CSE 5311: Design and Analysis of Algorithms

Intractability

Alex Dillhoff

University of Texas at Arlington

Introduction

Most of the algorithms discussed in a typical algorithms course run in polynomial time.

This focus is reasonable since algorithms that run worse than polynomial time have little practical use.

$n^{O(1)}$ n^k

Introduction

To simplify this notion: a problem for which a polynomial-time algorithm exists is "easy" and a problem for which no polynomial-time algorithm exists is "hard".

Knowing how to determine whether a problem is easy or hard is extremely useful.

If one can identify a hard problem, then an approximate solution may be the best that can be achieved.

P, NP, and NP-Complete

We start with four classes of algorithms:

1. Polynomial-time
2. NP (nondeterministic polynomial time)
3. NP-complete
4. NP-Hard

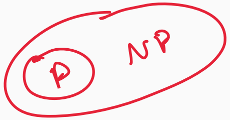
P, NP, and NP-Complete

$n^{O(1)}$

Problems in P are those solvable in polynomial time.

This means **any** constant k such that the running time is $O(n^k)$.

P, NP, and NP-Complete



The class NP is a superset of P: these are problems that can be **verified** in polynomial time.

This means that if someone gives you a solution to the problem, you can verify that it is correct in polynomial time.

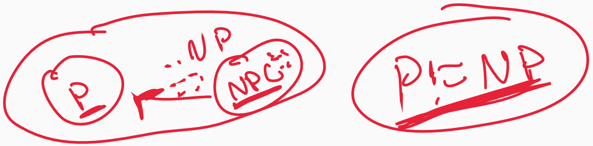
P, NP, and NP-Complete

This is different from solving the problem in polynomial time.

Problems in NP can be solved in **nondeterministic** polynomial time.

Such a model of computation does not exist in the real world.

P, NP, and NP-Complete



NP-Complete problems are problems in NP that are as **hard** as any other problem in NP.

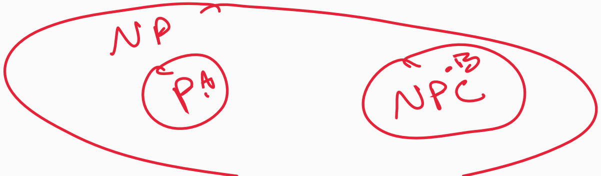
This means that if you can solve an NP-Complete problem in polynomial time, you can solve any problem in NP in polynomial time.

NP-Hard Problems

→P – poly time solution
NP – poly time verifier
NP-Hard –
NPC –

NP-Hard problems cannot be solved in polynomial time, nor can they be verified in polynomial times.

This class contains the problems that are not decision-based, including certain optimization problems and the classic Halting problem. $A \leq_p B$



Verifying a Solution

As long as we can come up with a verification algorithm for a problem in polynomial time, we can say that the problem is in NP

This is true even if we later find a polynomial-time algorithm for *solving* the problem.

Proving that a problem is NP-Complete

Proving that a problem belongs to either NP or NPC is difficult the first time you do it.

Luckily, now that problems have been proven to be NP-Complete, you can use these problems to prove that other problems are NP-Complete.

Proving that a problem is NP-Complete

Informally, a problem X is NP-Hard if it is at least as hard as any problem in NP.

If we can reduce every problem $Y \in NP$ to X in polynomial time, then X is NP-Hard.

2.

If X is also in NP, then X is NP-Complete.

1.

Optimization vs. Decision Problems

Many problems are framed as optimization problems.

Given some criteria, the goal is to find the best solution according to that criteria.

For the shortest path problem, the algorithm finds a path between two vertices in the fewest number of edges.

Optimization vs. Decision Problems

Decision problems are often easier to come up with than optimization problems.

If one can provide that a $\underline{\text{decision}}$ problem is hard, then its $\underline{\text{optimization}}$ problem is also hard.

Reductions

Reductions

- **Key idea:** Show the problem is NP-Complete.
- First proof was done by Cook in 1971.
- **Cook's Theorem:** $\boxed{\text{SAT}}$ is NP-Complete.
- Other problems can be shown to be NP-Complete by reducing them to SAT.

Reductions

Lemma 34.8

If L is a language such that $L' \leq_p L$ for some $L' \in \text{NPC}$, then L is NP-hard. If, in addition, we have $\boxed{L \in \text{NP}}$, then L is NP-Complete.

$x_1 = 1$

$x_2 = 1$

n'

literals

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

OR

AND

NOT

Circuit Satisfiability (CIRCUIT-SAT)

Given a boolean combinatorial circuit, is it satisfiable.

