Introduction to Complexity Theory

Read K & S Chapter 6.

Most computational problems you will face your life are solvable (decidable). We have yet to address whether a problem is "easy" or "hard". Complexity theory tries to answer this question.

Recall that a computational problem can be recast as a language recognition problem.

Some "easy" problems:
- Pattern matching
- Parsing
- Database operations (select, join, etc.)
- Sorting

Some "hard" problems:
- Traveling salesman problem
- Boolean satisfiability
- Knapsack problem
- Optimal flight scheduling

"Hard" problems usually involve the examination of a large search space.

Big-O Notation

- Gives a quick-and-dirty measure of function size
- Used for time and space metrics

A function f(n) is O(g(n)) whenever there exists a constant c, such that $|f(n)| \leq c \cdot |g(n)|$ for all $n \geq 0$.

(We are usually most interested in the "smallest" and "simplest" function, g.)

Examples:

$2n^3 + 3n^2 \cdot \log(n) + 75n^2 + 7n + 2000$ is $\underline{O(n^3)}$

$75 \cdot 2^n + 200n^5 + 10000$ is $\underline{O(2^n)}$

A function f(n) is *polynomial* if f(n) is O(p(n)) for some polynomial function p.

If a function f(n) is not polynomial, it is considered to be *exponential*, whether or not it is O of some exponential function (e.g. $n^{\log n}$).

In the above two examples, the first is polynomial and the second is exponential.

Comparison of Time Complexities

Speed of various time complexities for different values of n, taken to be a measure of *problem size*. (Assumes 1 step per microsecond.)

f(n)\n	10	20	30	40	50	60
n	.00001 sec.	.00002 sec.	.00003 sec.	.00004 sec.	.00005 sec.	.00006 sec.
n^2	.0001 sec.	.0004 sec.	.0009 sec.	.0016 sec.	.0025 sec.	.0036 sec.
n^3	.001 sec.	.008 sec.	.027 sec.	.064 sec.	.125 sec.	.216 sec.
n^5	.1 sec.	3.2 sec.	24.3 sec.	1.7 min.	5.2 min.	13.0 min.
2^n	.001 sec.	1.0 sec.	17.9 min.	12.7 days	35.7 yr.	366 cent.
3^n	.059 sec.	58 min.	6.5 yr.	3855 cent.	$2x10^8$ cent.	$1.3x10^{13}$ cent.

Faster computers don't really help. Even taking into account Moore's Law, algorithms with exponential time complexity are considered *intractable*. ∴Polynomial time complexities are strongly desired.

Polynomial Land

If $f_1(n)$ and $f_2(n)$ are polynomials, then so are:

- $f_1(n) + f_2(n)$
- $f_1(n) \cdot f_2(n)$
- $f_1(f_2(n))$

This means that we can sequence and compose polynomial-time algorithms with the resulting algorithms remaining polynomial-time.

Computational Model

For formally describing the time (and space) complexities of algorithms, we will use our old friend, the deciding TM (decision procedure).

There are two parts:

- The problem to be solved must be translated into an equivalent language recognition problem.
- A TM to solve the language recognition problem takes an encoded instance of the problem (of size *n* symbols) as input and decides the instance in at most $T_M(n)$ steps.

We will classify the time complexity of an algorithm (TM) to solve it by its big-O bound on $T_M(n)$.

We are most interested in polynomial time complexity algorithms for various types of problems.

Encoding a Problem

Traveling Salesman Problem: Given a set of cities and the distances between them, what is the minimum distance tour a salesman can make that covers all cities and returns him to his starting city?

Stated as a decision question over graphs: Given a graph G = (V, E), a positive distance function for each edge d: E→N+, and a bound B, is there a circuit that covers all V where $\sum d(e) \leq B$? (Here a *minimization* problem was turned into a *bound* problem.)

A possible encoding the problem:

- Give |V| as an integer.
- Give B as an integer.
- Enumerate all (v_1, v_2, d) as a list of triplets of integers (this gives both E and d).
- All integers are expressed as Boolean numbers.
- Separate these entries with commas.

Note that the sizes of most "reasonable" problem encodings are polynomially related.

