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).

- A problem is a *decision problem* if its answer is either yes or no.

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).

- A problem is a *decision problem* if its answer is either yes or no.
- A decision problem A is *polynomial-time reducible* to a decision problem B if there is a polynomial-time algorithm φ that takes any instance a of problem A and returns an instance of problem B, such that $\varphi(a)$ has the same answer as a; that is, the answer to $\varphi(a)$ is yes if and only if the answer to a is yes.
  - We denote this by $A \leq_{\mathrm{P}} B$.

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. We need to 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 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. We need to answer whether there exists an independent set of G of size at least k.

Recall: 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 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. We need to answer whether there exists an independent set of G of size at least k.

Recall: 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. We need to answer whether there exists a vertex cover of size at most k.

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*) is the set of decision problems for which a *polynomial-time verifier* algorithm V exists for the following the task: if the answer to an instance of the problem is yes, then there exists a *polynomial-length certificate*, such that V, when provided with both the instance and the certificate, will return yes; on the other hand, if the answer to an instance of the problem 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*) is the set of decision problems for which a *polynomial-time verifier* algorithm V exists for the following the task: if the answer to an instance of the problem is yes, then there exists a *polynomial-length certificate*, such that V, when provided with both the instance and the certificate, will return yes; on the other hand, if the answer to an instance of the problem is no, then V returns no, no matter what certificate it is given.
- Obviously, P $\subseteq$ 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*) is the set of decision problems for which a *polynomial-time verifier* algorithm V exists for the following the task: if the answer to an instance of the problem is yes, then there exists a *polynomial-length certificate*, such that V, when provided with both the instance and the certificate, will return yes; on the other hand, if the answer to an instance of the problem is no, then V returns no, no matter what certificate it is given.
- Obviously, P $\subseteq$ NP.
- The most famous question in (theoretical) computer science: NP $\subseteq$ 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 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 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 need to decide 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.

Classical NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is an AND or 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.

Classical NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is an AND or 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.

Classical NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is an AND or 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.
- Karp: Many other often-encountered combinatorial problems are NP-complete.

Classical NP-complete problems

- A boolean formula is in *conjunctive normal form (CNF)* if it is an AND or 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.
- Karp: Many other often-encountered combinatorial problems are NP-complete.

Example

The Independent set problem is NP-complete.

Classical NP-complete problems

Definition

In a directed graph, a *Hamiltonian path* is a path that visits each vertex exactly once. A *Hamiltonian cycle* is a cycle that visits each vertex exactly once. In the Hamiltonian cycle(path) problem, we are given a graph and need to decide whether there exists a Hamiltonian cycle(path).

Classical NP-complete problems

Definition

In a directed graph, a *Hamiltonian path* is a path that visits each vertex exactly once. A *Hamiltonian cycle* is a cycle that visits each vertex exactly once. In the Hamiltonian cycle(path) problem, we are given a graph and need to decide whether there exists a Hamiltonian cycle(path).

Example

Hamiltonian cycle problem $\leq_P$ Hamiltonian path problem.

Classical NP-complete problems

Definition

In a directed graph, a *Hamiltonian path* is a path that visits each vertex exactly once. A *Hamiltonian cycle* is a cycle that visits each vertex exactly once. In the Hamiltonian cycle(path) problem, we are given a graph and need to decide whether there exists a Hamiltonian cycle(path).

Example

Hamiltonian cycle problem $\leq_P$ Hamiltonian path problem.

Example

The Hamiltonian cycle problem is NP-complete.

Classical NP-complete problems

Definition

In a *Traveling Salesman Problem (TSP)*, we are given a directed graph with weights on each edge, and a bound D. We need to decide whether there is a *tour* of total weight at most D.

Example

Hamiltonian Cycle $\leq_P$ TSP. $\Rightarrow$ TSP is NP-complete.

Classical NP-complete problems

Definition

Given disjoint sets X, Y and Z, each of size n, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, the *3-Dimensional Matching* problem asks whether there exist a set of n triples in T so that each elemtn of $X \cup Y \cup Z$ is contained in exactly one of these triples.

Classical NP-complete problems

Definition

Given disjoint sets X, Y and Z, each of size n, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, the *3-Dimensional Matching* problem asks whether there exist a set of n triples in T so that each elemtn of $X \cup Y \cup Z$ is contained in exactly one of these triples.

