Limits of Computation

20 - Complete Problems
Bernhard Reus

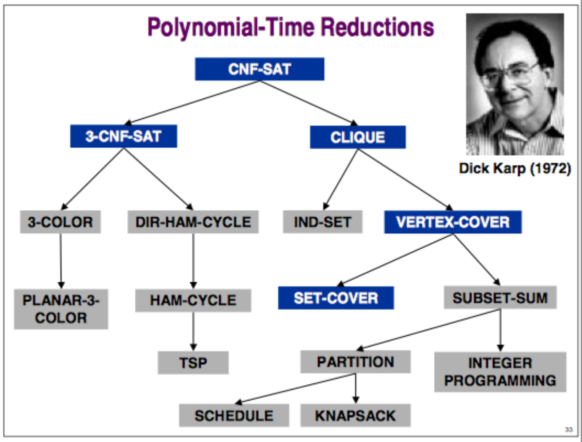
The story so far

- we have seen **NP** contains problems that seem intractable
- we don't know whether **P** = **NP**
- we have defined **NP**-complete problems ("hardest" problems in **NP**)
- "angle" to analyse **NP**

“Hard” Problems in **NP**

THIS TIME

- “the mother” of **NP**-complete problems (SAT)
- more examples of **NP**-complete problems in all areas of Computer Science (obtained via reductions)



http://www.cs.princeton.edu/~wayne/cs423/lectures/reductions-poly-4up.pdf

Boolean Expressions

Definition 20.1 (Truth assignment and evaluation). A *truth assignment* for $\mathscr{F}$ is a function θ mapping variables (of $\mathscr{F}$) to truth values (i.e. true and false). This mapping is similar to the stores we used to execute `WHILE`-programs where each program variable was mapped to a binary tree. If we have truth assignment θ for a boolean expression $\mathscr{F}$ then we can apply the truth assignment to the expression, obtain a closed formula and then evaluate this. For this we briefly write $\theta(\mathscr{F})$.

Example 20.1. For instance, if θ maps x to *true*, y and z to false, and $\mathscr{F} = (x \wedge y) \vee \neg z$ then $\theta(\mathscr{F}) = (\text{true} \wedge \textit{false}) \vee \neg \textit{false}$ which can be evaluated to true.

CNF

Definition 20.2 (conjunctive normal form (CNF)). A boolean expression is in *conjunctive normal form* iff it is a finite conjunction of finite disjunctions of literals:

$$(A_{11} \vee A_{12} \ldots \vee A_{1n_1}) \wedge \cdots \wedge (A_{m1} \vee A_{m2} \vee \ldots \vee A_{mn_m})$$

where each A_{ij} is a literal, i.e. either a variable or a negated variable (x or $\neg x$). The disjunctive formulae $(A_{i1} \vee A_{i2} \ldots \vee A_{in_i})$ are called *clauses*.

Example 20.2. An example for a Boolean expression in CNF is:

$$(p \vee \neg q) \wedge \neg q \wedge (\neg p \vee p \vee q)$$

is it satisfiable?

Satisfiability

Definition 20.3 (Satisfiability). A boolean expression $\mathscr{F}$ is called *satisfiable* if it evaluates to true for some truth assignment θ.

Example 20.2. An example for a Boolean expression in CNF is:

$$(p \vee \neg q) \wedge \neg q \wedge (\neg p \vee p \vee q)$$

is it satisfiable?

The SATisfiability Problem

Definition 20.4 (SAT). The *Satisfiability problem*[2], short SAT, is defined as follows

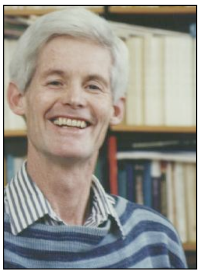
$$\text{SAT} = \{\mathscr{F} \mid \mathscr{F} \text{ is a satisfiable boolean CNF expression } \}$$

In other words, the SAT problem can be presented like this:

- **instance**: a boolean expression $\mathscr{F}$ in CNF (conjunctive normal form)
- **question**: is $\mathscr{F}$ satisfiable?

SAT$_{\text{isfiability}}$ problem

SAT is clearly decidable as one can try all possible truth assignments (there are only finitely many variables) but this leads to an exponential time algorithm.


Stephen A. Cook

Cook-Levin Theorem:
SAT is **NP**-complete.


Leonid Levin

Proof sketch follows.

SAT is in **NP**

Theorem: SAT is in **NP**.

Proof: We have to provide a polynomial time verifier for SAT:

Verifier takes a formula *F* and as certificate a truth assignment θ.

It checks whether *F* evaluates to true under the assignment θ (in time linear in number of variables of *F* and thus size of *F*).

Therefore the verifier runs in time polynomial in size of *F*.

SAT is **NP**-hard

Theorem: SAT is **NP**-hard.

