# 08: NP Completeness

CS 473u - Algorithms - Spring 2005

May 9, 2005

# 1 Introduction

## 1.1 Efficient Solution

**Problem 1.1** What is an efficient algorithm?

Answer: Runs quickly.

**Question 1.2** *What is quickly?*

1. Scale with input size.
2. Dont care about constants (faster cpus).
3. Asymptotic running time: polynomial time.

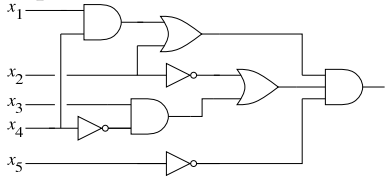
**Question 1.3** *Can we solve all problems in polynomial time?*

Answer: No.

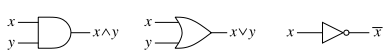
**Problem 1.4 Circuit Satisfiability**

Input: A circuit C with $m$ inputs.

Output: *True* if there exists input for C that the circuit return true on. *False* otherwise.




where




Currently, all solutions konwn to Circuit Satisfiablity require checking all possiblities, requiring $2^m$time. Whcih is exponential time, which is bad.

**Question 1.5** *Can one solve Circuit Satisfiablity in polynomial time?*

*This is essentially the main open question in theoretical compute science.*

Common belief: Circuit Satisfiablity can NOT be solved in polynomial time.

Circuit Satisfiablity has an interesting properties:

1. Given a supposed positive solution, with a detailed assignment (i.e., proof):
   $x_1 \leftarrow 0$, $x_2 \leftarrow 1, ..., x_m \leftarrow 1$
   one can verify in polynomial time if this assignement really satisifies $C$.
   This is the well known difference in hardness between coming up with a proof (hard), and checking that a proof is correct (easy).

2. It is a decision problem.

**Definition 1.6** $P$ is the class of all decision problems that can be solved in polynomial time (in the size of the input).

**Definition 1.7** $NP$ is the class of all decision problems that can be verified in polynomial time.

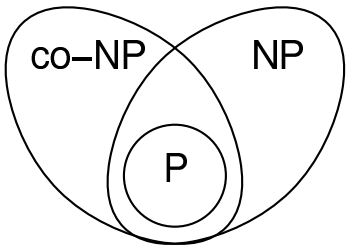
More precisely, if an answer to an instance (i.e., an input circuit $C$) is 'yes' then there exists a proof of this fact, and this proof can be verfiied in polynomial time.

Clearly, if a decision problem can be solved in polynomial time, then it can be verified in polynomial time. Thus,

$$P \subseteq NP.$$

**Remark 1.8** $NP$ stands for Non-detereministicly Polinomial. The name come from a formal defintion of this class using Turing machines.

**Definition 1.9** $co - NP$ is the opposite of $NP$ - if the answer is No, then there exists a short proof, and this proof can be verified in polynomial time.




**Definition 1.10** A problem $\Pi$ is $NP$-hard $\Leftrightarrow$If $\Pi$ can be solved in polynomial time then $P = NP$.

**Question 1.11** *Are there any problems which are NP-hard???*

*Intuitively being NP-hard implies that it is redicilously hard.*

*Intuitively, it would imply that proving and verifying are equally hard - which nobody that did 373 belives is true.*

*In particular, a problem which is NP-hard is at least as hard as ALL the problems in $NP$, as such it is safe to assume, based on overwhelming evidence that it can not be solved in polynomial time.*

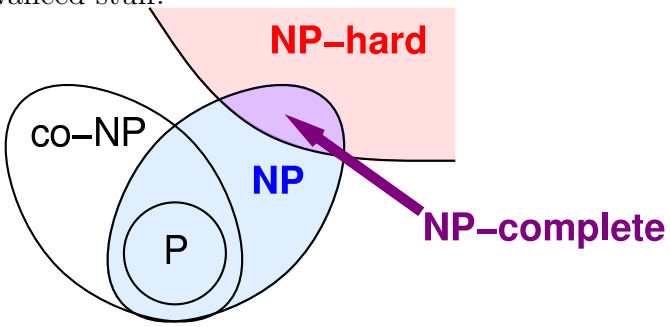
**Theorem 1.12** *(Cook's Theorem) Circuit satisfiablity is $NP$-hard.*

**Definition 1.13** A problem Πis $NP$-Complete (NPC in short) if it both $NP$-hard and in $NP$.

Clearly, Circuit Satisfiablity is $NP$-Complete.

