# Polynomial Time Reduction

Arnav Gupta

March 28, 2024

## Contents

## 1 Decision Problems

**Decision Problem**: given a problem instance $I$, answer a certain question yes or no

**Problem Instance**: input for the specified problem

**Problem Solution**: correct answer (yes or no) for the specified problem instance

- I is a yes-instance if the correct answer for the instance I is yes

- I is a no-instance if the correct answer for the instance I is no

**Size of a problem instance**: the number of bits required to specify (or encode) the instance I

## 2 Polynomial Time Reduction

**Polynomial Time Reduction**: a decision problem $A$ is polynomial time reducible to a decision problem $B$, if there is a polynomial time algorithm $F$ that transforms any instance $I_A$ of $A$ to an instance $I_B$ of $B$ such that $I_A$ is a yes-instance of $A$ if and only if $F(I_A) = I_B$ is a yes-instance of $B$.

$A \leq_P B$ means such a polynomial time reduction exists. $A =_P B$ means $A \leq_P B$ and $B \leq_P A$.

For decision problems $A$ and $B$, with a polynomial time algorithm to solve $B$ and a polynomial reduction $F$ which gives a polynomial time reduction from $A$ to $B$, $A$ can be solved in polynomial time by applying $F$ and using the algorithm to solve $B$. Further, the size of $F(I_A)$ must be polynomial since returning $F(I_A)$ fits in the polynomial-time reduction from $A$ to $B$.

### 2.1 Transitivity

$A \leq_P B$ and $B \leq_P C$ gives $A \leq_P C$

### 2.2 Proving Hardness

Given a problem A known to be impossible to be solved in polynomial time, \$A $\leq_\mathrm{P}$ B $\implies B$ cannot be solved in polynomial time.

$A \leq_P B$ means $B$ is at least as hard as $A$. In other words, if $B$ could be solved in polynomial time, $A$ could be solved in polynomial time.

### 2.3 Simple Reductions

The following 3 problems are equivalent in terms of polynomial-time solvability.

#### 2.3.1 Maximum Clique (Clique)

$S \subseteq V$ is a clique if $\{u, v\} \in E$ for all $u, v \in S$

**Input**: $G = (V, E)$ and an integer $k$

**Output**: (answer to the question) is there a clique in $G$ with at least $k$ vertices

### 2.3.2 Maximum Independent Set (IS)

$S \subseteq V$ is an independent set if $\{u, v\} \notin E$ for all $u, v \in S$

**Input**: $G = (V, E)$ and an integer $k$

**Output**: (answer to the question) is there an independent set in $G$ with at least $k$ vertices

### 2.3.3 Minimum Vertex Cover (VC)

$S \subseteq V$ is a vertex cover if $\{u, v\} \cap S \neq \emptyset$ for all $\{u, v\} \in E$

**Input**: $G = (V, E)$ and an integer $k$

**Output**: (answer to the question) is there a vertex cover in $G$ with at most $k$ vertices

### 2.3.4 Connections

To get Clique $\leq_P$ IS:

- Clique informally means find a set of size at least $k$ vertices with all edges in between
- IS informally means fnd a set of size at least $k$ vertices with no edges in between
- **idea**: change edges in the input to non edges (find complement of the graph, where an edge in the original means no edge in the new graph and vice versa)
- $F$ is the algorithm to construct the complement of the input $G$
- clearly this can be done in polynomial time and one can check that $S \subseteq V$ is a clique in $G$ if and only if $S$ is an independent set in $\bar{G}$
- hence, $\{G, k\}$ is a yes instance for Clique if and only if $\{\bar{G}, k\}$ is a yes instance for IS

**Lemma**: assume $G = (V, E)$ is a graph, then $S \subseteq V$ is a vertex cover if and only if $V - S$ is an independent set in $G$

- assume $S$ is a \$VC in $G$

  - if $V \backslash G$ is not an IS, then we have $x, y \in V \backslash S$ such that $\{x, y\} \in E$
  - by definition of a vertex cover, one of $x$ or $y$ (or both) must be in $S$, which is a contradiction

- let $V \setminus S$ be an IS
  - if $S$ is not a vertex cover, then there exists an edge $\{x, y\} \in E$ such that $x \notin S$ and $y \notin S$
  - this means that $x, y \in V \setminus S$ and there is an edge between them, which is a contradiction

This lemma gives that $G$ has a VC of size at most $k$ if and only if $G$ has an IS of size at least $n-k$. This means the reduction algorithm maps $\{G, k\}$ for VC to $\{G, n-k\}$ for IS. This runs in poly-time and maps yes/no instances for VC to yes/no instances for IS, which shows that

$$Clique =_P IS =_P VC$$

### 2.3.5 Hamiltonian Cycle (HC)

A cycle is a Hamiltonian Cycle if it touches every vertex exactly once.

