Polynomial-time Reductions

Disclaimer: Many definitions in these slides should be taken as "the intuitive meaning", as the precise meaning of some of the terms are hard to pin down without introducing the formal machinery of computational models (the Turing machine in particular).

Definition

A problem is a *decision problem* if its answer is either TRUE or FALSE.

Polynomial-time Reductions

Disclaimer: Many definitions in these slides should be taken as "the intuitive meaning", as the precise meaning of some of the terms are hard to pin down without introducing the formal machinery of computational models (the Turing machine in particular).

Definition

A problem is a *decision problem* if its answer is either TRUE or FALSE.

Definition

A decision problem A is *polynomial-time reducible* to a decision problem B if there exists an algorithm that solves any instance of A with a polynomial number of standard computational steps, plus a polynomial number of calls to a (black-box) oracle (think of it as a function) that solves instances of B.

- We denote this as $A \leq_{\mathrm{P}} B$.

Polynomial-time Reductions

Disclaimer: Many definitions in these slides should be taken as "the intuitive meaning", as the precise meaning of some of the terms are hard to pin down without introducing the formal machinery of computational models (the Turing machine in particular).

Definition

A problem is a *decision problem* if its answer is either TRUE or FALSE.

Definition

A decision problem A is *polynomial-time reducible* to a decision problem B if there exists an algorithm that solves any instance of A with a polynomial number of standard computational steps, plus a polynomial number of calls to a (black-box) oracle (think of it as a function) that solves instances of B.

- We denote this as $A \leq_{\mathrm{P}} B$.
- If $A \leq_{\mathrm{P}} B$, and B can be solved in polynomial time, then A can be solved in polynonmial time.

A special class of reductions: Karp reductions

- One way to show $A \leq_P B$: give a polynomial-time algorithm φ, with
  - Input: an instance a of A
  - Output: an instance $\varphi(a)$ of B
  - Guarantee: the answer to a is TRUE $\Leftrightarrow$ the answer to $\varphi(a)$ is TRUE.

A special class of reductions: Karp reductions

- One way to show $A \leq_P B$: give a polynomial-time algorithm φ, with
  - Input: an instance a of A
  - Output: an instance $\varphi(a)$ of B
  - Guarantee: the answer to a is TRUE $\Leftrightarrow$ the answer to $\varphi(a)$ is TRUE.
- The resulting polynomial-time reduction: take an instance a of A, run φ on a, and call the oracle to solve $\varphi(a)$, and return the answer to $\varphi(a)$.

A special class of reductions: Karp reductions

- One way to show $A \leq_P B$: give a polynomial-time algorithm φ, with
  - Input: an instance a of A
  - Output: an instance $\varphi(a)$ of B
  - Guarantee: the answer to a is TRUE $\Leftrightarrow$ the answer to $\varphi(a)$ is TRUE.
- The resulting polynomial-time reduction: take an instance a of A, run φ on a, and call the oracle to solve $\varphi(a)$, and return the answer to $\varphi(a)$.
- Such a reduction calls the oracle to solve B only once. This is called a *Karp* reduction (a.k.a. a *many-to-one* reduction).
- A general polynomial-time reduction is called a poly-time *Turing* reduction, a.k.a. a *Cook* reduction.

A special class of reductions: Karp reductions

- One way to show $A \leq_P B$: give a polynomial-time algorithm φ, with
  - Input: an instance a of A
  - Output: an instance $\varphi(a)$ of B
  - Guarantee: the answer to a is TRUE $\Leftrightarrow$ the answer to $\varphi(a)$ is TRUE.
- The resulting polynomial-time reduction: take an instance a of A, run φ on a, and call the oracle to solve $\varphi(a)$, and return the answer to $\varphi(a)$.
- Such a reduction calls the oracle to solve B only once. This is called a *Karp* reduction (a.k.a. a *many-to-one* reduction).
- A general polynomial-time reduction is called a poly-time *Turing* reduction, a.k.a. a *Cook* reduction.
- In this class we always do Karp reductions. In the sequel, whenever we say poly-time reductions, think of Karp reductions.

Polynomial time reduction example

Definition

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

Polynomial time reduction example

Definition

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

Definition

In the *Independent Set* problem, we are given a graph $G = (V, E)$ and an integer k, and answer whether there exists an independent set of G of size at least k.

Polynomial time reduction example

Definition

Given a graph $G = (V, E)$, a set of nodes $S \subseteq V$ is a *vertex cover* if every edge is incident to at least one node in S.

Polynomial time reduction example

Definition

Given a graph $G = (V, E)$, a set of nodes $S \subseteq V$ is a *vertex cover* if every edge is incident to at least one node in S.

Definition

In the *Vertex Cover* problem, we are given a graph $G = (V, E)$ and an integer k, and answer whether there exists a vertex cover of size at most k.

Polynomial time reduction example

Definition

Given a graph $G = (V, E)$, a set of nodes $S \subseteq V$ is a *vertex cover* if every edge is incident to at least one node in S.

Definition

In the *Vertex Cover* problem, we are given a graph $G = (V, E)$ and an integer k, and answer whether there exists a vertex cover of size at most k.