By now, thousands of problems have been shown to be $NP$-Complete. It is extremely unlikely that any of them can be solved in polynomial time.

The standard text on this topic is [GJ79]. [ACG+99] is a more recent and more updated, and contain more advanced stuff.




**Definition 1.14** In the FORMULA SATISIFIABLITY problem, (aka SAT) we are given a formula, for example:

$$\Big(a \vee b \vee c \vee \overline{d}\Big) \Leftrightarrow \Big((b \wedge \overline{c}) \vee (\overline{\overline{a} \Rightarrow d}) \vee (c \neq a \wedge b)\Big)$$

and the question is whether we can find an assignment to the variables $a, b, c, \ldots$ such that the formula evaluates to ***True***.

It seems that SAT and Circuit Satisifiablity are "similar" and as such both should be NP-hard.

## 1.2 Reductions

Let $A$ and $B$ be two decision problems.

Given an input $I$ for problem $A$, a *reduction* is a transformation of the input $I$ into a new input $I'$ such that:

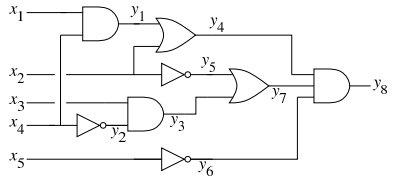
$$A(I) \;\; is \;\; TRUE \quad \Leftrightarrow \quad B(I') \;\; is \;\; TRUE$$

Thus, one can solve $A$ by first transforming and input $I$ into an input $I'$ of $B$, and solving $B(I')$.

This idea of using reductions is omnipresent, and used almost in any program you write.

$T : I \rightarrow I'$ be the input transformation. How fast is $T$?

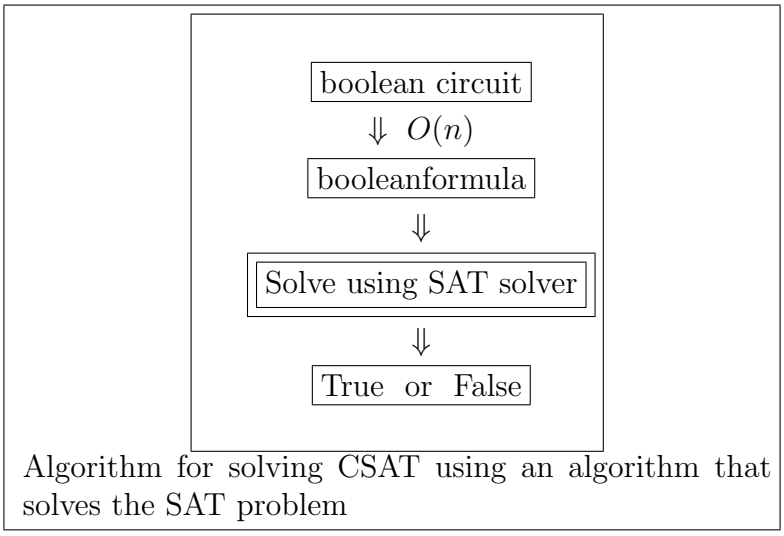
Polynomial reductions - reductions that take polynomial time.
For example, given an instanct of Circuit Satisifiablity:




