Lecture 7: NP-completeness and Reductions

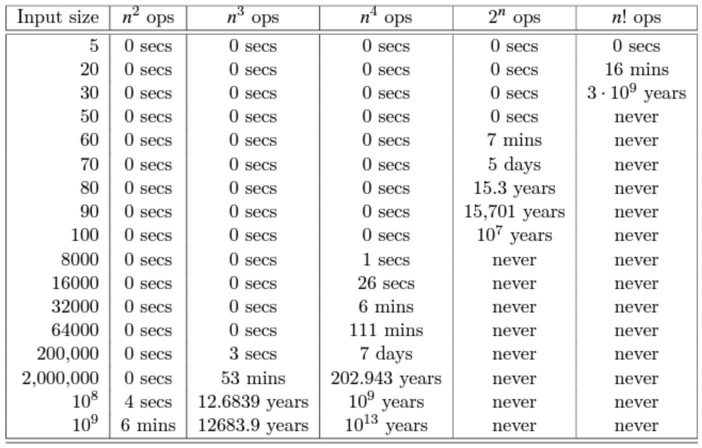
Erin McGowan

October 19th, 2023

Contents

1 Introduction

Consider the table of approximate time complexity with respect to input size from Har-Peled's lecture on NP Completeness [1] below.



Input size	n^2 ops	n^3 ops	n^4 ops	2^n ops	$n!$ ops
5	0 secs	0 secs	0 secs	0 secs	0 secs
20	0 secs	0 secs	0 secs	0 secs	16 mins
30	0 secs	0 secs	0 secs	0 secs	$3 \cdot 10^9$ years
50	0 secs	0 secs	0 secs	0 secs	never
60	0 secs	0 secs	0 secs	7 mins	never
70	0 secs	0 secs	0 secs	5 days	never
80	0 secs	0 secs	0 secs	15.3 years	never
90	0 secs	0 secs	0 secs	15,701 years	never
100	0 secs	0 secs	0 secs	10^7 years	never
8000	0 secs	0 secs	1 secs	never	never
16000	0 secs	0 secs	26 secs	never	never
32000	0 secs	0 secs	6 mins	never	never
64000	0 secs	0 secs	111 mins	never	never
200,000	0 secs	3 secs	7 days	never	never
2,000,000	0 secs	53 mins	202.943 years	never	never
10^8	4 secs	12.6839 years	10^9 years	never	never
10^9	6 mins	12683.9 years	10^{13} years	never	never

Figure 1.1: Running time as function of input size. Algorithms with exponential running times can handle only relatively small inputs. We assume here that the computer can do $2.5 \cdot 10^{15}$ operations per second, and the functions are the exact number of operations performed. Remember – never is a long time to wait for a computation to be completed.

We use these asymptotic time complexity estimates to divide problems into "easy" problems and "not easy" problems. We say a problem is "easy" if it has a **polynomial** time algorithm to solve it. That is, a problem is easy if there exists some algorithm that solves all instances of said problem in $O(n^c)$, where $|n|$ = problem size and c is a constant.

2 Decision Problems

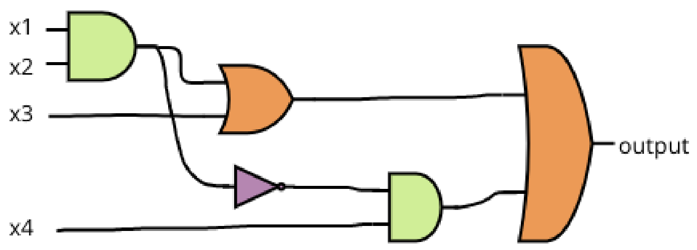
Today we focus on decision problems. A **decision problem** is a problem for which the possible answers are always a binary (e.g. yes/no, true/false, 1/0). A notable example of a decision problem is CIRCUIT-SAT, described below.

2.1 CIRCUIT-SAT

The CIRCUIT-SAT problem asks the following question:

Given a circuit with inputs $x_1, x_2, ..., x_n$ and gates, does there exist an assignment of values to the inputs such that the output is 1?

For instance, suppose we are given the circuit below with inputs x_1, x_2, x_3, and x_4. The green gates are "and" ($\wedge$) gates, orange gates are "or" ($\vee$) gates, and purple gates are "not" ($>$) gates.



We see that CIRCUIT-SAT is a decision problem (as its answer is always either "yes" or "no").

How fast can we solve this problem? The obvious solution is to check all 2^n possible truth statements, resulting in a runtime of approximately 2^n. Moreover, working backwards from the output cannot guarantee a faster runtime, as we may end up with many branches of possible solutions to check. However, given the proper extra information (the values of the inputs), it is easy to check whether the answer to the CIRCUIT-SAT problem is "yes" (however this does not mean it is easy to check if the answer is "no" - CIRCUIT-SAT is *not* symmetric).

3 P and NP

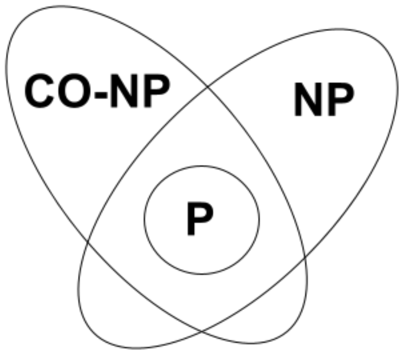
We define two notable sets of decision problems as follows:

- **P** is the class of all polynomial-time *solvable* decision problems.
- **NP** is the class of all polynomial-time *verifiable* decision problems.

We note that "P" stands for "polynomial" and "NP" stands for "non-deterministic polynomial."