What about Turing Machine Extensions?

Most TM extensions are can be simulated by a standard TM in a time polynomially related to the time of the extended machine.

- k-tape TM can be simulated in $O(T^2(n))$
- Random Access Machine can be simulated in $O(T^3(n))$

(Real programming languages can be polynomially related to the RAM.)

BUT… The nondeterminism TM extension is different.

A nondeterministic TM can be simulated by a standard TM in $O(2^{p(n)})$ for some polynomial p(n).
Some faster simulation method might be possible, but we don't know it.

Recall that a nondeterministic TM can use a "guess and test" approach, which is computationally efficient at the expense of many parallel instances.

The Class P

P = { L : there is a polynomial-time deterministic TM, M that decides L }

Roughly speaking, P is the class of problems that can be solved by deterministic algorithms in a time that is polynomially related to the size of the respective problem instance.

The way the problem is encoded or the computational abilities of the machine carrying out the algorithm are not very important.

Example: Given an integer n, is there a positive integer m, such that n = 4m?

Problems in P are considered tractable or "easy".

The Class NP

NP = { L: there is a polynomial time nondeterministic TM, M that decides L }

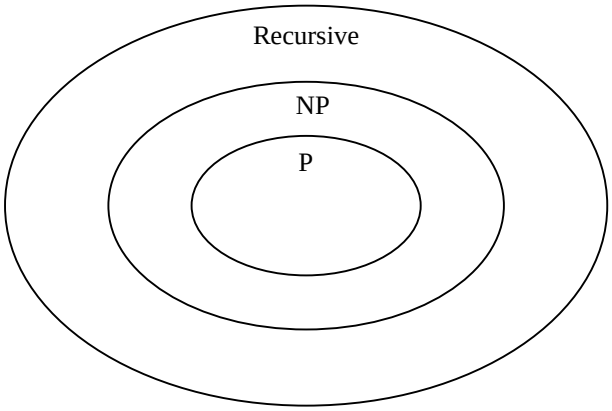
Roughly speaking, NP is the class of problems that can be solved by nondeterministic algorithms in a time that is polynomially related to the size of the respective problem instance.

Many problems in NP are considered "intractable" or "hard".

Examples:

- **Traveling salesman problem**: Given a graph G = (V, E), a positive distance function for each edge d: E→N+, and a bound B, is there a circuit that covers all V where $\sum d(e) \leq B$?
- **Subgraph isomorphism problem**: Given two graphs G_1 and G_2, does G_1 contain a subgraph isomorphic to G_2?

The Relationship of P and NP



We're considering only solvable (decidable) problems.

Clearly P ⊆ NP.

P is closed under complement.

NP probably isn't closed under complement. Why?

Whether P = NP is considered computer science's greatest unsolved problem.

Why NP is so Interesting

- To date, nearly all decidable problems with polynomial bounds on the size of the solution are in this class.
- Most NP problems have simple nondeterministic solutions.
- The hardest problems in NP have exponential deterministic time complexities.
- Nondeterminism doesn't influence decidability, so maybe it shouldn't have a big impact on complexity.
- Showing that P = NP would dramatically change the computational power of our algorithms.

Stephen Cook's Contribution (1971)

- Emphasized the importance of polynomial time reducibility.
- Pointed out the importance of NP.
- Showed that the Boolean Satisfiability (SAT) problem has the property that every other NP problem can be polynomially reduced to it. Thus, SAT can be considered the hardest problem in NP.
- Suggested that other NP problems may also be among the "hardest problems in NP".

This "hardest problems in NP" class is called the class of "NP-complete" problems.

Further, if any of these NP-complete problems can be solved in deterministic polynomial time, they all can and, by implication, P = NP.

Nearly all of complexity theory relies on the assumption that $P \neq NP$.

Polynomial Time Reducibility

A language L_1 is *polynomial time reducible* to L_2 if there is a polynomial-time recursive function τ such that $\forall x \in L_1$ iff $\tau(x) \in L_2$.

If L_1 is polynomial time reducible to L_2, we say L_1 reduces to L_2 ("polynomial time" is assumed) and we write it as $L_1 \propto L_2$.

