NP Completeness:

Complexity and (In)tractability
Complexity and (In)Tractability

• Scenario: Your pointy-haired boss asks you to solve a problem, e.g., checking that specifications for a custom made product are logically consistent before configuring the product.

• You spend weeks working on it but can’t find an efficient algorithm.

• Do you want to go back to him and say “I can’t find an efficient algorithm. I guess I’m too dumb.”

• Better to say “I can’t find an efficient algorithm, but neither can all these other smart (and famous) computer scientists!”
Complexity and (In)Tractability

• We have studied the complexity of *Algorithms* because this is more useful than studying the run time of particular implementations.

• Now we study the complexity of *Problems* and ask whether algorithms of a given complexity are *possible* for a given problem.

• (First example of this was analysis of comparison based sorting.)
Intro to Tractability

• So far we have studied “easy” problems: $O(\log n)$, $O(n \log n)$, $O(n^2)$ etc.

• Many real world problems are “hard:” best known algorithms are $O(2^n)$ and worse.
  – Traveling Salesman Problem
  – Linear Programming

We already know that it matters! Minutes versus years.

• Some problems are unsolvable (undecidable) by algorithms, most notably the Halting Problem. Set these aside ...
... Intro to Tractability ...

• Which solvable problems are inherently hard (exponential), and for which have we just not found the right algorithm yet?

• Sometimes a slight change makes a big difference!
  – Easy: Shortest path decision problem: exists path x-y with weight \( \leq m \)? \( O(n^2) \)
  – Hard(?): Longest path decision problem: exists path x-y with weight \( \geq m \)? Only \( O(2^n) \) algorithms are known.
... Intro to Tractability

• Initially we’ll look at decision problems only. (Decision problems are no harder: if you can optimize you can decide.)

• Professionals find this convenient because proofs can be done in terms of languages:
  – encode problem and problem solutions in a finite set of symbols
  – decision = “is this a solution?” = recognizing the language consisting of strings that encode problems and their solutions.
Polynomial Time Problems

- Assuming “reasonable” encoding scheme (e.g., log N bits used to represent number N; no “padding”).

- \( \mathbf{P} \) = set of all problems that can be solved by deterministic algorithms in polynomial time.
  - Deterministic: next step always unambiguous.

- \( \mathbf{P} \) includes most of what we have studied. (E.g. sorting can be done in \( \mathcal{O}(n^2) \) so is in \( \mathbf{P} \).)
Nondeterminism

- When faced with several options, a nondeterministic algorithm has the power to “guess” the right one.
- Equivalently, the machine can replicate computation paths as needed to try all options in parallel.
- This is not the same as having an N processor machine!
- This is a fictitious machine, only for purposes of definition of a class of problems.
Nondeterministic Polynomial

- **NP** = set of all problems that can be solved by nondeterministic algorithms in polynomial time.
  - Paradigm: *generate* nondeterministically, *test* in polynomial time.

- **Examples**
  - There are problems *not* in NP.
  - P is a subset of NP.
  - Million dollar question: *Is it a proper subset?* In other words, are there problems in NP that *cannot* be solved in polynomial time?
P = NP?

- NP includes problems for which solutions (once guessed nondeterministically) can be checked in polynomial time.
- The ability to guess seems so powerful it must be useful - but no one has been able to prove that it is!
- No one has proven that there is a problem that can only be solved in polynomial time if we can guess solutions.
- That is, no problem has been shown to be in NP but not in P!
Complexity of NP

• Best bound proven to date: all problems in NP can be solved in $O(2^{p(n)})$ time for some polynomial function $p$.
  – Let $q(n)$ be a polynomial bound on the time complexity of checking a solution. Hence it is also a bound on the length of a solution. (*why?*)
  – Also let $k = \text{size of encoding alphabet}$.
  – Then there are $k^{q(n)}$ possible guesses, each taking $q(n)$ to check.
  – We can choose $p(n)$ such that $2^{p(n)} > q(n)k^{q(n)}$

But this is an upper not lower bound!
Why care?

• **Many** important problems are known to be in NP, but no polynomial solution is known.

• Until we prove P!=NP the tantalizing possibility remains that such polynomial solutions exist.

• Yet programmers who are unaware of this issue may spend huge amounts of time trying to find a polynomial algorithm when the best minds have failed. (Remember pointy-hair!)

• Would help to know for sure!
Reducibility

• Polynomial Transformation from \( L_1 \) to \( L_2 \) (formal definition in terms of languages):
  
  \[ f : \Sigma_1^* \rightarrow \Sigma_2^* \text{ can be computed in deterministic polynomial time} \]
  \[ \forall x \in \Sigma_1^*, \ x \in L_1 \iff f(x) \in L_2. \]

• Polynomial Reduction (informal definition in terms of problems and solutions):
  
  – can transform problem A into problem B in polynomial time (and give it to a machine for B)
  
  – can transform a solution for B back into a solution for A in polynomial time
NP-Completeness

• **NP-Complete** problems are those to which any problem in NP can be reduced in polynomial time. They are the hardest of NP.