That is, is there an assignment of values to the inputs that makes the output true?

This problem is NP-Complete.

Circuit Satisfiability (CIRCUIT-SAT)

Exercise 34.3-1: Verify that the given circuit is unsatisfiable.

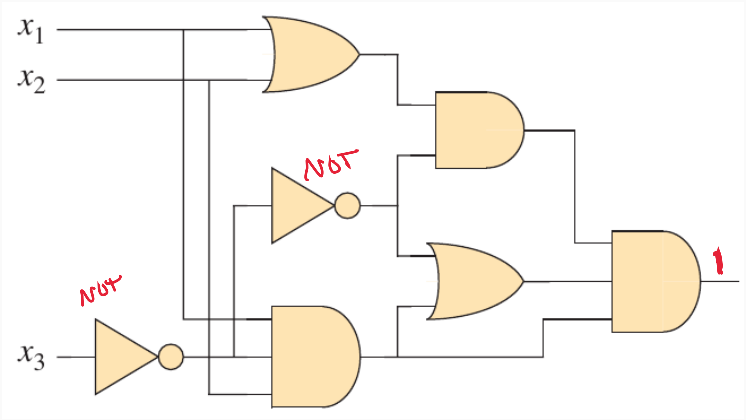
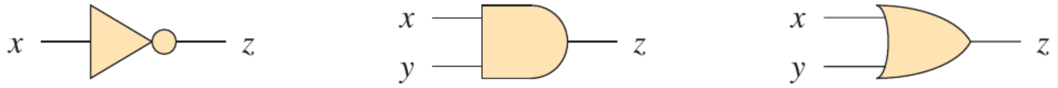


Figure 34.8 from (Cormen et al.)

Circuit Satisfiability (CIRCUIT-SAT)



x	$\neg x$
0	1
1	0

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Gate definitions (Cormen et al.)

Circuit Satisfiability (CIRCUIT-SAT)

How can we prove this circuit is unsatisfiable?

Circuit Satisfiability (CIRCUIT-SAT)

How can we prove this circuit is unsatisfiable?

The easiest way to do this is to code it up and brute force it.

Circuit Satisfiability (CIRCUIT-SAT)

The fact that we can verify this in polynomial time means that this problem is in NP.

The full proof for completeness is in the book.

Formula Satisfiability (SAT)

An instance of the **formula satisfiability (SAT)** problem is a boolean formula ϕ with

- n variables $x_1, x_2, \ldots, x_n$
- m clauses $C_1, C_2, \ldots, C_m$

$$\underset{C_1}{(x_1 \vee x_2)} \wedge \underset{C_2}{(x_3 \wedge \neg x_4)}$$

Formula Satisfiability (SAT)

For example, the formula

$$\phi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee x_3)$$

has the satisfying assignment $x_1 = 1, x_2 = 0, x_3 = 1$.

Formula Satisfiability (SAT)

SAT belongs to NP

Showing that SAT is in NP is straightforward.

Given a boolean formula ϕ and an assignment of values to the variables, one can verify that the formula is satisfied in polynomial time.

This is enough to show that SAT is in NP.

Formula Satisfiability (SAT) Basic Proof

SAT is NP-Complete

Turing

- For any problem in NP, there exists some machine that can verify solutions in polynomial time.
- Cook's proof involves simulating this machine with a Boolean formula.
- Encode the computation of the machine as a sequence of logical operations that can be expressed as a Boolean formula.
- Since the boolean formula is satisfiable if and only if the machine accepts the input, the satisfiability problem is $\boxed{\text{NP-Complete.}}$

$\boxed{SAT \geq_p L}$ $P = NP$

Formula Satisfiability (SAT)

To demonstrate the basic idea of reduction, we will show that CIRCUIT-SAT is reducible to SAT.

Formula Satisfiability (SAT)

CIRCUIT-SAT $\leq_p$ SAT

If we can reduce an instance of CIRCUIT-SAT, which is known to be NP-Complete, to SAT, then SAT is also NP-Complete.

If there existed a polynomial-time solution for SAT, then there would also exist a polynomial-time solution for CIRCUIT-SAT.