Lemma: If $L_1 \propto L_2$, then $(L_2 \in P) \Rightarrow (L_1 \in P)$. And conversely, $(L_1 \notin P) \Rightarrow (L_2 \notin P)$.

Lemma: If $L_1 \propto L_2$ and $L_2 \propto L_3$ then $L_1 \propto L_3$.

L_1 and L_2 are *polynomially equivalent* whenever both $L_1 \propto L_2$ and $L_2 \propto L_1$.

Polynomially equivalent languages form an equivalence class. The partitions of this equivalence class are related by the partial order $\propto$.
P is the "least" element in this partial order.

What is the "maximal" element in the partial order?

The Class NP-Complete

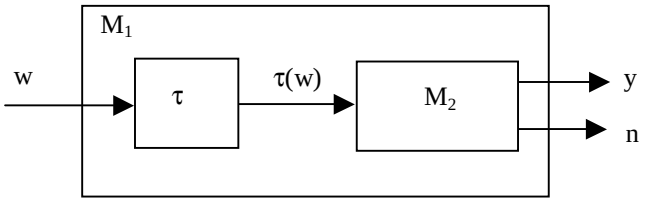
A language L is *NP-complete* if $L \in NP$ and for all other languages $L' \in NP$, $L' \propto L$.

NP-Complete problems are the "hardest" problems in NP.

Lemma: If L_1 and L_2 belong to NP, L_1 is NP-complete and $L_1 \propto L_2$, then L_2 is NP-complete.

Thus to prove a language L_2 is NP-complete, you must do the following:

1. Show that $L_2 \in NP$.
2. Select a known NP-complete language L_1.
3. Construct a reduction τ from L_1 to L_2.
4. Show that τ is polynomial-time function.



How do we get started? Is there a language that is NP-complete?

Boolean Satisfiability (SAT)

Given a set of Boolean variables $U = \{u_1, u_2, \ldots, u_m\}$ and a Boolean expression in conjunctive normal form (conjunctions of clauses—disjunctions of variables or their negatives), is there a truth assignment to U that makes the Boolean expression true (satisfies the expression)?

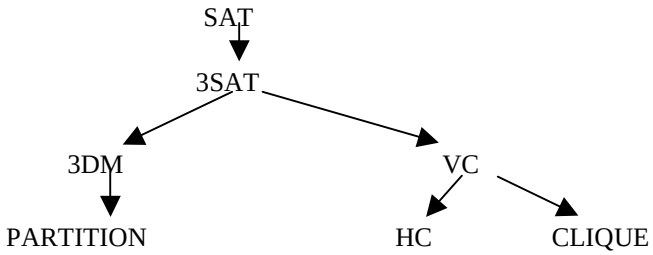
Note: All Boolean expressions can be converted to conjunctive normal form.
Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_3 \vee x_4 \vee \neg x_2)$

Cook's Theorem: SAT is NP-complete.

1. Clearly $SAT \in NP$.
2. The proof constructs a complex Boolean expression that satisfied exactly when a NDTM accepts an input string x where $|w| = n$. Because the NDTM is in NP, its running time is O(p(n)). The number of variables is polynomially related to p(n).

SAT is *NP-complete* because $SAT \in NP$ and for all other languages $L' \in NP$, $L' \propto SAT$.

Reduction Roadmap



The early NP-complete reductions took this structure. Each phrase represents a problem. The arrow represents a reduction from one problem to another.

Today, thousands of diverse problems have been shown to be NP-complete.

Let's now look at these problems.

3SAT (3-satisfiability)

Boolean satisfiability where each clause has exactly 3 terms.

3DM (3-Dimensional Matching)

Consider a set $M \subseteq X \times Y \times Z$ of disjoint sets, X, Y, & Z, such that $|X| = |Y| = |Z| = q$. Does there exist a *matching*, a subset $M' \subseteq M$ such that $|M'| = q$ and M' partitions X, Y, and Z?

This is a generalization of the marriage problem, which has two sets men & women and a relation describing acceptable marriages. Is there a pairing that marries everyone acceptably?