  – If you solve one NPC problem in polynomial time, you can solve *all* problems in NP in polynomial time: $P = NP$.

  – If one NPC problem cannot be solved in polynomial time, then all problems in NPC cannot be solved in polynomial time: $P \neq NP$. 
We Have Failed (so far)

• Many famous and not so famous computer scientists have failed to find a polynomial solution to any NP-Complete problem.
  – Hence many consider it to be highly unlikely that $P=NP$.

• Many have also failed to prove $P!=NP$, so a few believe that to be unlikely.

• Regardless, you should be aware of this class of problems so you don’t waste time trying to find an optimal solution (unless you accept the odds).
If P!=NP

- If P!=NP then there are problems in NP that are neither in P nor NP-Complete.
Example Problem: Satisfiability

• Given a Boolean formula over variables U in conjunctive normal form, is there an assignments to its variables that makes the formula true?
  – CNF: one or more clauses C where each clause is of form \((x_1 + x_2 + \ldots + x_n)\) where \(x_i\) are variables or their negation.
  – Example:
    • \(U = \{u_1, u_2\}\)
    • \(C = \{\{u_1,-u_2\},\{- u_1,u_2\}\}\): Yes
    • \(C' = \{\{u_1,u_2\},\{u_1,-u_2\},\{-u_1\}\}\): No
Cook’s Theorem: The 1\textsuperscript{st} NP-Complete Problem

• Cook showed that Satisfiability is NP-Complete:
  – Described a machine capable of solving any problem in NP: a nondeterministic Turing Machine (Church-Turing Thesis: Turing machines can compute any computable function)
  – Described the Nondeterministic Turing Machine with logic in CNF.
  – Showed that solving the Satisfiability problem corresponds to running the NTM, i.e., potentially solving any NP problem.
Reductions

• That was a very long, complex and technical proof!
• Easier approach: Now that we have one NP-Complete problem, prove others NP-Complete by reduction.
  – Restriction
  – Local Replacement
  – Component Design
(See Garey & Johnson)
Example Problems

• Hamiltonian Circuit:
  – Instance: A graph $G=(V,E)$
  – Question: Does $G$ contain an ordering $<v_1, v_2, ..., v_{|V|}>$ of $V$
    where $\{v_{|V|}, v_1\} \in E$ and $\{v_i, v_{i+1}\} \in E$ for all $1 \leq i < n$ ?

• Traveling Salesman
  – Instance: Set $C$ of $m$ cities; distance $d(c_i, c_j)$ for each pair of cities $c_i, c_j \in C$; positive integer $B$.
  – Question: Is there a tour of $C$ having length $B$ or less?
Example Reduction: Hamiltonian Circuit to Traveling Salesman

- Assume we’ve proven HC is NP-Complete

- Given G=(V,E) for a Hamiltonian Circuit problem, construct G’=(V’,E’) for TSP:
  - V’ = V
  - G’ is complete (E’ = V’xV’)
  - weights on edges in E’ are 1 if the corresponding vertices in G are connected; 2 if not.

- Ask TSP whether there is a tour of distance less than |V’|+1

Therefore TSP is NP-Complete
Basic NP-Complete Problems

- 3-Satisfiability
- 3-Dimensional Matching
- Vertex Cover
- Clique
- Hamiltonian Circuit
- Partition

**Satisfiability**

![Diagram showing the relationships between basic NP-Complete problems. The diagram shows 3SAT as a central node with arrows pointing to 3DM, VC, PARTITION, HC, and CLIQUE.]
NP-Hard Problems

• Generalization to problems not in NP, and problems other than decision problems.

• NP-Hard problems are those problems to which we can transform an NP-Complete problem.

• Cannot be solved polynomial unless $P=NP$
Approaching NP-Complete Problems

• There is still hope.

• **Approximation**: Solution guaranteed to be close to the best within some bound. (Sometimes there are NP-Complete as well).

• **Average** or partial coverage: Write algorithm that finds many but not all optimal solutions.

• Use most efficient exponential algorithm known (e.g., linear programming).
Sources


• Garey & Johnson: “Computers and Intractability,” Bell Telephone Laboratories 1979. Thorough and technical introduction to NP-Completeness. A classic. If you can find it, buy it and keep it!

• http://www.netaxs.com/people/nerp/automata/p-and-np0.html Short nontechnical introduction.

• http://userwww.sfsu.edu/~kroll/CLASSSES/510/42.htm Short slightly technical introduction.