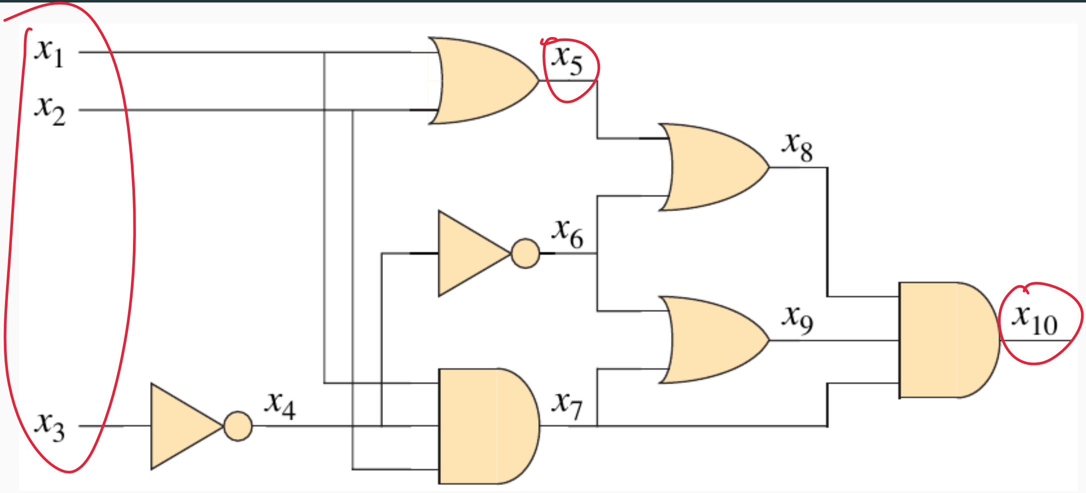
Since CIRCUIT-SAT is NP-Complete, this is not possible.

Formula Satisfiability (SAT)

CIRCUIT-SAT $\leq_p$ **SAT**

The reduction starts by introducing a variable for each wire and a clause for each gate.

Formula Satisfiability (SAT)



Circuit to formula reduction (Cormen et al.)

Formula Satisfiability (SAT)

The reduction algorithm produces a formula for each gate in terms of an "if and only if" statement.

$$
\begin{aligned}
\phi = x_{10} &\wedge (x_4 \leftrightarrow \neg x_3) \\
&\wedge \underline{(x_5 \leftrightarrow (x_1 \vee x_2))} \\
&\wedge \underline{(x_6 \leftrightarrow \neg x_4)} \\
&\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\
&\wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\
&\wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\
&\wedge (x_{10} \leftrightarrow (x_6 \wedge x_8 \wedge x_9))
\end{aligned}
$$

Formula Satisfiability (SAT)

A simpler explanation for this reduction is that a circuit can be represented as a boolean formula.

This formula can be solved by the SAT algorithm.

3SAT

The 3SAT problem is a special case of SAT where each clause has exactly three literals.

1. Show that $3SAT \in NP$
2. $SAT \leq_p 3SAT$

This problem is also NP-Complete.

Many problems can be "easily" reduced to 3SAT, which is why it is so important.

3SAT

An instance of 3SAT is a boolean formula ϕ with

- n **literals** $x_1, x_2, \ldots, x_n$
- m **clauses** $C_1, C_2, \ldots, C_m$

3SAT

Each clause has exactly three literals and is in **conjunctive normal form** (CNF), which means it is expressed as an AND of clauses. For example, the formula

$$\phi = \underbrace{(x_1 \vee \overset{\downarrow}{x_2} \vee x_3)}_{C_1} \wedge \underbrace{(\neg x_1 \vee \overset{\downarrow}{x_2} \vee x_3)}_{C_2} \wedge \underbrace{(x_1 \vee \overset{\downarrow}{\neg x_2} \vee x_3)}_{C_3}$$

is a 3SAT formula.

3SAT

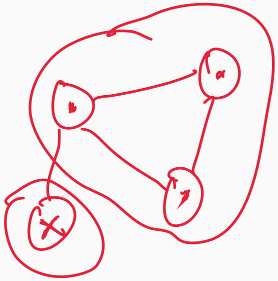
3SAT is NP-Complete

The 3SAT problem is NP-Complete.

This can be shown by reducing $\boxed{\text{SAT to 3SAT.}}$

A thorough proof is provided in the textbook.

Clique Problem



Clique Problem

A clique is a **complete subgraph** of an undirected graph G.

That is, a clique is a set of vertices such that every pair of vertices is connected by an edge.

The **clique problem** is to find the largest clique in a graph.

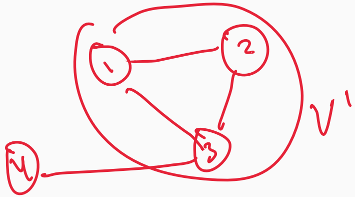
$3SAT \leq_p$ Clique

Clique Problem

$$\text{CLIQUE} = \{\langle \underset{\uparrow}{G}, \underset{\uparrow}{k} \rangle \mid G \text{ has a clique of size } k\}$$

Clique is in NP

n



Let's say you have access to a clique of size k.

