CSE 200 - Computability and Complexity
NP and NP-complete problems

Russell Impagliazzo and Jiawei Gao

Oct., 2016
Goals for today

1. Introduce the class \( NP \)
2. Show that \( NP \) problems are (equivalent to) the naturally occurring search and optimization problems that arise in almost all computational applications
3. Introduce \( NP \)-completeness
4. Show some (artificial) examples of \( NP \)-complete problems.
5. Work towards more natural examples (might need to be continued...)
Are natural computational problems hard?

We’ve seen computational problems that cannot be solved algorithmically. The time hierarchy theorem tells us that there are computable problems that require arbitrarily large times to solve. But are these the problems that we need to solve most of the time? How frequently are problems that arise in applications hard to solve? How hard? Do we know they are hard, or just believe it?
Problems from different domains have similar structure

Consider the following list of problems that seem computationally hard for one reason or another, that arise in a variety of situations.

- Solving a big sudoku puzzle
- Packing boxes into the trunk of a car
- Finding the lowest energy folding of a protein
- Finding the arrangement of components on a chip that minimizes area
- Deciding whether a protocol allows a security violation
- Factoring a large integer

What do these problems have in common?
Search and Optimization

These are all examples of search and optimization problems, where there are a huge number of possible solutions and you are looking either for one that meets certain constraints, or for the “best” solution. Can we come up with some other examples?
Other examples

1.
2.
3.
4.

Not quite the same:
1.
2.
3.
4.
Search problems

SearchR = “Given $x$, find a $y$ so that $R(x, y)$ or, if there is no such $y$, output 0.”
If we are even to imagine a polynomial-time solution to $SearchR$, what properties should $R$ have?

1.
2.
Search problems

SearchR: “Given x, find a y so that R(x, y) or, if there is no such y, output 0.”

If we are even to imagine a polynomial-time solution to SearchR, R should have the two properties

1. y should be represented as a reasonable sized binary string, |y| \leq poly(|x|).
2. R(x, y) should be efficiently decidable, R \in P.

SearchP = \{ SearchR(x, y) : R \in P, |y| \leq poly(|x|) \}
Examples

1. Factoring: Given $N$, find $p, q > 1$ so that $pq = N$
2. 3-coloring: Given $G$, find a map of vertices to $R, G, B$ so that no two adjacent vertices map to the same color (or say 0 if no 3-coloring exists).
3. BigIndSet: Given $G$ and $1 \leq k \leq n$, find a set of $k$ vertices that have no edges between them
4. CircuitSAT: Given $C(x_1..x_n)$, find an assignment $a_1...a_n$ with $C(a_1..a_n) = 1$. 
Decision problems

Let $R(x, y)$ be a relation as before.

We’ll use $\text{DecR}$ to denote the decision version of the above Search problem: Given $x$, decide: is there a $y$ so that $R(x, y)$?

Not every decision problem is equivalent to the corresponding search problem: e.g., primality testing vs. factoring. But we’ll see soon that the class of decision problems is equivalent to the class of search problems.

1. Prime testing: Given $N$, are there $p, q > 1$ so that $pq = N$?
2. 3-colorability: Given $G$, is there a map of vertices to $R, G, B$ so that no two adjacent vertices map to the same color (or say 0 if no 3-coloring exists).
3. BigIndSet: Given $G$ and $1 \leq k \leq n$, is there a set of $k$ vertices that have no edges between them
4. CircuitSAT: Given $C(x_1..x_n)$, is there an assignment $a_1...a_n$ with $C(a_1..a_n) = 1$?
A related class is $\text{OptP}$
Let $F(x, y)$ be a function.
$\text{OptF}$ is the problem, given $x$, find $\max_y(F(x, y))$
$\text{OptP}$ is the class of problems $\text{OptF}$ for $F \in P$ with $|y| \leq \text{poly}(|x|)$.

1. $\text{MIS}$: Given $G$, how big is the largest independent set in $G$?
2. $\text{LongestPath}$: Given $G$, a graph with edge weights, how long is the longest simple path in $G$?
3. Chromatic Number: What is the smallest number of colors we can use to color $G$?
A class that captures search and optimization

Remember that a *non-deterministic* Turing Machine has possibly multiple transitions for the same state and read symbols, and so an number of runs on the same input. We view the machine as accepting if there is any run which accepts.

\( \text{NTIME}(T(n)) \) is the class of problems decided by a non-deterministic machine where every (particular)run takes time at most \( O(T(n)) \). (In aggregate, all runs might be exponential in \( T \)).

\( \text{NP} = \bigcup_k \text{NTIME}(n^k) \).

\( P \subseteq \text{NP} \subseteq \text{EXP} = \bigcup_k \text{TIME}(2^{n^k}) \).
The connection

**Theorem:** \( NP = \{ DecR(x, y) : R \in P, |y| \leq poly(|x|) \} \)
So \( NP \) is the decision version of natural search problems.
Proof: Let \( L = DecR \) for \( R(x, y) \in P \) with \( |y| \leq c|x|^k \). Consider the following non-deterministic machine for \( L \): Write a sequence of \( c|x|^k \) bits on a second tape, non-deterministically branching on writing 1 or 0.
Verify that \( R(x, y) \) for \( x \) the input and \( y \) the contents of that tape. Accepting run iff \( R(x, y) \) for some \( y \).
Other direction

Assume $L$ is accepted in time $T(n)$ by some non-deterministic machine $M$. Consider the relation $R(x, y)$ that denotes: “$y$ is a complete description of a run of $M$ on $x$ for at most $T(n)$ steps”. $|y| = O(T(n))$ and it can be easily verified by simulating $M$ step by step, using the transitions in $y$. 
Equivalence of decision, search, optimization

Theorem TFAE:

1. \( P = NP \)
2. Every problem in \( \text{SearchP} \) can be solved in \( P \)
3. Every problem in \( \text{OptP} \) can be solved in \( P \)

We’ll prove the third implies the second, the second implies the first and the first implies the third, completing the chain.

