Reductions and $\mathcal{NP}$–Completeness

Non-Deterministic Polynomial Time Languages

A language L is in the class $\mathcal{NP}$–TIME (or simply $\mathcal{NP}$) if there is a non-deterministic machine M which accepts L in time which is polynomial in the number of bits of its input.[1] The computation tree of M given an input w is a binary tree whose height is a polynomial function of $|w|$, the number bits required to write w. The input is *accepted* by if M is in an accepting state at at least one leaf of the computation. Thus, L can be decided by a deterministic machine in exponential time, by simply exploring the entire computation tree. But could we determine the answer in polynomial time? That is an unsolved problem, the famous "$\mathcal{P} = \mathcal{NP}$" problem.

Verification Definition of $\mathcal{NP}$

A language L is $\mathcal{NP}$ if and only if there is some machine V and 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, where $n = |w|$.
2. If $w \notin L$ and c is any string, V does not accept the string (w, c).

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 $L_1 \leq_R L_2$. We say "L_1 reduces to L_2." If R is $\mathcal{P}$–TIME, we write $L_1 \leq_{\mathcal{P}} L_2$. 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 $\mathcal{P}$.*

Instances. A reduction from a problem to another need only be defined on instances of of the first problem. since we can let $R(w) = \lambda$ if w is not an instance. Reductions are typically defined only on instances.

$\mathcal{NP}$–Completeness

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

[1] We can assume that at any step, M has at most two legal choices.

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

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. Let Let L_2 be any $\mathcal{NP}$ language 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

Many $\mathcal{NP}$–COMPLETE problems (languages) have been identified, and the number grows constantly. The first such problem identified is SAT, Boolean satisfiability, proved $\mathcal{NP}$–complete by Theorem 2, the Cook Levin theorem. Using that theorem and Theorem 3, many additional $\mathcal{NP}$–complete problems have been found.

Let Bool be the languages of all *Boolean expressions*, defined to be expressions consisting of variables and operators, where all variables have Boolean type and all operators are Boolean. To shorten our notation, we use "+" for *or*, "·" for *and* and "!" for *not*. We also use $\Rightarrow$ for implies, $=$ for equality, and $\neq$ for inequality. 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. An 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 assigned value of x, while $x \cdot !y$ is satisfiable, because the assignment $x = 1$, $y = 0$ is satisfying. Let SAT $\subseteq$ BOOL be the satisfiable expressions. We also write SAT to be the problem of determining whether $E \in$ SAT. Any satisfying assignment of a E is a certificate which verifies that $E \in$ SAT.

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

The proof of Theorem 2 is long, but straightforward. You can find it in books or on the internet.

Theorem 3 (Bootstrap) *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. □

Starting with SAT, we use the Bootstram theorem to prove a sequence of problems $\mathcal{NP}$ complete:

$$SAT \leq_{\mathcal{P}} 3-SAT \leq_{\mathcal{P}} IND \leq_{\mathcal{P}} SubsetSum \leq_{\mathcal{P}} Partition$$

k-SAT

A Boolean expression is in CNF, *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 at most k terms.

Note that k-CNF $\subseteq$ CNF $\subseteq$ BOOL.

We define k-SAT = k-CNF $\cap$ SAT.

We prove 3-SAT $\mathcal{NP}$-complete by reduction from SAT. For $k > 3$, k-SAT is a special case of 3-SAT, hence reduces trivially to 3-SAT, and thus is also $\mathcal{NP}$-complete.

2-SAT is $\mathcal{P}$–TIME, and thus is not known to be $\mathcal{NP}$-complete. This theorem is a challenge problem for CS656.

Independent Set

An instance of the independent set problem, abbreviated IND, is an ordered pair (G, K) where G is a graph and K is a positive integer. We say a set of vertices of G is *independent* if no two are connected by an edge. A solution (certificate) of (G, K) is an independent set of K vertices of G, thus IND is $\mathcal{NP}$ by the verification definition of $\mathcal{P}$. We give a polynomial time reduction R of 3-SAT to IND. We define R only on 3-CNF, the language of instances of 3-SAT. By Theorem 3, IND is $\mathcal{NP}$-complete. We give a polynomial time reduction of IND to subset sum, and thus subset sum is $\mathcal{NP}$-complete.