Proposition

Independent Set $\leq_P$ *Vertex Cover.*
Vertex Cover $\leq_P$ *Independent Set.*

The classes P and NP

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

The classes P and NP

- The class P is the set of all decision problems that can be solved in polynomial time.
- The class NP (standing for *nondeterministic polynomial time*): a decision problem A is in NP if there exists a *polynomial-time verifier* algorithm V for the following task: 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; on the other hand, if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.

The classes P and NP

- The class P is the set of all decision problems that can be solved in polynomial time.
- The class NP (standing for *nondeterministic polynomial time*): a decision problem A is in NP if there exists a *polynomial-time verifier* algorithm V for the following task: 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; on the other hand, if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.
- $\mathrm{P} \subseteq \mathrm{NP}$.

The classes P and NP

- The class P is the set of all decision problems that can be solved in polynomial time.
- The class NP (standing for *nondeterministic polynomial time*): a decision problem A is in NP if there exists a *polynomial-time verifier* algorithm V for the following task: 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; on the other hand, if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.
- $\mathrm{P} \subseteq \mathrm{NP}$.
- Example: Independent Set $\in$ NP.

The classes P and NP

- The class P is the set of all decision problems that can be solved in polynomial time.
- The class NP (standing for *nondeterministic polynomial time*): a decision problem A is in NP if there exists a *polynomial-time verifier* algorithm V for the following task: 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; on the other hand, if the answer to an instance a of A is NO, then V returns NO, no matter what certificate it is given.
- $\mathrm{P} \subseteq \mathrm{NP}$.
- Example: Independent Set $\in$ NP.
- One of the most famous questions in (theoretical) computer science: $\mathrm{NP} \subseteq \mathrm{P}$?

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 an "appropriate" 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 an "appropriate" reduction.
- For the class NP, the "appropriate" reductions are the polynomial-time reductions. A problem is NP-*complete* if it is in NP and if all other problems in NP can be reduced to it in polynomial time.

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 an "appropriate" reduction.
- For the class NP, the "appropriate" reductions are the polynomial-time reductions. A problem is NP-*complete* if it is in NP and if all other problems in NP can be reduced to it in polynomial time.

Theorem (Cook-Levin)

SAT is NP-*complete.*

Definition

In a *Boolean satisfiability (SAT)* problem, we are given a Boolean formula, and answer whether there exists an *interpretation* of the variables that makes the formula true. That is, we need to decide whether there is a way of assigning TRUE and FALSE to each variable so that the formula evaluates to TRUE.

First examples of NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is "AND's of OR's". The problem of 3-SAT is the problem of deciding the satisfiability of a boolean formula in CNF where each OR case involves at most 3 literals.

First examples of NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is "AND's of OR's". The problem of 3-SAT is the problem of deciding the satisfiability of a boolean formula in CNF where each OR case involves at most 3 literals.
- 3-SAT is NP-complete.

First examples of NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is "AND's of OR's". The problem of 3-SAT is the problem of deciding the satisfiability of a boolean formula in CNF where each OR case involves at most 3 literals.
- 3-SAT is NP-complete.
- Observation: Polynomial reduction is transitive, i.e., if $A \leq_{\mathrm{P}} B$, $B \leq_{\mathrm{P}} C$, then $A \leq_{\mathrm{P}} C$.

First examples of NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is "AND's of OR's". The problem of 3-SAT is the problem of deciding the satisfiability of a boolean formula in CNF where each OR case involves at most 3 literals.
- 3-SAT is NP-complete.
- Observation: Polynomial reduction is transitive, i.e., if $A \leq_{\mathrm{P}} B$, $B \leq_{\mathrm{P}} C$, then $A \leq_{\mathrm{P}} C$.

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.*

First examples of NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is "AND's of OR's". The problem of 3-SAT is the problem of deciding the satisfiability of a boolean formula in CNF where each OR case involves at most 3 literals.
- 3-SAT is NP-complete.
- Observation: Polynomial reduction is transitive, i.e., if $A \leq_{\mathrm{P}} B$, $B \leq_{\mathrm{P}} C$, then $A \leq_{\mathrm{P}} C$.

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.*

First examples of NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is "AND's of OR's". The problem of 3-SAT is the problem of deciding the satisfiability of a boolean formula in CNF where each OR case involves at most 3 literals.
- 3-SAT is NP-complete.
- Observation: Polynomial reduction is transitive, i.e., if $A \leq_P B$, $B \leq_P C$, then $A \leq_P C$.

Corollary

If a problem A is in NP*, and if there is any* NP*-complete problem B such that $B \leq_P A$, then A is* NP*-complete.*

Example

The Independent set problem is NP-complete.

Procedure to show NP completeness

Recall 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.

Procedure to show NP completeness

Recall Corollary: If a problem A is in NP, and if there is any NP-complete problem B such that $B \leq_P A$, then A is NP-complete.
Given a problem A, to show it is NP-complete, we show that

1. It is in NP, by constructing a polynomial-time verifier and polynomial-length certificates for TRUE instances;
2. There is an NP-complete problem B, such that $B \leq_P A$. To do this, we
   - Give a polynomial time algorithm φ which takes in an instance of B and outputs an instance of A, and
   - Show that an instance b of B has answer TRUE if and only if the instance $\varphi(b)$ has answer TRUE.
   - Argue that φ runs in polynomial time.