If every optimization problem can be solved in \( P \), let \( R \) be a relation as before. To solve \( \text{SearchR} \) define \( F(x, y) = \)

Then solving \( \text{OptF} \) gives a solution to \( \text{SearchR} \).
Search to optimization

We’ll prove the third implies the second, the second implies the first and the first implies the third, completing the chain. If every optimization problem can be solved in \( P \), let \( R \) be a relation as before. To solve \( \text{Search}_R \) define \( F(x, y) = \text{binvalue}(y) \) if \( R(x, y) \), 0 ow.

Then solving \( \text{Opt}_F \) gives a solution to \( \text{Search}_R \).
The easiest step is Decision to Search
Say we can solve SearchR. Solve it. If the answer is 0, we know no solution exists, and return 0. Otherwise, the algorithm returned some solution, so some solution exists, and return 1.
This is the trickiest one, but still straight-forward. Let $F(x, y)$ be the function we want to maximize. Note that since $F \in P$, $|F(x, y)| \leq poly(|x|)$, so as a number $1 \leq F(x, y) \leq 2^{poly(|x|)}$. Using an oracle for a decision version in $NP$, we can maximize $F$ using
Define: $R((x, V), y) = 1$ iff
Optimization to Decision

This is the trickiest one, but still straight-forward. Let $F(x, y)$ be the function we want to maximize. Note that since $F \in P$, $|F(x, y)| \leq poly(|x|)$, so as a number $1 \leq F(x, y) \leq 2^{poly(|x|)}$

Using an oracle for a decision version in $NP$, we can maximize $F$ using binary search

Define: $R((x, V), y) = 1$ iff $F(x, y) \geq V$
Significance of $P$ vs $NP$

Thus, if the class $NP$ contains only easy problems, every type of search and optimization is easy. So $P = NP$ is a very strong hypothesis; but $P \neq NP$ still leaves open a lot of possibilities.
**NP-complete problems**

The question of whether some problem in $NP$ is hard is pretty abstract. $NP$-completeness is a way to capture the complexity of the whole class in a single problem. We’ll first prove that artificial problems are $NPC$ but later use this to show very concrete natural problems $NPC$, coming from all domains of optimization and search.

For a notion of reduction, we use the notation $A \leq B$ to mean that problem $A$ can be reduced to problem $B$. For $NPC$, the most common form of reduction is polynomial time mapping reductions, $A \leq_{mp} B$.

$A$ is $NP$-complete if

- In class $A \in NP$

Hard for class For every $B \in NP$, $B \leq_{mp} A$.

If it has property 2 but not necessarily 1, we say $A$ is $NP$-hard.
Theorem: Let $A$ be $NP$-complete. Then $P = NP$ if and only if $A \in P$.

Proof: If $P = NP$, then since $A \in NP$, $A \in P$. If $A \in P$, then every $B \in NP$ reduces to $A$, so each such $B$ is in $P$, so $P = NP$. 
An NPC problem

Our first $NP$-complete problem will involve circuits. Let $C(x_1, \ldots x_n)$ be a circuit with $n$ variables. We say values $x_1 = a_1, \ldots x_n = a_n$ is an assignment, and it satisfies $C$ if $C(a_1, \ldots a_n) = 1$. The Circuit-SAT problem is to decide whether a satisfying assignment exists.
Circuit SAT is \textit{NPC}

We did all the work for this last class, in showing $P$ has uniform families of Boolean circuits.
Say we have a language $\text{Dec}R \in \text{NP}$: Given $x$ is there a $y$ so that $R(x,y)$?
To reduce to circuit sat, let $N = |x| + |y|$. Build a circuit $C_R(x_1\ldots x_n, y_1\ldots y_m)$ that simulates $R(x,y)$. Let $C_x$ be $C_R$ with the $x$ input gates substituted by the actual values of the bits of $x$.
Then $C_x$ is satisfiable if and only if $C_x(y_1\ldots y_m) = 1$ for some $y$, iff $C_R(x_1\ldots x_n, y_1\ldots y_m) = 1$ iff $R(x,y)$ as desired.
Special cases of interest

A CNF is a very special form of circuit. A literal is a variable or its negation. A clause is an OR of literals. A conjunctive normal form circuit is an AND of clauses.
\textit{k-CNF}

The width of a clause is the number of literals in it. Every function can be written as a \textit{CNF}, but functions such as parity require all variables in some clause, so the width might be as large as \( n \).

\( k \)-CNF circuits are CNF’s with every clause having width at most \( k \). This means they only compute some very special types of functions. Nevertheless, we’ll show that \( k \)-\textit{SAT}, the satisfiability problem restricted to \( k \)-CNF’s, is also \( NP \)-complete.