Partition

An instance of the partition problem is a sequence $\tau = y_1, \ldots y_k$ of positive numbers. A solution to τ is a subsequence of τ whose sum is half the sum of the terms of τ, which is an easily verified certificate. We give a polynomial time reduction of subset sum to partition, proving jartition is $\mathcal{NP}$-complete.

Polynomial Time Reduction of 4-SAT to 3-SAT

We preface our chain of reductions by proving a simpler special case of the reduction of SAT to 3-SAT, namely a $\mathcal{PN}$–TIME reduction of 4-SAT to 3-SAT.

Let $E_1 = C_{1,1} \cdot C_{1,2} \cdots C_{1,n}$ be a 4-CNF Boolean expressions. consisting of n 4-clauses. We construct a 3-CNF expression $R(E_1) = E_2 = C_{2,1} \cdot C_{2,2} \cdots C_{2,2n-1} \cdot C_{2,2n}$ consisting of $2n$ 3-clauses, which is equisatisfiable with E_1.

Let $C_{1,i} = t_{i,1} + t_{i,2} + t_{i,3} + t_{i,4}$, where each term $t_{i,j}$ is either a variable or the negation of a veriable. For each $1 \leq i \leq n$, let u_i be a variable that does not appear in E_1. Now define:

$C_{2,2i-1} = u_i + t_{i,1} + t_{i,2}$

$C_{2,2i} = !u_i + t_{i,3} + t_{i,4}$

Claim: E_1 is satisfiable if and only if E_2 is satisfiable.

Proof: Suppose there is a satisfying assignment of E_2. Then, each clause of E_2 must be true. For any i, if u_i is false, then $t_{i,1} + t_{i,2}$ must be true, while if u_i is true, $t_{i,3} + t_{i,4}$ must be true. In either case, $C_{1,i}$ is true, and hence E_1 is satisfied.
Conversely, suppose there is a satisfying assignment of E_1. For each i, one of the four terms of E_i must be true. If $t_{i,1} + t_{i,2}$ is true we assign u_i to be false, while otherwise we assign u_i to be true. In either case, both $C_{1,2i-1}$ and $C_{1,2i}$ are true, and thus L_2 is satisfied. □

Polynomial Time Reduction of SAT to 3-SAT

Theorem 4 *3-SAT is $\mathcal{NP}$-complete.*

Theorem 4 is proven by giving a polynomial time reduction R of SAT to 3-SAT. Given a Boolean expression e, $R(e)$ is a Boolean expression in 3-CNF form which is satisfiable of and only if e is satisfiable. We say that e and $R(e)$ are *equisatisfiable*. Two Boolean expressions are usually said to be equal if each can be obtained from the other by using the identity rules of Boolean algebra. However, it is not necessary to show that e and $R(e)$ are equal in that sense. Instead, $R(e)$ makes use of all the variables of e, together with new variables.

We first define a context-free grammar for BOOL. There are many Boolean operators in use, but we will only use five. The start symbol of the grammar is E, for "expression." For simplicity, we use the symbol **id** to stand for any identifier.
$E \to E + E$ (or)
$E \to E \cdot E$ (and)
$E \to !E$ (not)
$E \to E \Rightarrow E$ (implies)
$E \to E \equiv E$ (equivalent, same truth value as)
$E \to E \not\equiv E$ (not equivalent)
$E \to (E)$ (this is not actually an operation)
$E \to$ **id**

Note that the right hand side of each of the first five productions yields a Boolean expression if each instance of E is replaced by a variable.

in Table 1 below, we give a reduction of each of certain exressions, that we use in the proof, to 3-SAT.

