Reductions and $\mathcal{NP}$–Completeness

Reductions

If L_1, L_2 are languages over the alphabets Σ_1 and Σ_2, respectively, a *reduction* from L_1 to L_2 is function $R : \Sigma_1^* \to \Sigma_2^*$ such that $R(w) \in L_2$ if and only if $w \in L_1$. We write $L_1 \leq L_2$ to mean that there is a *recursive* (computable) reduction of L_1 to L_2, and we write $L_1 \leq_{\mathcal{P}} L_2$. if there is a $\mathcal{P}$–TIME reduction.

Reductions are used often in practice to shortcut calculations. A problem that can be easily reduced to an easy problem is easy.

Remark 1 *If $L_1 \leq_{\mathcal{P}} L_2$ and L_2 is $\mathcal{P}$, then L_1 is $\mathcal{P}$.*

Proof: Let $F : L_2 \to \{0, 1\}$ be a polynomial time function which decides L_2 and R a polynomial time reduction of L_1 to L_2. The composition $F \circ R$ decides L_1 in polynomial time. □

Instances. A reduction from a problem $P_1 \subseteq \Sigma_1^*$ to a problem $P_2 \subseteq \Sigma_2^*$ need only be defined on instances of P_1, since we define $R(w) = \lambda$ if w is not an instance of P_1 Reductions found in the literature or on the internet (or in my class) are typically defined only on instances.

A language L is in the class $\mathcal{P}$–TIME (or simply $\mathcal{P}$) if there is some constant k and some machine[1] which decides L in $O(n^k)$ time, where n is the number of bits of input.

A language L is in the class $\mathcal{NP}$–TIME (or simply $\mathcal{NP}$) if there is some constant k and some non-deterministic machine which accepts L is polynomial time. The flow chart of that non-determinstic computation is a binary tree of height $O(n^k)$, where the input is accepted at at least one leaf. Thus, L is accepted by some deterministic machine in exponential time; simply try every path through the computation tree! Every deterministic machine is also a non-deterministic machine, hence $\mathcal{P} \subseteq \mathcal{NP}$. The converse is an open question.

Verification Definition of $\mathcal{NP}$

A language L is $\mathcal{NP}$ if and only if there is some machine V some integer k such that:

1. For every $w \in L$ there exists a string c, called a *certificate* for w, such that V accepts the string (w, c) in $O(n^k)$ time.

2. If $w \notin L$ and c is any string, V does not accept the string (w, c).

[1]Deterministic, unless otherwise specified.

$\mathcal{NP}$–Completeness

We define a language L to be $\mathcal{NP}$*–complete* if

1. $L \in \mathcal{NP}$, and
2. Every $\mathcal{NP}$ language reduces to L in polynomial time.

If any given $\mathcal{NP}$–complete problem is $\mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$, as stated by Theorem 1 below.

Theorem 1 *If there is any language which is both $\mathcal{P}$–TIME and $\mathcal{NP}$–complete, then $\mathcal{P} = \mathcal{NP}$.*

Proof: Suppose that there is a language L_1 which is both $\mathcal{P}$–TIME and $\mathcal{NP}$–complete. Trivially, $\mathcal{P} \subseteq \mathcal{NP}$. We need to show that $\mathcal{NP} \subseteq \mathcal{P}$. Let L_2 be $\mathcal{NP}$, then $L_2 \leq_{\mathcal{P}} L_1$ by the definition of $\mathcal{NP}$-completeness. Since L_1 is $\mathcal{P}$, L_2 is $\mathcal{P}$ by Remark 1. □

Boolean Satisfiability

We define a *Boolean expression* to be an expression involving variables and operators, where all variables have Boolean type and all operators have Boolean operands. To shorten our notation, we use "+" for *or*, "·" for *and*, and "!" for *not*. An *assignment* of a Boolean expression E is an assignment of truth values (there are only two truth values, *true* $= 1$ and *false* $= 0$) to each variable that appears in E. That assignment is *satisfying* if given those values, E is *true*. E is *satisfiable* if it has a satisfying assignment, otherwise E is a *contradiction*. For example, $x{\cdot}!x$ is a contradiction, since its value is false regardless of the value of x. A satisfiable expression can have assignments that are not satisfying, such as $x{+}!y$, which has three satisfying asignments and one assignment that is not satisfying, namely $x = 0, y = 1$. Any satisfying assignment of $E \in$ SAT is a certificate for E.

Let BOOL be the set of all Boolean expressions, which is a context-free language, and let SAT $\subseteq$ BOOL be the satisfiable expressions.