Proof Sketch: Given a decision problem $x \in A$ we must find a polynomial time reduction *f* that maps x into a CNF *F* such that $x \in A$ if and only if *F* is *satisfiable*.

We know A is in **NP** and thus (see Lecture 18) there is a nondet. TM program p that accepts A.

Idea: *F* describes *all transition sequences of* p *with input* x and ensure satisfiability of *F* iff there is an accepting sequence *of* p *with input* x.

Formula *F* can be constructed from p and x in time *polynomial in* x.

SAT is **NP**-complete

Corollary: SAT is **NP**-complete.

Proof:

SAT is in **NP-hard** (previous slide).

SAT is in **NP** (two slides ago)

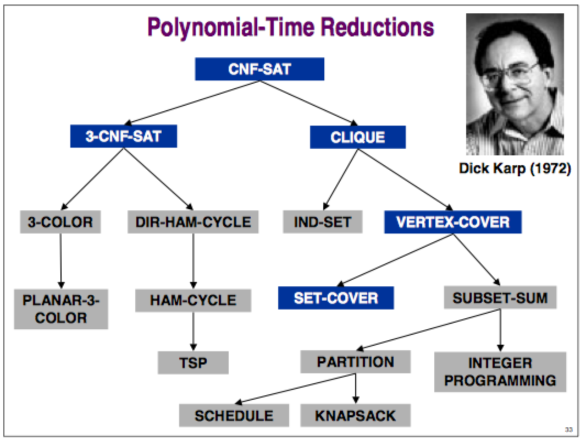
thus, by definition, SAT is **NP**-complete.

NP-complete problems

- The "hard" problems from Lecture 17: TSP, Graph Colouring, 0-1 Knapsack, Integer Linear Programming are all **NP**-complete (i.e. really "hard" in a precise sense).
- This can be shown by reducing SAT (or another **NP**-complete problem) in polynomial time to them.

because polynomial time reducibility is a transitive relation

More NP-complete Problems



http://www.cs.princeton.edu/~wayne/cs423/lectures/reductions-poly-4up.pdf

- SAT = CNF-SAT
- 3-CNF-SAT is SAT where every clause has exactly 3 literals
- SAT $\leq_p$ 3-SAT (!)
- (DIR-)HAM-CYCLE (Hamiltonian Cycle): given (directed) graph, is there a tour that visits each vertex exactly once.
- 3-SAT $\leq_p$ DIR-HAM-CYCLE
- HAM-CYCLE $\leq_p$ TSP why?

Games

need to be generalised to arbitrary size

More NP-complete problems

Theorem: The Sudoku problem is NP-complete.

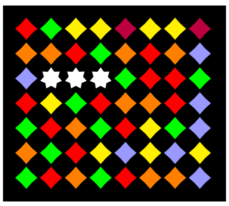
rank 3

			4	5				
							6	
1	2						5	
				3				
	4							
			8	1	2			
		1				8		
		6					9	

Definition 20.7 (Sudoku problem).

- **instance**: a Sudoku board of rank n partially filled with numbers in $\{1,2,\ldots,n^2\}$.
- **question**: can one fill the blank cells with numbers in $\{1,2,\ldots,n^2\}$, such that each row, each column, and each of the n blocks contain each number exactly once?

More NP-complete problems



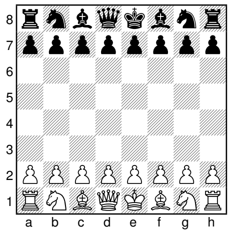
Theorem:
Deciding whether for a (generalised) Three-Tile-Matching game, like *Bejeweled* or *Candy Crush*, there is a sequence of moves such that two specific tiles (gems, candies) can be swapped is NP-complete.

An **EXP**-complete game

Theorem:
The (generalised) Chess problem is **EXP**-complete.

how generalise chess to a n x n board???

8x8 chess



Definition 20.6 (Chess problem).

- **instance**: an arbitrary position of a generalized chess-game on an $n \times n$ chess-board
- **question**: can White (or Black) win from that position?

Database Query Evaluation Problem

Definition 20.10 (Query Evaluation problem).

relations

$\exists y_1 \ldots \exists y_n . \alpha_1 \wedge \ldots \wedge \alpha_m$

- **instance**: given a relational database $(D, R_1, \ldots, R_s)$ and a boolean conjunctive query ϕ
- **question**: does ϕ evaluate to true in database $(D, R_1, \ldots, R_s)$?

D is (finite) domain of schema columns

Theorem 20.6 ([1]). *The Query Evaluation problem is **NP**-complete when applying the combined complexity measure, i.e. measure time usage w.r.t. database and query expression size.*

Polynomial for all queries in size of data alone (data complexity)

Polynomial for so-called acyclic queries, where joins can be done without holding "intermediate results"

END

Next time:
How do we deal with **NP**-complete problems then if they are so hard?