$u \equiv (x + y)$	$(!u + x + y) \cdot (u+!x) \cdot (u+!y)$
$u \equiv (x \cdot y)$	$(!u + x) \cdot (!u + y) \cdot (u+!x+!y)$
$u \equiv !x$	$(u + x) \cdot (!u+!x)$
$u \equiv (x \Rightarrow y)$	$y+!x$
$u \equiv (x \equiv y)$	$(!u+!x + y) \cdot (!u + x+!y) \cdot (u+!x+!y) \cdot (u + x + y)$
$u \equiv (x \not\equiv y)$	$(!u+!x+!y) \cdot (!u + x + y) \cdot (u+!x+!y) \cdot (u + x+!y)$

Table 1

Next, we construct a parse tree T for e. Each leaf is either an operator or xi for some i. We label the internal nodes of T with new variables $u1, u2, \ldots um$. For each internal node, we obtain an expression which is an instance of one of the expressions in Table 1, and we reduce e to the conjunction of those expressions and the root variable.

As an example, we let $e = (x1+!x2) \Rightarrow (!x1 \cdot x3)$. The parse tree T is shown below.

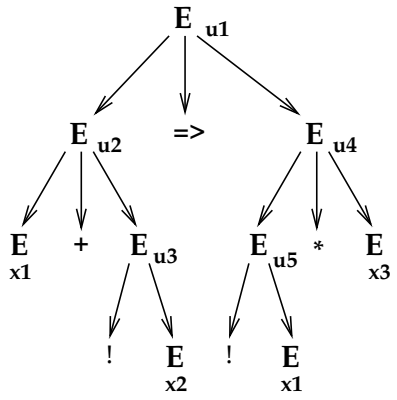


Figure 1

Our next step is to translate the parse tree into the conjunction of clauses, one for each internal node, plus one for the root. In our example, we obtain:

$(u1) \cdot (u2 = x1 + u3) \cdot (u3 =!x2) \cdot (u5 =!x1) \cdot (u4 = u5 \cdot x3) \cdot (u1 = u2 \Rightarrow u4) \cdot$

Finally, we translate that conjunction into 3-CNF form by replacing each of its clauses by a conjunction of CNF clauses of at most three terms, using Table 1:

$(u1) \cdot (u1 + u2) \cdot (u1+!u4) \cdot (!u1+!u2 + u4) \cdot (u2+!x1) \cdot (u2+!u3) \cdot (!u2 + x1 + u3) \cdot (u3 + x2) \cdot$
$(!u3+!x2) \cdot (u5 + x1) \cdot (!u5+!x1) \cdot (u4+!u5+!x3) \cdot (!u4 + u5) \cdot (!u4 + x3)$

If we are required to have exactly three terms in each clause, we can "pad" each clause of length less than three by duplicating terms. For example, we can replace $(u2+!x1)$ with

$(u2 + u2 + !x1)$, and $(u1)$ with $(u1 + u1 + u1)$.

Reduction of 3-SAT to IND

Theorem 5 *IND is $\mathcal{NP}$ complete.*

Proof: Let $E = C_1 \cdot C_2 \cdot \cdots \cdot C_K$ be a Boolean expression in 3-CNF form. We can assume each clause has exactly three terms, since we can pad a clause with duplicate terms. For example, we can replace $x + y$ with $x + x + y$, and x with $x + x + x$.

Let $C_i = t_{i,1} + t_{i,2} + t_{i,3}$, the disjunction of three terms. Thus, E contains $3K$ terms.

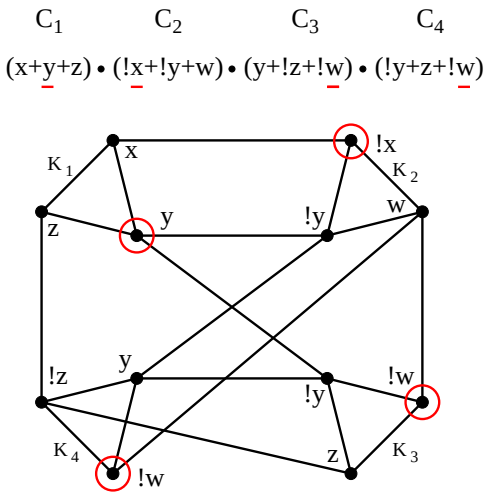
Define a graph G_E whose vertices are $\{x_{i,j} \ : \ 1 \leq i \leq j, \ j = 1, 2, 3\}$. That is there is a 1-1 correspondence between terms of E and vertices of G_E. The vertices $v_{i,1}, v_{i,2}, v_{i,3}$ form a 3-clique, a complete subgraph of G_E. Besides the clique edges, G_E has contradiction edges, namely an edge from any $v_{i,j}$ to $v_{i',j'}$ if $t_{i,j}$ contradicts $t_{i',j'}$. Thus, E is satisfiable if and only if G_E has an independent set of K vertices. □

