:
S must have one node from each 3-cycle corresponding to a clause.

Proof cont.

Claim

The 3-SAT formula is satisfiable if and only if G has an independent set of size at least m.

Proof.

G having an independent set S of size $m \Rightarrow$ 3-SAT formula satisfiable:
S must have one node from each 3-cycle corresponding to a clause.
Construct a truth assignment by letting the corresponding literal be TRUE. (After this, if some variables don't have an assignment, give them arbitrary assignment.)

1. There is no contradiction in this assignment.
2. All clauses are satisfied.

A little summary

- We have shown: 3-SAT $\leq_P$ INDEPENDENT SET $\leq_P$ VERTEX COVER $\leq_P$ SET COVER
- These problems are all clearly in NP.
- Both SAT and 3-SAT are NP complete
- Therefore all these problems are NP complete.

A little summary

- We have shown: 3-SAT $\leq_P$ INDEPENDENT SET $\leq_P$ VERTEX COVER $\leq_P$ SET COVER
- These problems are all clearly in NP.
- Both SAT and 3-SAT are NP complete
- Therefore all these problems are NP complete.
- Note that VERTEX COVER can be solved in polynomial time for bipartite graphs.

A little summary

- We have shown: 3-SAT $\leq_P$ INDEPENDENT SET $\leq_P$ VERTEX COVER $\leq_P$ SET COVER
- These problems are all clearly in NP.
- Both SAT and 3-SAT are NP complete
- Therefore all these problems are NP complete.
- Note that VERTEX COVER can be solved in polynomial time for bipartite graphs.
- For non-bipartite graphs, maximum matching can still be solved in polynomial time. But the size of the smallest vertex cover can be strictly larger than the size of the maximum matching.