Analysis of Algorithm

Reductions Among Classic NP-Complete Problems: 3SAT, MIS, MVC, and 3-Colorability

Instructor: Dr. Mudassir Shabbir

ITU

May 10, 2025

Scribe: Abdul Basit
Muhammad Okasha Khan

What is Reduction?

Definition

Reduction in algorithms is a technique where one problem is **translated into another problem** in such a way that a solution to the second problem can be used to solve the first one.

Reduction should be:

- Polynomial Time
- Translates yes instance to yes
- Translates no instance to no
- In $\rightarrow$ input of A, Out $\rightarrow$ input of B

As we did 3-SAT $\leq_p$ MIS

Reduction Direction Matters

Key Insight: The difficulty of a reduction depends on its direction.

- Sorting $\leq_p$ 3SAT is easy.
- 3SAT $\leq_p$ Sorting would imply P $=$ NP. (which is a big deal)

Computational Complexity Principle

If A $\leq_p$ B and B is in P, then A is in P.
But if A is NP-complete and B is in P, then $\mathbf{P = NP}$!

Commonly Used In:

- Algorithm Design
- Complexity Theory
- Problem Classification

Key Benefit

Helps break complex problems into simpler or already-solved problems.

Example: Sorting $\rightarrow$ Selection

- If you can repeatedly find the smallest element (selection)
- You can sort the entire list by:
  1. Select minimum element
  2. Remove it from list
  3. Repeat until list is sorted

Implementation

`SelectionSort` directly uses this reduction approach!

Reduction in Complexity Theory

Proving Problem Hardness

Used to compare difficulty of problems, especially for :

- NP-completeness proofs
- Undecidability proofs

Standard Proof Technique

To show **Problem A** is hard:

1. Take a known hard problem (e.g., SAT)
2. Reduce it to A in polynomial time
3. Show solution to A solves SAT
4. Conclude: A is at least as hard as SAT

Why Reduction Matters

Practical Benefits

- **Algorithm Reuse**: Leverage existing solutions
- **Optimal Design**: Build on proven strategies
- **Code Efficiency**: Avoid reinventing the wheel

Theoretical Importance

- **Hardness Proofs**: Classify problem difficulty
  Completeness: Identify fundamental problems
  Computability: Establish decidability limits

3-SAT Problem Definition

What is 3-SAT?

A special case of the Boolean satisfiability problem (SAT), and one of the most famous **NP-complete** problems.

Key Characteristics

- Boolean formula in **CNF** (Conjunctive Normal Form)
- Each clause has **exactly 3 literals**
- NP-complete (can verify solutions quickly, but no known efficient solution)

Decision Problem: "Is there a True/False assignment to variables that makes the entire formula evaluate to True?"

Formal Definition of 3-SAT

Mathematical Representation

Given a formula ϕ in 3-CNF form:

$$\phi = \bigwedge_{i=1}^{m}(l_{i1} \vee l_{i2} \vee l_{i3})$$

where:

- m = number of clauses
- Each l_{ij} is a **literal** (variable or its negation)
- $\wedge$ = AND (conjunction) $\vee$ = OR (disjunction)

Computational Challenge

The problem remains NP-complete even when:

- No clause contains duplicate literals
- Each variable appears in at most 3 clauses

3-SAT Example

Sample 3-CNF Formula

$$(x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4) \land (\neg x_3 \lor \neg x_4 \lor x_5)$$

Components

- **Variables:** x_1, x_2, x_3, x_4, x_5
- **Clauses:** 3
- **Literals:** 9 (3 per clause)

Satisfying Assignment

One possible solution:

- $x_1 = \text{True}$
- $x_2 = \text{False}$
- $x_3 = \text{True}$
- $x_4 = \text{True}$
- $x_5 = \text{True}$

Why 3-SAT Matters

Theoretical Significance

- **Canonical NP-complete problem**
- Used as starting point for thousands of NP-completeness proofs
- Fundamental in computational complexity theory

Practical Applications

- Circuit design verification
- AI planning problems
- Software verification
- Cryptography

3-SAT vs. General SAT

3-SAT

- Every clause has **exactly 3 literals**
- Still NP-complete
- Often easier to reduce to
- Standard benchmark problem

General SAT

- Clauses can have **any number** of literals
- First problem proven NP-complete (Cook-Levin)
- More general but often reduced to 3-SAT

Key Theorem

Any NP problem can be **polynomially reduced** to 3-SAT while preserving satisfiability.