Example

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



Subset Sum

An instance of the subset sum problem consists of a sequence of numbers $\sigma = x_1, \ldots x_k$, together with a number B. That instance has a solution if there is some subsequence of σ whose sum is B. Without loss of generality, we assume that the x_k are positive. A subset of sum B is an easily verified certificate, hence subset sum is $\mathcal{NP}$.

Theorem 6 *The subset sum problem is $\mathcal{NP}$-complete.*

Proof: Trivially, the subset sum problem satisfies the verifiability definition of $\mathcal{NP}$.

We reduce the independent set problem to the subset sum problem. Suppose $\langle G \rangle \langle k \rangle$ is an instance of the independent set problem, where G has n vertices and m edges. Let $e_0, \ldots e_{m-1}$ be the edges of G and $v_m \ldots v_{n+m-1}$ the vertices of G. Define $R(\langle G \rangle \langle B \rangle)$ to be the instance of the subset sum problem $w = (x_0, x_1, \ldots x_{n+m-1}, B)$ where

- $x_i = 4^i$ for $0 \leq i < m$
- For $m \leq i < n + m$, let $J_i = \{j : v_i \text{ is an endpoint of } e_j\}$, then $x_i = 4^m + \sum_{j \in J_i} 4^j$.
- $B = k4^m + \sum_{0 \leq j < m} 4^j$

For $0 \leq j < m$, x_j corresponds to the edge e_j, while for $m \leq i < m + n$, x_i corresponds to the vertex v_i.

We need to prove R is a reduction of IND to Subset Sum. Suppose $I \subseteq V$ is an independent set of k vertices of G. Let

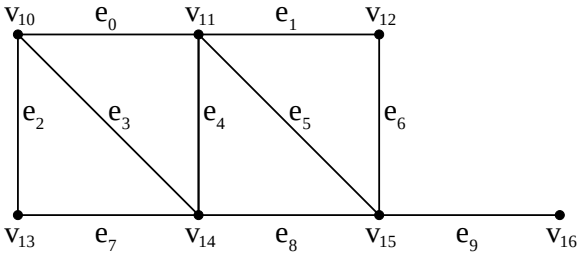
$$S = \{x_j : 0 \leq j < m \text{ and no endpoint of } e_j \text{ is in } I\} \cup \{x_i : m \leq i < n + m \text{ and } v_i \in I\}$$

We need to show that $\sum S$, the sum of the members of S, equals B. Since $|I| = k$, the coefficient of 4^m is k. An e_j is adjacent to either just one member of I or none. If e_j is adjacent to vertex $v_i \in I$, then the coefficient of 4^j in x_i is 1, matching that of B. Otherwise, S contains x_j, which is simply 4^j, hence $\sum S = B$.

Conversely, suppose there is a subset S of the sequence whose sum is B. Let $I = \{v_i : x_i \in S\}$. Since the coefficient of 4^m in B is k, I must contain exactly k vertices. Since the coefficient of every x_j in B is 1, no two members of the I can be adjacent, hence I is a solution to the subset sum problem. □

Example

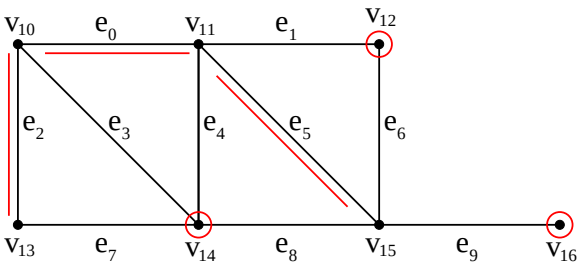
Let G be the graph shown below, and let $k = 3$.