The marriage problem is in P, but this "3-sex version" of the problem is NP-complete.

PARTITION

Given a set A and a positive integer size, $s(a) \in N^+$, for each element, $a \in A$. Is there a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a) \ ?$$

VC (Vertex Cover)

Given a graph G = (V, E) and an integer K, such that $0 < K \leq |V|$, is there a *vertex cover* of size K or less for G, that is, a subset $V' \subseteq V$ such that $|V'| \leq K$ and for each edge, $(u, v) \in E$, at least one of u and v belongs to V'?

CLIQUE

Given a graph G = (V, E) and an integer J, such that
$0 < J \leq |V|$, does G contain a *clique* of size J or more, that is a subset $V' \subseteq V$ such that $|V'| \geq J$ and every two vertices in V' are joined by an edge in E?

HC (Hamiltononian Circuit)

Given a graph G = (V, E), does there exist a *Hamiltonian circuit*, that is an ordering $\langle v_1, v_2, \ldots, v_n \rangle$ of all V such that $(v_{|V|}, v_1) \in E$ and $(v_i, v_{i+1}) \in E$ for all i, $1 \leq i < |V|$?

Traveling Salesman Prob. is NP-complete

Given a graph G = (V, E), a positive distance function for each edge d: E→N+, and a bound B, is there a circuit that covers all V where $\sum d(e) \leq B$?

To prove a language TSP is NP-complete, you must do the following:

1. Show that $TSP \in NP$.
2. Select a known NP-complete language L_1.
3. Construct a reduction τ from L_1 to TSP.
4. Show that τ is polynomial-time function.

TSP ∈ NP: Guess a set of roads. Verify that the roads form a tour that hits all cities. Answer "yes" if the guess is a tour and the sum of the distances is ≤ B.

Reduction from HC: Answer the Hamiltonian circuit question on G = (V, E) by constructing a complete graph where "roads" have distance 1 if the edge is in E and 2 otherwise. Pose the TSP problem, is there a tour of length ≤ |V|?

Notes on NP-complete Proofs

The more NP-complete problems are known, the easier it is to find a NP-complete problem to reduce from.

Most reductions are somewhat complex.

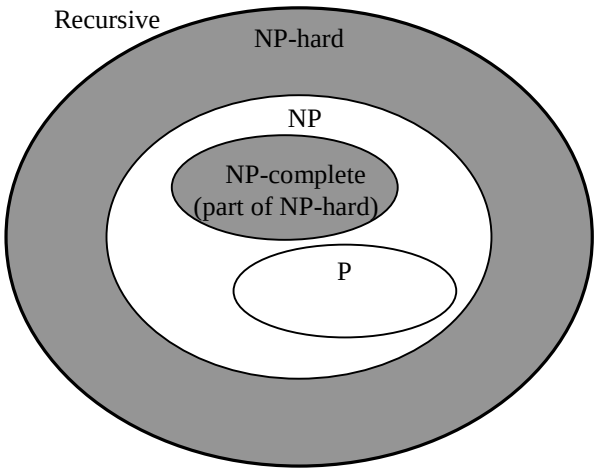
It is sufficient to show that a restricted version of the problem is NP-complete.

More Theory

NP has a rich structure that includes more than just P and NP-complete. This structure is studied in later courses on the theory of computation.

The set of recursive problems outside of NP (and including NP-complete) are called *NP-hard*. There is a proof technique to show that such problems are at least as hard as NP-complete problems.

Space complexity addresses how much tape does a TM use in deciding a language. There is a rich set of theories surrounding space complexity.



Dealing with NP-completeness

You will likely run into NP-complete problems in your career. For example, most optimization problems are NP-complete.

Some techniques for dealing with intractable problems:

- Recognize when there is a tractable special case of the general problem.
- Use other techniques to limit the search space.
- For optimization problems, seek a near-optimal solution.

The field of *linear optimization* springs out of the latter approach. Some linear optimization solutions can be proven to be "near" optimal.

A branch of complexity theory deals with solving problems within some error bound or probability.

For more: Read Computers and Intractability: A Guide to the Theory of NP-Completeness by Michael R. Garey and David S. Johnson, 1979.