3-SAT Example Walkthrough

Given Formula

$$(x \lor y \lor z) \land (\neg x \lor \neg y) \land (x \lor \neg y \lor z)$$

Variables: x, y, z
Clauses: 3 (corrected from original)

Verification Process:

Try assignment: $x = \text{True}, y = \text{False}, z = \text{True}$

1. $(T \lor F \lor T) \equiv T$
2. $(\neg T \lor \neg F) \equiv (F \lor T) \equiv T$
3. $(T \lor \neg F \lor T) \equiv (T \lor T \lor T) \equiv T$

Result: All clauses satisfied Formula is satisfiable

Note on Original Example:

The original formula contained a 4-literal clause $(y \lor x \lor y \lor z)$ which violates 3-SAT rules. This has been corrected.

Vertex Cover Problem

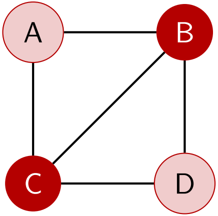
Definition

A **vertex cover** of a graph $G = (V, E)$ is a subset $C \subseteq V$ such that:

$$\forall (u, v) \in E,\ u \in C \ \vee \ v \in C$$

Decision Problem

Input: Graph G and integer k
Question: Does G have a vertex cover of size $\leq k$?



Minimum vertex cover: {B, C}

Vertex Cover Properties

Simple Algorithm

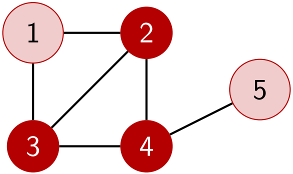
1. While edges remain:
   - Pick any edge (u, v)
   - Add both u and v to cover
   - Remove all edges incident to u or v
2. Output the selected vertices

Note

This greedy algorithm doesn't give minimal cover but guarantees $|C| \leq 2\times$ optimal

Vertex Cover Example

Graph Instance



- {2,3,4} (size 3)
- {1,3,4} (size 3)
- {2,3,5} (size 3)

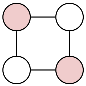
Types of Vertex Cover Problems

Decision Problem

Input:

- Graph $G = (V, E)$
- Integer k

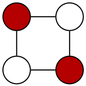
Question:
Does G have a vertex cover of size $\leq k$?



$k = 2$

Optimization Problem

Input: Graph $G = (V, E)$
Goal: Find the minimum vertex cover C^*



$|C^*| = 2$

Equivalence: The decision version is NP-complete $\iff$ The optimization version is NP-hard

Maximum Independent Set (MIS)

Definition

Given an undirected graph $G = (V, E)$, an **independent set** is a subset $S \subseteq V$ where:

$$\forall u, v \in S, \ (u, v) \notin E$$

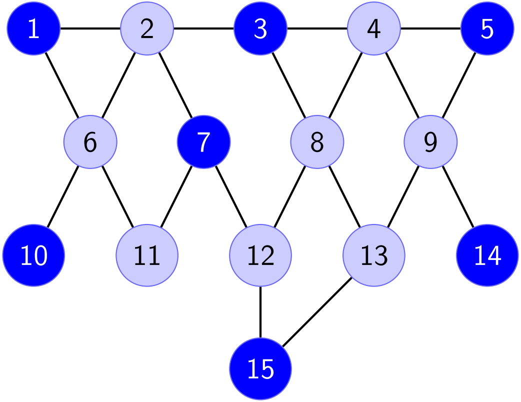
The Maximum Independent Set is the largest such S.

Decision Problem

Input: Graph G, integer k
Question: Does G have an independent set of size $\geq k$?

Optimization Problem

Find the independent set with maximum cardinality:
$S^* = \arg\max_{S \subseteq V} |S|$



One possible MIS: $\{1,3,5,7,10,14,15\}$

Reduction: MIS $\leq_p$ MVC

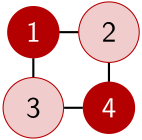
Objective

Transform any MIS instance into an MVC instance such that:

$$S \text{ is MIS of } G \iff V \setminus S \text{ is MVC of } G$$

Reduction Steps

1. Start with graph $G = (V, E)$ for MIS
2. Use **same graph** for MVC
3. Let $k = |V| - t$ (where t is target MIS size)
4. Solve MVC(G, k)
5. Return $V \setminus C$ as MIS



MIS: {1,4},{2,3}
MVC: {2,3},{1,4}

Proof of Correctness

Key Lemma:

For any graph $G = (V, E)$ and subset $S \subseteq V$:

$$S \text{ is independent set } \iff V \setminus S \text{ is vertex cover}$$

An independent set is a set of vertices with no two adjacent.
A vertex cover is a set of vertices that touches every edge (i.e. every edge has at least one endpoint in the set).