We show the reduced instance of the subset sum problem, where the x_i are written in base 4.

$$
\begin{aligned}
x_0 &= 1 \\
x_1 &= 10 \\
x_2 &= 100 \\
x_3 &= 1000 \\
x_4 &= 10000 \\
x_5 &= 100000 \\
x_6 &= 1000000 \\
x_7 &= 10000000 \\
x_8 &= 100000000 \\
x_9 &= 1000000000 \\
x_{10} &= 10000001101 \\
x_{11} &= 10000110011 \\
x_{12} &= 10001000010 \\
x_{13} &= 10010000100 \\
x_{14} &= 10110011000 \\
x_{15} &= 11101100000 \\
x_{16} &= 11000000000 \\
B &= 31111111111
\end{aligned}
$$

The graph below shows an independent set I of vertices of order 3, together with the set of edges which are not adjacent to members of I.



Finally, we show the members of the subsequence corresponding to the chosen vertices and edges.

$$\begin{array}{rcr} x_0 & = & 1 \\ x_2 & = & 100 \\ x_5 & = & 100000 \\ x_{12} & = & 10001000010 \\ x_{14} & = & 10110011000 \\ x_{16} & = & 11000000000 \\ \hline B & = & 31111111111 \end{array}$$

Subset Sum and Partition

The subset sum problem can be shown to be $\mathcal{NP}$-complete by reducing IND to subset sum. We can then show that the partition problem is $\mathcal{NP}$-complete by reducing Subset Sum to Partition.

Recall that an instance of the Subset Sum problem is a number followed by a sequence of positive numbers, followed by one number, K, that is, $(x_1, x_2, \ldots x_m, K)$, and that instance is in L_{subs} if there is some subsequence of $x_1, \ldots x_m$ whose total is K. Let L_{subs} be the language of all instances of the problem which have a solution.

Similarly, an instance of the Partition problem is a sequence of positive numbers, namely $\langle y_1, \ldots y_\ell \rangle$, and there is a solution to that instance if and only if there is some subsequence of $y_1, \ldots y_\ell$ whose sum is half the total, *i.e.* $\frac{1}{2}\sum_{j=1}^{\ell} y_j$. Let L_{part} be the language of all instances of the problem which have a solution.

The partition problem is trivially in the class $\mathcal{NP}$, since the solution, if any, can be verified in polynomial time.

Reduction

Given an instance $I_K = (x_1, x_2, \ldots x_m, K)$ of Subset Sum, let $A = \sum_{i=1}^{m} x_i$. Let $R(I_K) = I_P$ be the following instance of Partition:

$$I_P = (x_1, \ldots x_m, K+1, A-K+1)$$

That is, $I_P = (y_1, \ldots y_\ell)$ where $\ell = m+2$, $y_i = x_i$ for $i \leq m$, $y_{m+1} = K+1$, and $y_{m+2} = A - K + 1$.

We need to prove that this reduction works, that is, that $I_P \in L_{\text{part}}$ if and only if $I_K \in L_{\text{subs}}$. There are thus two directions to the proof.

Note that the sum of the items of I_P is $2A + 2$, and half of that is $A + 1$.

Suppose that $I_K \in L_{\text{subs}}$. Then there is a subsequence of $\{x_i\}$ whose total is K. The items of this subsequence, together with $A - K + 1$, total $A + 1$, and thus $I_P \in L_{\text{part}}$.

Conversely, suppose some subsequence S of I_P which has total $A+1$. That subsequence cannot contain both $K+1$ and $A-K+1$, since their total exceeds A+1. S must contain either $K+1$ or $A-K+1$. Without loss of generality, S contains $A-K+1$ but not $K+1$. The remaining members of S constitute a subsequence of $x_1, \ldots x_m$ whose total is K, and we are done.