Introduction to NP-Completeness

An outline of concepts from Introduction to Algorithms by Thomas Cormen, et al.

Running Time of an Algorithm

• Thus far we have primarily viewed the running times of algorithms as a function of the number of items in the input. (Recall sorting, various greedy algorithms, divide and conquer algorithms, some dynamic programming algorithms). Where would 0-1 knapsack fit in?

• For studying NP and NP-completeness the size of the input will be the size of the encoding required to represent it.

Before defining NP, let’s look at some problems, a form of which, will be in the class.

• Given a set of n objects with weights and profits and a knapsack with capacity, M find a set of maximum profit that will fit in the knapsack. (0-1 Knapsack Problem)

• Given a graph G=(V,E) find the smallest subset C of V such that each edge in E has at least one endpoint in C. (Vertex Cover)

• Given a graph G=(V,E) find the largest subset C of V, such that every pair of vertices in C are adjacent in G. (Clique)
• Given a Boolean expression consisting of disjunctive clauses, is there a truth assignment of the variables such that \( C_1 \land C_2 \land \cdots \land C_m \) is true. \( C_i = x_{i_1} \lor x_{i_2} \lor x_{i_3} \lor \cdots \lor x_{i_k} \), \( x_{i_1} \) is a literal. (Satisfiability)

• Given a finite set \( S \subseteq \mathbb{N} \) and a target \( t \), find a set \( S' \subseteq S \) such that \( \sum_{a \in S'} a = t \) (Subset Sum)

• Given a graph \( G = (V,E) \) find a simple cycle in \( G \) containing every vertex in \( V \).

---

**Decision versus Optimization**

The examples on the previous two slides can all be easily transformed to a decision problem, that is either answered “yes” or “no”.

---

Before defining NP, let’s look at some problems that will be in the class.

• Given a set of \( n \) objects with weights and profits and a knapsack with capacity, \( M \) find a set of maximum profit that will fit in the knapsack. (0-1 Knapsack Problem)
  
  Given \( k > 0 \) is there a set with profit \( k \)?

• Given a graph \( G = (V,E) \) find the smallest subset \( C \) of \( V \) such that each edge in \( E \) has at least one endpoint in \( C \). (Vertex Cover) Given \( k > 0 \) is there a cover of size \( k \)?

• Given a graph \( G = (V,E) \) find the largest subset \( C \) of \( V \) such that every pair of vertices in \( C \) are adjacent in \( G \). (Clique) Given \( k > 0 \) is there a Clique of size \( k \)?
• Given a boolean expression containing n literals (x or ¬x), find an assignment of the boolean variables that will make the expression true. (Satisfiability) Is there a truth assignment satisfying the boolean expression.

• Given a finite set \( S \subseteq N \) and a target \( t \), find a set \( S' \subseteq S \) such that \( \sum_{a \in S'} a = t \) (Subset Sum) Is there a subset such that the elements in the subset sum to \( t \)?

• Given a graph \( G = (V,E) \) find a simple cycle in \( G \) containing every vertex in \( V \). Does \( G \) contain a Hamiltonian cycle?

Verification

Thus far, we haven’t said anything about how hard or easy it is to answer these decision problems.

They do all have something in common, however, namely, given a certificate (trial solution), it takes only polynomial time to test whether or not the certificate solves the problem. (Finding a satisfying certificate is another matter.)

0-1 Knapsack

Given:
• n objects with assigned profits and weights;
• a knapsack with capacity \( M \);
• a number \( k > 0 \)

A certificate could be any set of objects whose total weight is less than or equal to \( M \). Determining the total weight and profit and comparing that to \( k \) can be done in \( O(n) \) time.
Vertex Cover

Given:
• Graph G = (V,E);
• integer k > 0;

A certificate is any subset S of V of size k. One would then need to check whether or not every edge in E has an endpoint in S. Can this be done in $O(|E| + |V|)$?

Clique

Given:
• Graph G = (V,E)
• k > 0;

A certificate is a set $S \subseteq V$, $|S| = k$. Must then verify that every pair of vertices in S are the endpoints of an edge in E. $O(|V|^2 + |E|)$

Satisfiability

Given:
• A Boolean expression in $n$ Boolean variables $x_1, x_2, x_3, \ldots, x_n$ which is the conjunction of disjunctive clauses.

A certificate is an assignment of each of the variables $x_i$ to true or false. To check whether or not each clause has at least one true literal requires $O(\sum_{C \ in \ clauses} length(C) + n)$ time.
Subset Sum

Given:
• A set $S$ of natural numbers
• a target integer value $t$

A certificate is any subset $S' \subseteq S$. Does $\sum_{a \in S'} a = t$? Can be checked in polynomial time as long as we use an efficient encoding for integers (such as a binary encoding).

Hamiltonian Cycle

Given
• Graph $G = (V, E)$
• $V = \{v_1, v_2 \ldots v_n\}$

A certificate is a sequence of vertices $< v_{i_1}, v_{i_2}, \ldots v_{i_n} >$, $v_{i_1} = v_{i_n}$. Must check that $\forall j, 1 \leq j \leq n-1, e = (v_{i_j}, v_{i_{j+1}}) \in E$. How fast can this be done – using edge list? using adjacency matrix?