Theorem 2 (Cook-Levin) *SAT is $\mathcal{NP}$–complete.*

Theorem 3 *If L_1 is $\mathcal{NP}$-complete and L_2 is $\mathcal{NP}$, and there is a polynomial reduction R_1 of L_1 to L_2, then L_2 is $\mathcal{NP}$–complete.*

Proof: We need only prove that every $\mathcal{NP}$ language reduces to L_2 in polynomial time. Let $L_3 \in \mathcal{NP}$. Since L_1 is $\mathcal{NP}$-complete, there is a polynomial time reduction R_2 of L_3 to L_1. The composition $R_2 \circ R_1$ is a polynomial time reduction of L_3 to L_2. □

SAT, or Boolean Satisfiability, is the "granddaddy" $\mathcal{NP}$–complete problem. Here are some reductions that give you additional $\mathcal{NP}$–complete problems.

A Boolean expression is in *conjuctive normal form* if it is the conjunction (and) of *clauses*, each of which is the disjunction (or) of *terms*, each of which is either a variable or the negation (not) of a variable. CNF $\subseteq$ BOOL is the set of all Boolean expressions written in conjunctive normal form, while k-CNF $\subseteq$ CNF is the subset where each clause has k terms.

1. For any $k \geq 3$, SAT $\leq_{\mathcal{P}}$ k-SAT: thus k-SAT is $\mathcal{NP}$-complete.
2. The *Independent Set* problem, IND is $\mathcal{NP}$-complete, since 3-SAT $\leq_{\mathcal{P}}$ IND.
3. The *subset sum*–problem, SS is $\mathcal{NP}$–complete, since INDT $\leq_{\mathcal{P}}$ SS.
4. The *Partition Problem* is $\mathcal{NP}$-complete, since SST $\leq_{\mathcal{P}}$ Partition.

The Independent Set Problem

Given a graph G an *independent set* of G is defined to be a set $\mathcal{I}$ of vertices of G such that no two members of $\mathcal{I}$ are connected by an edge of G. The *order* of $\mathcal{I}$ is defined to be its size, *i.e.*., simply how many vertices it contains.

An instance of the *independent set problem* is $\langle G\rangle\langle k\rangle$, where G is a graph and k is an integer. The question is, "Does G have an independent set of order k?"

The language IND We define IND to be the set of all $\langle G\rangle\langle k\rangle$ such that G has an independent set of order k.

Theorem 4 *IND is $\mathcal{NP}$ complete.*

Proof: Let E $\in$ 3-SAT. Then $e = C_1 \cdot C_2 \cdot \cdots \cdot C_k$, where $C_i = (t_{i,1} + t_{i,2} + t_{i,3})$, where each $t_{i,j}$ is either x or $!x$, where x is a variable.

We now define a graph $G[E] = (V, E)$, where $V = \{v_{i,j} : 1 \leq i \leq k, 1 \leq j \leq 3\}$ is the set of vertices of $G[e]$, and E the set of edges of $G[E]$, as follows:

1. For each $1 \leq i \leq k$, there is an edge from $v_{i,j}$ to $v_{i,j'}$ for all $1 \leq j < j' \leq 3$. Call these *short* edges.
2. If $t_{i,j} = x$ and $t_{i',j'} = !x$ for some variable x, there is an edge from $v_{i,j}$ to $v_{i',j'}$. Call these *long* edges.
3. There are no other edges.

Let $R(e) = G[e], k)$ We now show that $R(e) \in$ IND if and only if e is satisfiable. For each i, let K_i be the subgraph of $G[e]$ consisting of the three vertices $v_{i,1}$, $v_{i,2}$, $v_{i,3}$, and the edges connecting them. We call this a *3-clique.*

Suppose $G[e], k \in$ IND. Let $I \subset V$ be an independent set of of size k. Since K_i is a 3-clique, and the number of such cliques is equal to k, exactly one member of I must lie in each K_i.

We define an assignment of e. If $v_{i,j} \in I$ and $t_{i,j} = x$ for some variable x, assign the value *true* to x, while if $t_{i,j} =!x$, assign *false* to x. Assign all remaining variables arbitrary Boolean values. This assignment is well-defined, for if $v_{i,j}, v_{i',j'} \in I$ for $i \neq i'$, there can be no edge between those two vertices, which implies that $t_{i,j}$ does not contradict $t_{i',j'}$. Furthermore, each clause has one term which is assigned true, hence each clause is assigned true, and we thus the assignment is satisfying.

Conversely, suppose that there is a satisfying assignment of e. That means each clause C_i must contain one term, say $t_{i,j[i]}$ which is true under the assignment. Let $I = \left\{ v_{i,j[i]} \right\} \subseteq V$. No two elements of I are in the same clique K_i, hence there is no short edge connecting them, and there can be no long edge connecting them because $v_{i,j[i]}$ and $v_{i',j[i']}$ are both assigned true and hence cannot contradict each other. Thus I is an independent set. □

Example

A non-trivial example would have at least eight clauses, but I'll keep it simple. Let E be the 3CNF expression

$$(x + y + z) \cdot (!x + !y + w) \cdot (y + !z + !w) \cdot (!y + z + !w)$$

Then $k = 4$. The following diagram illustrates $G[E]$. The vertices of I are circled in red. The satisfying assignment shown is $x =$ false, $y =$ true, $w =$ false, while z can be assigned either true or false.