Lets explictly write down what the circuit computes:
$y_1 = x_1 \wedge x_4$
$y_2 = \overline{x_4}$
$y_3 = y_2 \wedge x_3$
$y_4 = x_2 \vee y_1$
$y_5 = \overline{x_2}$
$y_6 = \overline{x_5}$
$y_7 = y_3 \vee y_5$
$y_8 = y_4 \wedge y_7 \wedge y_6$
$y_8$
The circuit is satisfiable **if and only if** there is an assignment such that all the above formulas hold. Alternativley:

$$\begin{aligned}(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = y_2 \wedge x_3) \\ \wedge (y_4 = x_2 \vee y_1) \wedge (y_5 = \overline{x_2}) \\ \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \\ \wedge (y_8 = y_4 \wedge y_7 \wedge y_6) \wedge y_8\end{aligned}$$

the circuit has a SAT assignement iff so does the above formula.
Clearly, this transformation can be done in polynomial time.
What we get:




Algorithm for solving CSAT using an algorithm that solves the SAT problem

Namely, given a solver for SAT we can solve the CSAT problem.

$$2^n \leq T_{CSAT}(n) \leq O(n) + T_{SAT}(O(n))$$

If $SAT \in P$ then $CSAT \in P$.
We "know", that $T_{CSAT}(n)$ is exponential. Thus, $T_{SAT}(n)$ must also be exponential:

$$T_{SAT}(n) \geq T_{CSAT}(\Omega(n)) - O(n)$$

We just proved that SAT is as hard as CSAT. Clearly, $SAT \in P$.

**Theorem 1.15** *SAT (formula satisfiability) is NP-complete.*

# 2 More NP Complete problems

## 2.1 3SAT

A boolean formula is in conjunctive normal form (CNF) if it is a conjunction (AND) of several *cluases.*

A clause is the disjunction (or) of several *literals.*
A literal is either a variable or a negation of a variable.

$$\overbrace{(a \vee b \vee \overline{c})}^{clause} \wedge (a \vee \overline{e}) \wedge (c \vee e)$$

3CNF formula is a CNF *exactly* three literals in each clause.
3SAT is formula satisfiablity when the formula is restricted to be 3CNF.
Proof 3SAT is $NP$-complete by reduction from Circuit Satisfiablity:

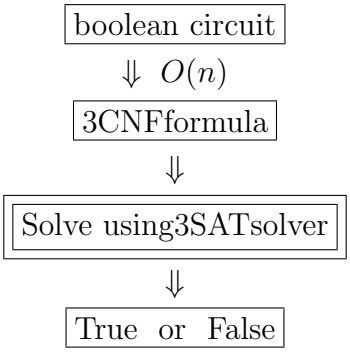
1. Make sure every AND/OR gate has only two inputs.
2. Write down the circuit as a formula.
3. Change every gate clause into a CNF:

$$\begin{aligned} a = b \wedge c &\rightarrow (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b) \wedge (\overline{a} \vee c) \\ a = b \vee c &\rightarrow (\overline{a} \vee b \vee c) \wedge (a \vee \overline{b}) \wedge (a \vee \overline{c}) \\ a = \overline{b} &\rightarrow (a \vee b) \wedge (\overline{a} \vee \overline{b}) \end{aligned}$$

4. Make sure every clause is exactly three literals:

$$\begin{aligned} a &\rightarrow (a \vee x \vee y) \wedge (a \vee \overline{x} \vee y) \wedge (a \vee x \vee \overline{y}) \wedge (a \vee \overline{x} \vee \overline{y}) \\ a \vee b &\rightarrow (a \vee b \vee y) \wedge (a \vee b \vee \overline{y}) \end{aligned}$$

This is it. We generated an equivalent 3CNF . We get the following solver for the Circuit Satsifiablity problem:




Namely, $T_{CSAT}(n) \leq O(n) + T_{3SAT}(O(n))$, which implies that if we have a polynomial time algorithm for 3SAT, we would solve CSAT is polynomial time.

**Theorem 2.1** *3SAT is NP-Complete.*

# References

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation.* Berlin, 1999.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.