Exploration 8.4 - NP-Completeness Proof Examples

Introduction

This section will examine several examples that illustrate how to establish a problem as NP-Complete. These examples will uncover the core principles of NP-Completeness and outline the proof steps.

Proof for NP-Completeness (Reading Time: 1 min)

To show that a problem X is NP-Complete we must demonstrate two key properties:

Step 1: Show that X is in NP

> This means that given a solution to X, we can verify its correctness in polynomial time.

Step 2: Show that X is NP-Hard

> This means X is at least as hard as every other problem in NP.
> To prove this, we must reduce a known NP-Complete problem (A) to X in polynomial time.

Let us understand these steps with examples.

Proof for NP-Completeness : 3-SAT Problem (Reading Time: 10 min)

Problem definition of 3-SAT problem.

Given a Boolean formula in Conjunctive Normal Form (CNF), where each clause contains exactly three literals, determine if there is an assignment to the variables that makes the formula true.

Step 1: Proving 3-SAT is in NP

If we are given a specific truth assignment to the variables, we can verify whether the formula is

satisfied in polynomial time by checking each clause. Each clause contains only three literals, allowing us to evaluate the formula and determine if the truth assignment satisfies all the clauses. This verification process can be completed in polynomial time, which means that 3-SAT is in NP.

Step 2: Proving 3SAT is NP-hard

To demonstrate that 3-SAT is NP-hard, we simply need to show that every NP problem can be converted to it in polynomial time. We can use the Boolean satisfiability problem (SAT), which is already established as NP-complete, and perform a reduction from SAT to 3-SAT.

Reduction from SAT to 3-SAT:

In SAT, a clause can contain any number of literals. However, in 3-SAT, each clause must consist of exactly three literals. To convert SAT into 3-SAT, we can transform clauses with more than three literals into multiple clauses, each containing exactly three literals. For clauses with fewer than three literals, we can add additional literals to ensure they also have exactly three literals.

Steps for reduction:

- If a clause has exactly 3 literals, No transformation is needed.
- If a clause has 2 literals (say $x \vee y$): We introduce a new variable, z, and create two new clauses:

  $$(x \vee y \vee z)$$
  $$(x \vee y \vee \neg z)$$

  This new variable, z, does not alter the value of the original formula, as we are also adding the negation of z. You can verify this by drawing a truth table.

- If a clause has 1 literal (say x): We introduce two new variables, y and z, and create four new clauses.

  $$(x \vee y \vee z)$$
  $$(x \vee y \vee \neg z)$$
  $$(x \vee \neg y \vee z)$$
  $$(x \vee \neg y \vee \neg z)$$

- If a clause has more than 3 literals, we break it into multiple 3-literal clauses by introducing new variables. For example, for a clause like $x \vee y \vee z \vee w$), we introduce a new variable v and

split it into:

$$(x \vee y \vee v),$$
$$(\neg v \vee z \vee w)$$

The slide deck linked here (https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/SAT_to_threeSAT.html) explains how these transformations are correct in detail.

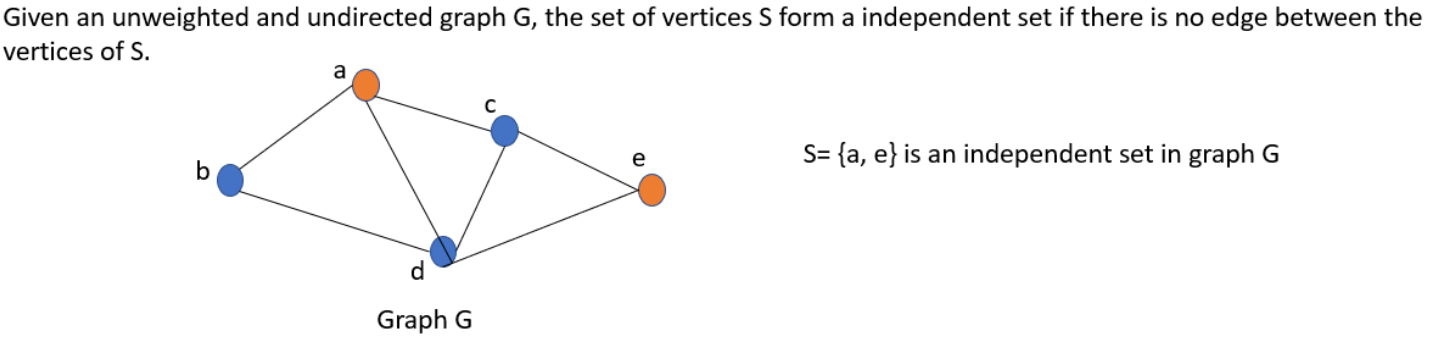
Hence, any instance of SAT can be transformed into an instance of 3-SAT while preserving satisfiability.