You can verify that this is a clique in polynomial time by checking that every pair of vertices is connected by an edge.

$\downarrow$

For each pair $u, v \in V'$, the edge (u, v) is in E, where V' is the set of vertices in the clique.

Clique is NP-Complete

NPC

Knowing that 3SAT is NP-Complete, we can reduce 3SAT to the clique problem.

The reduction may not be intuitive as a boolean formula seems to have no relation to a graph.

Clique is NP-Complete

Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be a 3SAT formula with n variables.

We construct a graph G with $3n$ vertices.

Each C_r has three literals, l_1^r, l_2^r, l_3^r.

Clique is NP-Complete

To construct the graph, we create a triplet of vertices v_1^r, v_2^r, v_3^r for each clause C_r such that there is no edge connecting any two vertices in the same triplet.

There is an edge $(v_i^r, v_j^s) \in E$ if

1. v_i^r and v_j^s are in different triplets
2. l_i^r and l_j^s are not negations of each other

Clique is NP-Complete

One such formula that we can convert is

$$\phi = (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor x_3).$$

Clique is NP-Complete

If a satisfying assignment exists for ϕ, then each C_r has at least one literal that is true.

Consider the corresponding vertices in the graph for a satisfying assignment.

Since there is at least one true literal in each clause, there is at least one edge between the corresponding vertices.

Thus, a reduction from 3SAT to the clique problem is possible.

Clique is NP-Complete

How does this show that Clique is NP-Complete?

Clique is NP-Complete

How does this show that Clique is NP-Complete?

It is true that this example shows a very specialized graph.

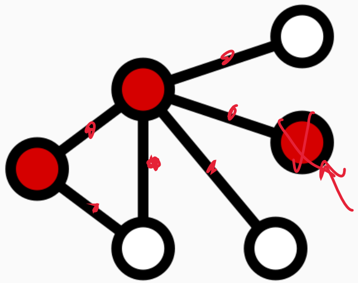
If there were a polynomial time solution for Clique on a general graph G, then surely it would work for a specialized graph as well.

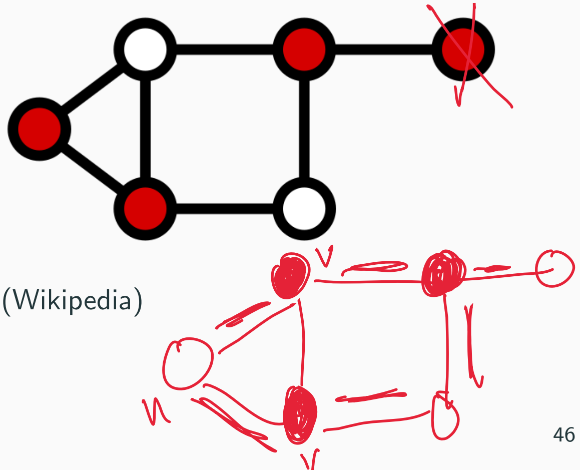
Clique is NP-Complete

If it could solve this one, then the corresponding 3SAT formula would be solvable as well.

This is a contradiction, so the Clique problem is NP-Complete.

Vertex Cover Problem





Vertex cover (Wikipedia)

Vertex Cover Problem

The **vertex cover problem** is to find the smallest set of vertices such that every edge in the graph is incident to at least one vertex in the set.

More formally, a vertex cover of a graph G is a set $V' \subseteq V$ such that for every edge $(u, v) \in E$, either $u \in V'$ or $v \in V'$.

Vertex Cover is in NP

Given a set of vertices V', one can verify that it is a vertex cover in polynomial time by checking that every edge is incident to at least one vertex in the set.

This is a polynomial-time verification algorithm, so the vertex cover problem is in NP.

Vertex Cover is NP-Complete

We can show that the vertex cover problem is NP-Complete by reducing it to an instance of the clique problem.

For this, we need to introduce the definition of a graph complement.

Given a graph $G = (V, E)$, the **complement** of G is the graph $\overline{G} = (V, E')$ where $E' = \{(u, v) \mid u, v \in V \text{ and } (u, v) \notin E\}$.

Vertex Cover is NP-Complete

Let G contain a clique $V' \subseteq V$, where $|V'| = k$.

Then $V - V'$ is a vertex cover of $\overline{G}$.

If $(u, v) \in \overline{E}$, but is not in E, then at least one of u or v is not in V'.