1 Computation and Intractability

In this series of lecture notes, we have discussed several problems for which there exist polynomial time algorithms for producing solutions – min-cut, shortest s-t path, whether a graph contains an Eulerian cycle, among others. However, it is quite often the case that if you change the statement for an "easy" problem slightly, the problem becomes "hard," i.e. that the best known solution algorithms have exponential running time in the size of the input. As examples we list hard problems similar to the problems mentioned above – max-cut, longest s-t path, and whether a graph contains a Hamiltonian cycle.

For many fundamental problems in computer science, biology, combinatorics and elsewhere, there is no efficient algorithm known. Moreover, for many problems we don't know how to prove that no efficient algorithms exist. However, some progress has been made. A large class of these difficult problems have been shown to be equivalent, so that an efficient solution to one may be used to give an efficient solution to all. This is the class of NP-complete problems.

1.1 Polynomial Time Reducibility

Problem X is polynomial time reducible to problem Y ($X \underset{p}{\leq} Y$) iff for every instance of X there is an algorithm that solves X with polynomially many operations and polynomially many calls to a "black box" that solves a given instance of Y. In other words, if for every instance of problem X we can represent it as a instance of Y (or sequence of such instances) using a polynomial number of computations, then we say that Y is **harder** than X and we write $X \underset{p}{\leq} Y$. This is because if we can solve Y efficiently, we can solve X efficiently. If Y can be in turn solved through black box calls that solve instances of X, i.e. $Y \underset{p}{\leq} X$, then X and Y are **equivalent** and we write $X \underset{p}{=} Y$.

Example: A **vertex cover** in $G(V, E)$ is a set of vertices $S \subset V$ such that every edge in E has at least one endpoint in S. An **independent set** in G is a set of vertices $S \subset V$ such that no two vertices in S share an edge.

Claim 1 *A set S is a vertex cover iff its complement $V - S$ is an independent set.*

Corollary 1 *Max. indep. set $\underset{p}{=}$ Min. vertex cover.*

Instead of posing problems as optimization problems, we pose problems as yes/no or **decision** problems. An algorithm for a decision problem takes an input string s (a **certificate**) and returns either 1 or 0 depending on whether the input satisfies the constraints of the decision problem. Posing questions as decision problems provides a common framework and makes reductions between problems conceptually cleaner. Some examples of decision problems:

- Is the length of longest path $\leq k$?
- Is it possible to color a given graph with 3 colors?

Generally, a black box that solves a decision problem can also solve the related optimization problem. For example if we know that the solution is $\leq k$, we can find the exact value through binary search.

1.2 Problem Classes

Formally, we can represent a particular instance of a decision problem as an input string s. s encodes all of the information about the instance of the problem we are interested in. Let X be the set of all problem instances represented by strings s that are solvable. Then the decision problem asks: is $s \in X$? We define two classes of decision problems as follows.

P: the class of decision problems that can be correctly decided in polynomial time. In other words, there exists some deterministic **verifier** $A(s)$ that will return T iff $s \in X$ in polynomial time.

NP: the class of all decision problems for which a YES certificate can be verified in polynomial time. More formally, a problem is in NP if there exists a polynomial-time deterministic verifier $B(s,t)$ such that if $s \in X$, then there exists some $t : |t| \leq |s|$ (called a **certificate**) such that $B(s,t)$ returns T, otherwise (if $s \notin X$) $B(s,t)$ returns F for all t.

Co-NP: the class of all decision problems for which a NO certificate can be verified in polynomial time.

NP-complete: the class of decision problems $X \in$ NP such that $\forall\ Y \in$ NP, $Y \underset{p}{\leq} X$. We prove in the next section that this class is indeed non-empty.

Theorem 1 *$P \subseteq NP$*

Proof: This is straightforward from the definitions of P and NP. Given a polynomial verifier $A(s)$, we can construct a NP verifier $B(s,t)$ simply by discarding t and running $A(s)$. If $s \in X$, then $B(s,t)$ will return T for all certificates t, otherwise it will return false. Note that this does not say whether t is a "solution" of problem s (as respect to some arbitrary verifier $B'(s,t)$), just that we can construct some a verifier $B(s,t)$ that fits the requirements for NP. ■

1.3 NP-Complete Problems

Stephen Cook was the first to demonstrate the existence of NP-complete problems. Following his steps, we begin by defining the **circuit satisfiability** problem: given a circuit of logical operators (e.g. AND, NOT, OR), is there an assignment of T or F to the input variables that causes the circuit to output T?

Theorem 2 *Circuit satisfiability is NP-complete.*

A circuit can be viewed as a tree T, where each leaf is a variable, and each internal node is a logical operation (AND, OR, NOT). The idea is that for any problem in NP, we can by definition take as input an assignment of variables and answer in a polynomial number of operations whether or not this assignment is valid or not. Cook's contribution was to show that for any problem, this verification process can be encoded as a polynomial-sized circuit. In other words, if $X \in NP$, then we can design a polynomial-sized circuit Y such that for a given variable assignment x, Y outputs true if and only if x is a valid certificate for X. Then if we have a black box that solves the circuit satisfiability instance Y, i.e. decide whether there is an assignment to the variables so that the answer is T, then we can decide the original problem X.

Finding the first NP-complete problem made it much easier to establish NP-completeness of other problems. In order to prove that $Y \in NP$ is NP-complete, it is sufficient to show that circuit-SAT $\leq_p Y$, i.e. that we can reduce any circuit satisfiability instance to Y.

A circuit that is a series of OR clauses joined by AND clauses is said to be in **conjunctive normal form** (CNF). For example:

$$(x_1 \wedge \bar{x}_2 \wedge \bar{x}_3) \vee (\bar{x}_2 \wedge x_1) \vee (x_4 \wedge x_5)$$

Every circuit can be rewritten in CNF using the rules of logical equivalence. That fact allows us to establish a second NP-complete problem.

Satisfiability (SAT): Given a boolen formula in conjuntive normal form, is there a logical assignment such that the formula is satisfied?

It is easy to see that

- SAT $\in$ NP, we can verify a solution in polynomial time.
- SAT $\underset{p}{\geq}$ circuit-SAT, we can solve circuit-SAT using reduction to SAT.

Therefore, SAT is NP-complete.

3-SAT: A SAT problem in which every clause has exactly 3 literals.

Claim 2 *Every SAT formula of size n can be described as a 3-SAT formula of size polynomial in n.*

Proof: We can split any clause C of size greater than 3 into two clauses, the first with 2 variables from C, the second with $k-2$ variables from C. We introduce a dummy variable d_c into the first new clause, and its complement $\bar{d}_c$ into the second clause. We note that an assignment satisfies the two new clauses iff it satisfies the original clause. We may repeat this splitting process until all clauses are of size 3. ■

Corollary 2 *3-SAT $\in$ NP-C.*

Theorem 3 *3-SAT $\underset{p}{\leq}$ Maximum independent set*

Proof: For each clause C_i, construct a 3-cycle with vertices v_{i1}, v_{i2}, v_{i3} representing the 3 variables in clause i, say x_j, x_k, x_l. For each node representing variable x_j, put an edge between it and each vertex representing its complement $\bar{x}_j$. Let G be the graph constructed in this way.

Given a satisfying assignment to $C = \{C_1, \ldots, C_k\}$, we can construct an independent set of size k in G by picking one true variable in each C_j. Also, given an independent set of size k in G, we can reconstruct a satisfying assignment to C by setting the variables corresponding to the independent set vertices to be T and assigning the rest arbitrarily. Therefore, C has a satisfying assignment if and only if G has an independent set of size $\geq k$. ■