In conclusion, since 3-SAT is both in NP and NP-Hard, it is NP-Complete.

Click to reveal the video detailing this section.

Proof for NP-Completeness: Independent Set Problem (Reading Time: 12 min)

Independent Set Problem: An independent set in a graph G=(V,E) is a subset of vertices S⊆V such that no two vertices in S are adjacent (i.e., there is no edge between any two vertices in S).



Decision version of this problem: Given a graph G and an integer k, does there exist an independent set of size at least k?

Step 1: Proving Independent Set Problem is in NP

To demonstrate that the Independent Set problem is in NP, we need to show that a given solution (a subset of vertices) can be verified efficiently in polynomial time.

Given a subset of k vertices, we can determine in polynomial time whether there are any edges connecting them.

If the graph is represented using an adjacency matrix (an n×n matrix where M[u][v] = 1 if there is an edge (u, v) $\in$ E, and M[u][v] = 0 otherwise), then checking whether an edge exists between vertices u and v takes O(1) time. Since there are at most k^2 pairs of vertices to check in the subset, the overall verification process takes O(k^2) time.

Therefore, we can conclude that the Independent Set problem is in NP.

Step 2: Proving Independent Set Problem is NP-hard

To demonstrate that the Independent Set problem is NP-hard, we will reduce a known NP-complete problem to it. We can use the 3-SAT problem for this reduction.

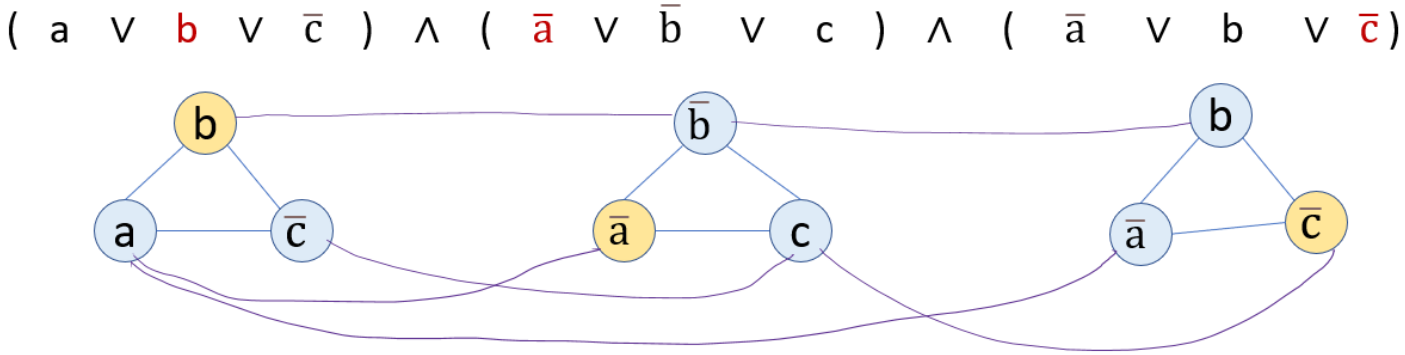
Reduction from 3-SAT to Independent Set:

Given a 3-SAT formula with k clauses and n variables, we construct a graph with the following components:

1. Clause Representation: Each clause in the 3-SAT formula is represented by three vertices, each corresponding to one of the three literals in the clause.
2. Intra-Clause Connections: The three literals within the same clause are connected to form a triangle (a fully connected subgraph); now each literal in the triangle becomes a vertex.
3. Inter-Clause Conflicting Edges: An edge is added between two literals (vertices) if they correspond to conflicting literals (for example, x_i and $\neg x_i$).

The goal is to find an independent set of size k (one vertex per clause). By selecting one vertex from each clause, we ensure that there is a satisfying assignment for the 3-SAT formula.

Reduction - General Idea to convert 3SAT to a graph :
To force a condition to chose only either a literal or it's negation we can draw a connected edge between the literals and their negations between different triangles.



Finding an Independent Set of Size k

- An independent set is a set of vertices where no two vertices are adjacent (i.e., no edges exist between them).
- The goal is to find an independent set of size k (one vertex per clause).
- Selecting one vertex from each clause ensures that the 3-SAT formula has a satisfying assignment because we are choosing non-conflicting literals

Since this transformation is done in polynomial time, and the existence of an independent set of size k implies the satisfiability of the original 3-SAT formula, we conclude that the independent Set is at least as hard as 3-SAT (i.e., it is NP-Hard).

The key takeaway is understanding the process of proving NP-completeness rather than memorizing specific proofs.