Abstract Problem

An abstract problem is defined as:

$Q = \{(I, S) | I \text{ is a problem instance and } S \text{ is a solution}\}$

Note that $Q$ is a relation (not necessarily a function from $I$ to $S$) on $I \times S$, since there might be more than one solution to a given problem instance.
Concrete Problem and Running Time

If we use an efficient coding and represent instances of a problem as binary strings, the problem is said to be a concrete problem.

An algorithm $\mathcal{A}$ is said to solve a concrete problem in $O(T(n))$ time, if given an instance $i$ of the problem, $|i| = n$, $\mathcal{A}$ finds a solution in $O(T(n))$ time.

Polynomial Algorithm and P

Considering the previous definition of running time, if $T(n) = O(n^k)$ for some $k > 0$, then the concrete problem is said to be *polynomial-time solvable*. The complexity class $P$ is the set of all concrete problems that are polynomial-time solvable.

What about the following? Minimum spanning tree; job sequencing with deadlines; task scheduling; fractional knapsack;

Was our dynamic programming solution to the 0-1 Knapsack problem polynomial? Why or why not?

Efficient Encodings

How an instance is encoded matters. Here is a simple example: Suppose integer $k$ is the sole input to an algorithm $\mathcal{A}$, and the running time on this input is $\Theta(k)$. If $k$ is represented in unary (represented by $k$ 1s) then the running time is $\Theta(n)$ on inputs of length $n$. But if $k$ is represented in binary the input length is $n = \lceil \log_2(k) \rceil + 1$. Based on the length of the input, the running time is now $\Theta(2^n)$ - exponential.
\{0, 1\}^*

This is the set of all finite strings of 0’s and 1’s.

Suppose \(I\) is a set of problem instances. An encoding is a mapping \(e:I \rightarrow \{0, 1\}^*\)

Polynomially Related Encodings

Suppose that \(f: \{0,1\}^* \rightarrow \{0,1\}^*\) and \(g: \{0,1\}^* \rightarrow \{0,1\}^*\) and each can be computed in polynomial time (on the length of the input string).

If \(e_1\) and \(e_2\) are two encodings for an instance set \(I\), such that \(f(e_1) = e_2\) and \(g(e_2) = e_1\), then the encoding are said to be polynomially related.

Fact

If an abstract problem \(Q\) has polynomially related encodings \(e_1\) and \(e_2\), then \(e_1(Q)\) can be solved in polynomial time if and only if \(e_2(Q)\) can be solved in polynomial time.

In other words \(e_1(Q) \in P \iff e_2(Q) \in P\)
Algorithm Acceptance

An algorithm $A$ accepts a string $x \in \{0,1\}^*$ if given $x$ as input, $A$ outputs 1. If the output is 0, then $A$ is said to reject $x$.

$L = \{x \mid A\text{accepts } x\}$ is called the language accepted by $A$. Note:
If $x \notin L$, $A$ may not reject $x$, since $A$ could loop forever.

Decidability

$A$ is said to decide a language $L \subseteq \{0,1\}^*$ if given a string $x \in \{0,1\}^*$ $A$ produces 1 for $x \in L$ and 0 for $x \notin L$.

Deciding a language is much stronger than accepting a language.
If $A$ runs in $O(n^k)$ time for strings of length $n$, then the language $L$ is said to be decided in polynomial time.

Complexity Class P

$P = \{L \subseteq \{0,1\}^* \mid \exists \text{ an algorithm } A \text{ deciding } L \text{ in polynomial time.}\}$

FACT:
$P = \{L \mid L \text{ is accepted by a polynomial time algorithm}\}$

How can you see this?
The Class NP

A language L belongs to the class NP if and only if there exists a two input polynomial-time algorithm \( A \) and a constant \( c \) such that:

\[
L = \{ x \in \{0,1\}^* \mid \exists y \in \{0,1\}^*, |y| = O(|x|^c) \text{ such that } A(x, y) = 1 \}
\]

\( A \) is said to verify the language in polynomial time. This does not say anything about finding the certificate \( y \).

Definition of Reducibility

A language \( L_1 \) is polynomially reducible to \( L_2 \) if there exists a polynomial-time function \( f: \{0,1\}^* \rightarrow \{0,1\}^* \) such that \( x \in L_1 \iff f(x) \in L_2 \). This is denoted: \( L_1 \leq_P L_2 \).

Question

If \( L_1 \leq_P L_2 \) and \( L_2 \in P \) what do you think can be said about \( L_1 \)?
NP – Complete (NPC)

A language $L$ which is in NP and which is at least as hard as any other language in NP, is called NP complete. More precisely:

$L \subseteq \{0,1\}^*$ is NP-complete if:
$L \in \text{NP}$ and
if $L' \in \text{NP}$ then $L' \leq_P L$

History

In looking at the definition of NP-complete, a question arises: “Is there at least one problem that is in NP and is as hard as any other problem in the class?”
The existence of such a problem was established in 1971 by S. A. Cook, who showed that satisfiability is NP-complete.
The decision problem is: Given a set $V$ of Boolean variables and a set of clauses $C$, is there a truth assignment for the variables in $V$ satisfying all clauses? - $(C_1 \lor C_2 \lor C_3 \lor \cdots \lor C_N \text{ is true})$ where each clause is the disjunction of literals (at least one literal in each clause must be true).

Showing NP-Completeness

Suppose $L$ is known to be NP-Complete. If $L'$ is in NP and $L \leq_P L'$ is it true that $L'$ is NP-Complete? Why?