Example

3-Dimensional Matching is NP-complete.

Classical NP-complete problems

Definition

Given disjoint sets X, Y and Z, each of size n, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, the *3-Dimensional Matching* problem asks whether there exist a set of n triples in T so that each elemtn of $X \cup Y \cup Z$ is contained in exactly one of these triples.

Example

3-Dimensional Matching is NP-complete.

Corollary

Intersection of Three Matroids is NP*-complete.*

Definition

A *coloring* of a graph is an assignment of colors to vertices, so that no two adjacent vertices share the same color. In graph theory, the minimal number of colors for which such an assignment is possible is called the *chromatic number*, often denoted as $\chi(G)$ for graph G.
The prolbem of *3-Coloring* asks, given a graph G, whether $\chi(G) \leq 3$, i.e., whether G can be colored using three colors.

Definition

A *coloring* of a graph is an assignment of colors to vertices, so that no two adjacent vertices share the same color. In graph theory, the minimal number of colors for which such an assignment is possible is called the *chromatic number*, often denoted as $\chi(G)$ for graph G.
The prolbem of *3-Coloring* asks, given a graph G, whether $\chi(G) \leq 3$, i.e., whether G can be colored using three colors.

Example

$\chi(G) \leq 2$ if and only if G is bipartite.

Definition

A *coloring* of a graph is an assignment of colors to vertices, so that no two adjacent vertices share the same color. In graph theory, the minimal number of colors for which such an assignment is possible is called the *chromatic number*, often denoted as $\chi(G)$ for graph G.
The prolbem of *3-Coloring* asks, given a graph G, whether $\chi(G) \leq 3$, i.e., whether G can be colored using three colors.

Example

$\chi(G) \leq 2$ if and only if G is bipartite.

Example

3-Coloring is NP-complete.

Definition

Given a set of integers $a_1, \ldots, a_n$ and a target K, the problem of *Subset Sum* asks whether there exists $S \subseteq [n]$ so that $\sum_{i \in S} a_i = K$, where $[n]$ denotes $\{1, 2, \ldots, n\}$.

Definition

Given a set of integers $a_1, \ldots, a_n$ and a target K, the problem of *Subset Sum* asks whether there exists $S \subseteq [n]$ so that $\sum_{i \in S} a_i = K$, where $[n]$ denotes $\{1, 2, \ldots, n\}$.

Example

Subset Sum is NP-complete.

Definition

Given a set of integers $a_1, \ldots, a_n$ and a target K, the problem of *Subset Sum* asks whether there exists $S \subseteq [n]$ so that $\sum_{i \in S} a_i = K$, where $[n]$ denotes $\{1, 2, \ldots, n\}$.

Example

Subset Sum is NP-complete.

Definition

Given a set of n items with weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$, the size of a knapsack B and a target value K, the *Knapsack Problem* asks whether one can fill the knapsack with items of total value at least K; that is, whether there is $S \subseteq [n]$, so that $\sum_{i \in S} w_i \leq B$, and $\sum_{i \in S} v_i \geq K$.

Definition

Given a set of integers $a_1, \ldots, a_n$ and a target K, the problem of *Subset Sum* asks whether there exists $S \subseteq [n]$ so that $\sum_{i \in S} a_i = K$, where $[n]$ denotes $\{1, 2, \ldots, n\}$.

Example

Subset Sum is NP-complete.

Definition

Given a set of n items with weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$, the size of a knapsack B and a target value K, the *Knapsack Problem* asks whether one can fill the knapsack with items of total value at least K; that is, whether there is $S \subseteq [n]$, so that $\sum_{i \in S} w_i \leq B$, and $\sum_{i \in S} v_i \geq K$.

Example

The Knapsack problem is NP-complete.

Categories of basic NP-complete problems

- Packing problems: Independent Set
- Covering problems: Vertex cover
- Sequencing problems: Hamiltonian Cycle, Hamiltonian Path, Traveling Salesman Problem
- Partitioning problems: 3-Dimensional Matching, 3-Coloring
- Numerical problems: Subset Sum, Knapsack

Don't forget about 3-SAT, often a very convenient starting point.