Each problem's proof is built upon prior proofs, establishing a chain of reductions that interconnects various NP-complete problems. The graph below illustrates a list of NP-complete problems that have been proven with the assistance of other known NP-hard problems. Some of these, such as the Knapsack problem, may be familiar to you. There is no requirement to memorize this graph; it merely demonstrates the interdependence of each problem's proof on others. SAT is referred to as Circuit SAT in the image below.

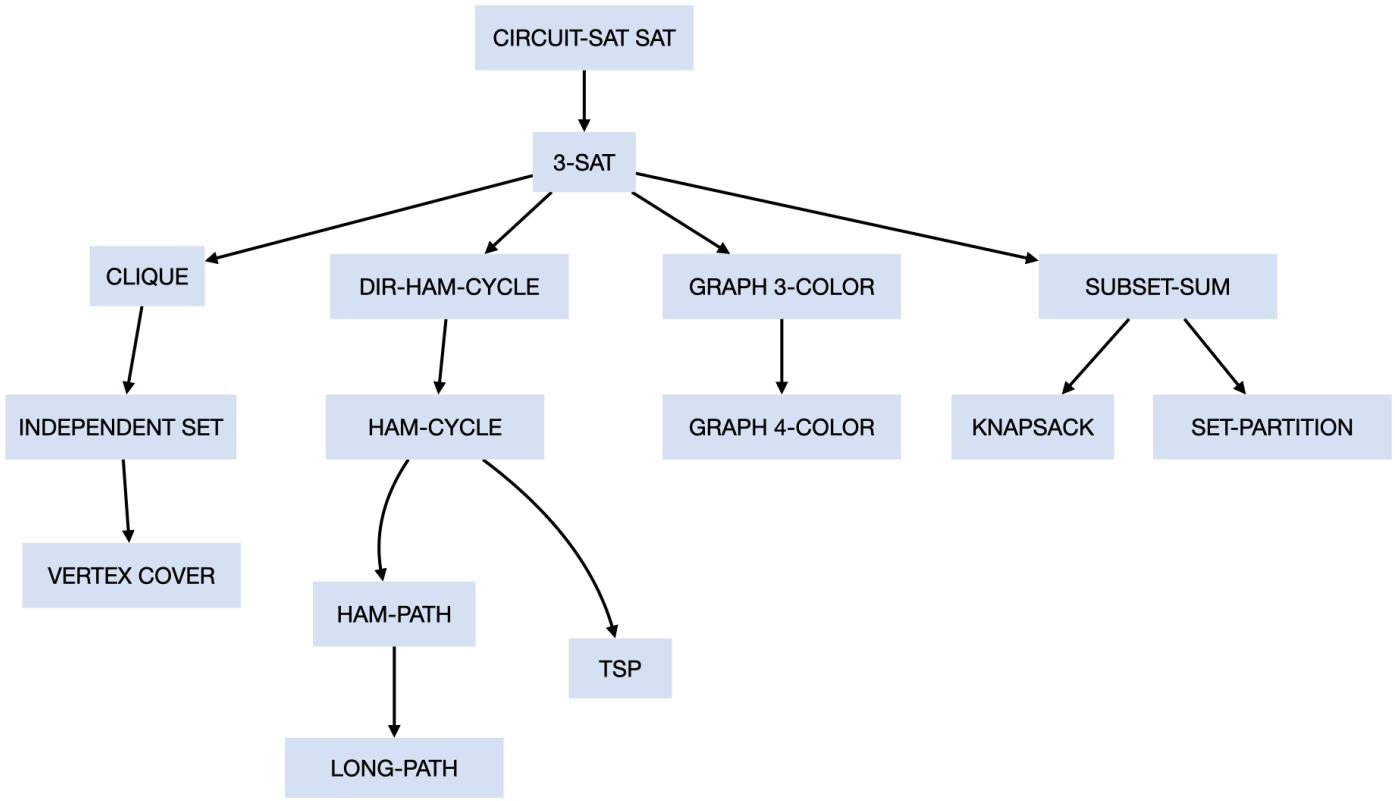


Image credits: Taken from Julianne Schutfort's lecture material.

Click to reveal the video detailing this section.

Exercises

Which transformation is used to reduce SAT to 3-SAT?

- ○ Splitting larger clauses into multiple clauses with at most three literals
- ○ Adding new variables to increase the number of literals
- ○ Removing literals to simplify clauses
- ○ None of the above

Check

Optional Additional Resources

If this topic interests you, then you might find the following optional resources useful.

- Algorithms by Jeff Erickson book, Chapter 12.
- Algorithms by Jeff Erickson book, Chapter 12, section 12.7 covers Independent Set Problem