Proof:

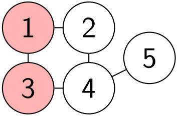
- ($\Rightarrow$) If S is independent, every edge has ≥ 1 endpoint in $V \setminus S$ (else S wouldn't be independent)
- ($\Leftarrow$) If $V \setminus S$ is vertex cover, no edge connects two vertices in S (else $V \setminus S$ wouldn't cover it)

Implications:

- Size relationship: $|S| = |V| - |C|$
- S is maximum independent set $\iff$ C is minimum vertex cover
- Reduction preserves optimality

Example Reduction

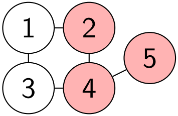
Original Graph G



MIS Instance:

- $V = \{1, 2, 3, 4, 5\}$
- Target size $t = 2$

Reduced MVC Instance



Parameters:

- Same graph G
- $k = |V| - t = 3$

Solution Mapping

- MVC solution: $\{2, 4, 5\}$ (size 3)
- MIS solution: $V \setminus \{2, 4, 5\} = \{1, 3\}$
- Verification: $\{1, 3\}$ is indeed an independent set

Complexity Implications

Reduction Properties

- **Time:** $O(1)$ (graph remains unchanged)
- **Space:** No additional memory needed
- **Approximation:** Preserves approximation ratios

Theoretical Consequences

- Since MIS is NP-hard, MVC must also be NP-hard
- MVC inherits inapproximability results from MIS
- Any algorithm for MVC can solve MIS via this reduction

Graph Colorability Problem

Definition A graph $G = (V, E)$ is k-**colorable** if there exists a function:

$$c : V \to \{1, 2, \ldots, k\}$$

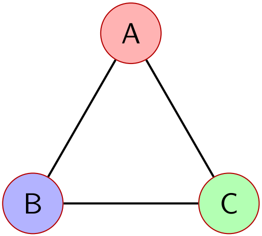
such that:

$$\forall (u, v) \in E,\ c(u) \neq c(v)$$

Decision Problem

Input: Graph G, integer k
Question: Is G k-colorable?

Optimization Version Find the chromatic number $\chi(G)$:
The smallest k for which G is k-colorable

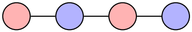


$\chi(G) = 3$ for this triangle graph

Colorability Examples

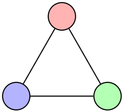
2-Colorable Graphs

- Bipartite graphs
- Trees
- Even-length cycles



Non-2-Colorable

- Odd-length cycles
- Complete graphs (K_n needs n colors)



Testing 2-Colorability

Equivalent to checking if the graph is **bipartite** - solvable in $O(|V| + |E|)$ time using BFS/DFS

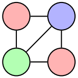
Graph Coloring Complexity

Theoretical Status

- **2-Coloring**: Polynomial time (bipartite check)
- **3-Coloring**: NP-complete (reducible from 3-SAT)
- k**-Coloring** $(k \geq 3)$: NP-complete
- **Approximation**: No constant factor approximation unless P=NP

Greedy Coloring Algorithm

1. Order vertices arbitrarily
2. For each vertex:
   - Assign smallest available color not used by neighbors
3. Uses $\leq \Delta + 1$ colors ($\Delta =$ maximum degree)



Brooks' Theorem:
Any connected graph is Δ-colorable, except complete graphs and odd cycles

Special Cases and Variants

Edge Coloring

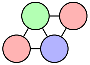
- Color edges instead of vertices
- No adjacent edges share color
- Vizing's Theorem: $\Delta \leq \chi'(G) \leq \Delta + 1$

List Coloring

- Each vertex has its own set of allowed colors
- More general than classic coloring

Perfect Graphs

- $\chi(G) = \omega(G)$ (clique number)
- Includes:
  - Bipartite graphs
  - interval graphs
  - Chordal graphs



Four Color Theorem:
Every planar graph is 4-colorable (famous mathematical result proved in 1976)