We know that P is a subset of NP, but we do not know if it is a *proper* subset of NP. Most computer scientists believe that P $\neq$ NP (that is, that P is a proper subset of NP), however this is an open question.

If NP $\neq$ P, then



where "CO-NP" is akin to showing that the answer to a decision problem is "no" rather than "yes." Whether P is equal to the intersection of NP and CO-NP is another open question.

A problem X is **NP-hard** if you can solve X in polynomial time (showing P = NP). X does not have to be a decision problem to be NP-hard. We know CIRCUIT-SAT is NP-hard (we are not expected to be able to prove this at this time).

Since CIRCUIT-SAT is in NP, it follows that if we could prove that CIRCUIT-SAT is also in P, then we would know that P = NP. **All NP problems can be reduced to a version of CIRCUIT-SAT.**

4 Reductions

We say a problem X is **NP-complete** if X is both NP-hard and $X \in NP$.

We can prove a problem is NP-complete by creating a **reduction**, or mapping, from a known NP-complete problem to said problem (note the order of this - we must reduce the KNOWN problem to the new problem). For instance, suppose we have decision problems A and B. We define a transformation

$$f : I \to I'$$

where I is an instance of A and I' is an instance of B. Then,

- f is easy to compute in C = polynomial time
- $A(I)$ is true if and only if $B(f(I))$ is true

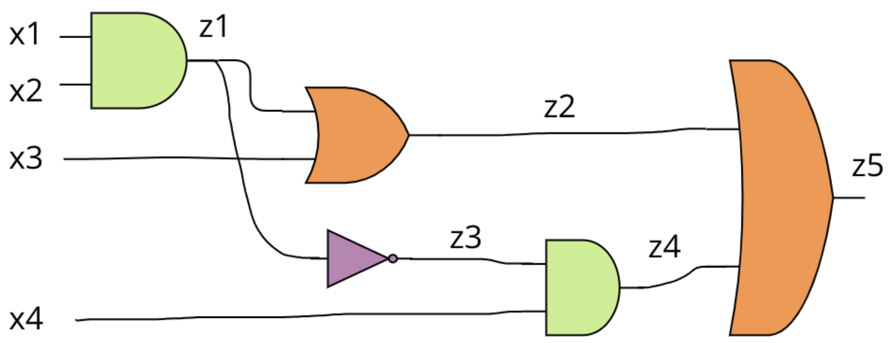
In this case, we say A is reduced to B in polynomial time, or $A \leq_p B$. This is an example of a **one-to-many reduction**.

4.1 FORMULA-SAT

FORMULA-SAT is a decision problem wherein the input is a logical formula and the output is whether or not that formula is satisfiable.

Claim: CIRCUIT-SAT reduces to FORMULA-SAT. That is, CIRCUIT-SAT $\leq_p$ FORMULA-SAT.

Recall our circuit from earlier. We label the outputs of each gate $z_1, .., z_5$ as follows:



Based on this circuit, we define the formula

$$(z_1 \leftrightarrow x_1 \wedge x_2) \wedge (z_2 \leftrightarrow x_3 \vee z_1) \wedge (z_3 \leftrightarrow z_1) \wedge (z_4 \leftrightarrow z_3 \wedge x_4) \wedge (z_5 \leftrightarrow z_2 \vee z_4) \wedge z_5.$$

This formula is satisfied when the circuit is satisfied, and vice versa. So we know FORMULA-SAT may be verified in polynomial time and is therefore in NP.

To prove FORMULA-SAT is NP-complete, we must also prove that it is NP-hard. We define

$$T_{CSAT}(n) = O(n) + T_{FSAT}(n).$$

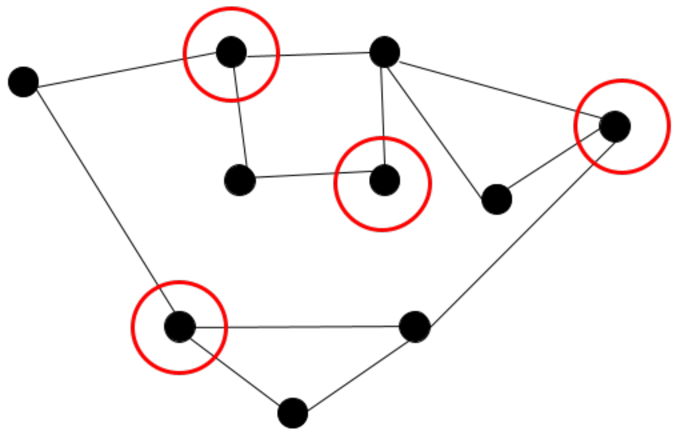
It follows that if $T_{FSAT}(n)$ is a polynomial, then $T_{CSAT}(n)$ is also a polynomial. So we may define a reduction

$$FSAT \in P \Rightarrow CSAT \in P.$$

That is, FSAT cannot be easier than CSAT. So FSAT is NP-hard.

4.2 IND and 3-SAT

Suppose we have an undirected graph G with a set of vertices V and set of edges E. We know $I \subset V$ is an independent set if for all $v, u \in I (v, u) \notin E$. An example of an independent set is shown below.



Now we may define the decision problem IND:

Given G and a constant k, is it possible for a set of vertices I to be independent if $|I| \geq k$?

We see that IND $\in$ NP. To show that IND is NP-hard, we must prove that a known NP-hard problem X reduced to IND. That is, that $X \leq_p$ IND. In class, we began to show how to use 3-SAT (3 conjunctive normal form satisfiability) as X to prove that IND is NP-hard, but ran out of time (I am assuming we will finish this example next lecture - I will update these notes at that time).

5 References

[1] Har-Peled, S. (2018). NP Completeness I. https://sarielhp.org/teach/notes/algos/files/01_npc.pdf