**Input**: undirected graph $G = (V, E)$

**Output**: (answer to) does $G$ have a Hamiltonian Cycle?

### 2.3.6 Hamiltonian Path (HP)

A path is a Hamiltonian Path if it touches every vertex exactly once.

**Input**: undirected graph $G = (V, E)$

**Output**: (answer to) does $G$ have a Hamiltonian path?

### 2.3.7 HC-HP Connection

$HP =_P HC$ can be shown as follows.

First, $HP \leq_P HC$ will be shown.

- given $G = (V, E)$ for HP, it must be transformed to $G' = (V', E')$ for HC
- construct $G'$ in the following way:

    - add a vertex $s$ to $V$: $V' = V \cup \{s\}$
    - add edges $(s, x)$ for $x \in V$
- this reduction runs in poly-time
- it will be shown that $G$ has a Hamiltonian path if and only if $G$ has a Hamiltonian cycle
    - (forward) assume $P$ is a Hamiltonian path in $G$, with endpoints $u, w$, then $P + su + sw$ is a Hamiltonian cycle in $G'$
    - (backward) assume $C'$ is a Hamiltonian cycle in $G'$, then there must be 2 indicident edges on $s$ which can be removed from $C'$ to get $C$ which is a Hamiltonian path in $G$

Next, $HC \leq_P HP$ will be shown.

- given $G = (V, E)$ for HC, construct $G' = (V', E')$ in the following way:
    - choose an arbitrary vertex $x \in V$
    - add a duplicate $x'$ of $x$
    - add vertices $t, t'$ with degree 1 with edges $tx$ and $t'x'$
- this reduction runs in poly-time
- it will be shown that $G$ has a Hamiltonian cycle if and only if $G'$ has a Hamiltonian path
    - (forward) assume there is a Hamiltonian cycle $C$ in $G$, take some vertex $x$ with neighbours $u$ and $w$ on $C$, then $C - xu + x'u + x't' + xt$ is a Hamiltonian path
    - (backward) assume there is a Hamiltonian path in $G'$, it should have endpoints $t, t'$ in $G'$ which are adjacent to $x, x'$ respectively, then an edge can be added from some neighbour of $x'$ to $x$ (since $x'$ and $x$ have the same neighbours), which gives a Hamiltonian Cycle

### 2.3.8 3-SAT

For $X = \{x_1, \ldots, x_n\}$ where $x_i$ is a Boolean variable:

- a literal term is either $x_i$ or $\overline{x_i}$
- a clause is a disjunctio of distinct literals, which has length $\ell$

- an assignment satisfies a clause $C$ if it causes $C$ to evaluate to true
- a conjunction of a finite set of clauses is called a formula in Conjunctive Normal Form (CNF)

  **Input**: a CNF-formula in which each clause has at most 3 literals

  **Output**: (answer to) is there truth assignment to these variables that satisfies all the clauses?

#### 2.3.9 3-SAT and IS Connection

Convert each clause to a graph with edges between each term. Then place edges between negated terms and find an independent set of this graph.

To force exactly one choice from each clause, set $k$ to be the number of clauses.

**Proof**: First, consider the forward direction. If there is a satisfying assignment, then choose 1 literal that is set to true in each clause and the corresponding vertex will be in the independent set. Since the CNF is satisfiable, there is at least 1 true literal in each clause and so the set has exactly $k$ vertices.

The $k$ vertices form an independent set as there are no clause edges between them, since only one literal vertex in each clause. Further, no negation edges between chosen vertices, since this cannot be chosen in the satisfying assignment.

Next, consider the reverse direction. Assume there is an independent set of size $k$ in $G$. Any independent set can choose only 1 vertex from each clause, since there are edges within a clause. Only $k$ clauses, so an independent set of size $k$ chooses exactly 1 vertex from each clause. Further, for each variable, choose either the literal or its negation. This means the assignment satisfies the CNF.