3-SAT $\leq_p$ 3-Coloring Reduction

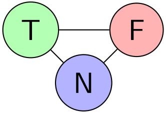
Objective

Given a 3-CNF formula Φ, construct graph G_Φ such that:

$$\Phi \text{ is satisfiable} \iff G_\Phi \text{ is 3-colorable}$$

Color Semantics:

- T (True)
- F (False)
- N (Base)



Color triangle

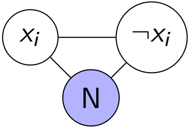
Key Components:

- Variable gadgets
- Clause gadgets
- Coloring constraints

Variable Gadget Construction

For each variable x_i

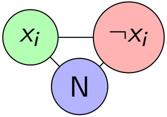
- Create triangle: x_i, $\neg x_i$, and common vertex N
- Forces x_i and $\neg x_i$ to be T/F (opposite colors)
- Base vertex enforces constraints



Coloring Implications

- If $x_i = \text{T}$, then $\neg x_i = \text{F}$
- If $x_i = \text{F}$, then $\neg x_i = \text{T}$

Example Assignment

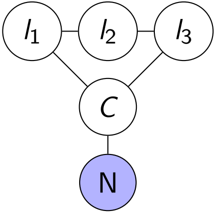


$x_i = \text{True}$

Clause Gadget Construction

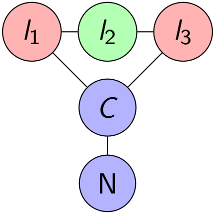
For each clause $(l_1 \lor l_2 \lor l_3)$

- Create triangle connecting literals to new vertex C
- Add edge to base vertex B
- Ensures at least one literal is T



Satisfaction Condition A clause gadget is properly colored iff at least one literal vertex is T

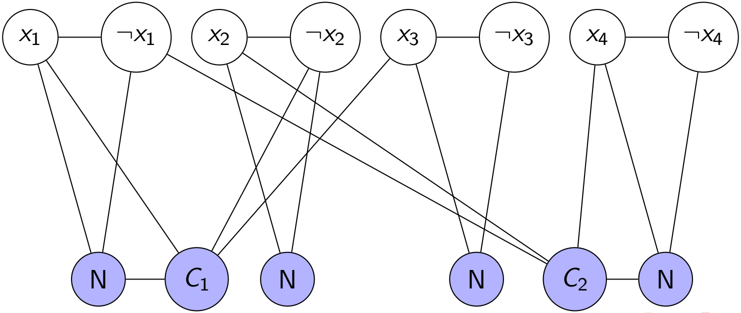
Valid Coloring: Clause satisfied by l_2



Full Construction Example

For formula $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

1. Build variable gadgets for x_1 to x_4
2. Connect literals to clause gadgets
3. Add base vertex connections



Correctness Proof

Key Observation

Any valid 3-coloring corresponds to a satisfying assignment, and vice versa.

($\Rightarrow$) Satisfiable $\implies$ 3-colorable

- Set True variables to T, False to F
- Color clauses using remaining color for C
- All constraints satisfied

($\Leftarrow$) 3-colorable $\implies$ Satisfiable

- Extract assignment from variable colors
- Each clause has at least one T literal
- Formula satisfied

Correctness Proof

Polynomial Time

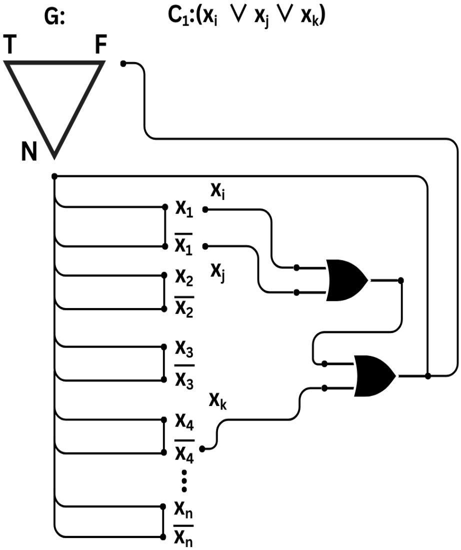
Construction is $O(n + m)$ where:

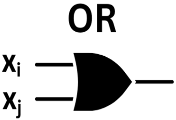
- $n =$ number of variables
- $m =$ number of clauses

NP-Completeness

Proves 3-Coloring is NP-complete via reduction from 3-SAT

Dr. Mudassir Shabbir's Figure (3-SAT $\leq_p$ 3-Coloring)





T	T	T
T	F	T
F	T	T
F	F	F