The Theory of NP-Completeness

Tractable and intractable problems
NP-complete problems
The theory of NP-completeness

- Tractable and intractable problems
- NP-complete problems
Classifying problems

- Classify problems as tractable or intractable.

- Problem is *tractable* if there exists at least one polynomial bound algorithm that solves it.

- An algorithm is *polynomial bound* if its worst case growth rate can be bound by a polynomial $p(n)$ in the size $n$ of the problem

$$p(n) = a_n n^k + ... + a_1 n + a_0 \text{ where } k \text{ is a constant}$$
Intractable problems

• Problem is *intractable* if it is not tractable.

• **All** algorithms that solve the problem are not polynomial bound.

• It has a worst case growth rate $f(n)$ which cannot be bound by a polynomial $p(n)$ in the size $n$ of the problem.

• For intractable problems the bounds are:

$$f(n) = c^n, \text{ or } n^\log n, \text{ etc.}$$
Why is this classification useful?

- If problem is intractable, no point in trying to find an efficient algorithm
- All algorithms will be too slow for large inputs.
Intractable problems

• Turing showed some problems are so hard that no algorithm can solve them (undecidable)

• Other researchers showed some decidable problems from automata, mathematical logic, etc. are intractable

No problem is known for certain to be in \( NP - P \)

Halting Problem is in here
Hard practical problems

• There are many practical problems for which no one has yet found a polynomial bound algorithm.

• Examples: traveling salesperson, 0/1 knapsack, graph coloring, bin packing etc.

• Most design automation problems such as testing and routing.

• Many networks, database and graph problems.
• If there are only 2 cities then the problem is trivial, since only one tour is possible. For the n cities, if all links are present (i.e., the graph is complete), then there are (n-1)! different tours.

• To see why this is so, pick any city as the first - then there are n-1 choices for the second city visited, n-2 choices for the third, and so on.

• The number of solutions becomes extremely large for large n, so that an exhaustive search is impractical.
How are they solved?

• A variety of algorithms based on backtracking, branch and bound, dynamic programming, etc.

• None can be shown to be polynomial bound
The theory of NP completeness

• The theory of NP-completeness enables showing that these problems are at least as hard as *NP-complete* problems

• Practical implication of knowing problem is NP-complete is that it is *probably* intractable (whether it is or not has not been proved yet)

• So any algorithm that solves it will probably be very slow for large inputs
We will need to discuss

- Decision problems
- Converting optimization problems into decision problems
- The relationship between an optimization problem and its decision version
- The class P
- Verification algorithms
- The class NP
- The concept of polynomial transformations
- The class of NP-complete problems
Decision Problems

• A decision problem answers yes or no for a given input

• Examples:
  – Given a graph $G$, is there a path from $s$ to $t$ of length at most $k$?
  – Does graph $G$ contain a Hamiltonian cycle?
  – Given a graph $G$ is it bipartite?
A decision problem: HAMILTONIAN-CYCLE

• A Hamiltonian cycle of a graph $G$ is a cycle that includes each vertex of the graph exactly once.
• Problem: Given a graph $G$, does $G$ have a Hamiltonian cycle?
Converting to decision problems

- Optimization problems can be converted to decision problems (typically) by adding a bound $B$ on the value to optimize, and asking the question:
  - Is there a solution whose value is at most $B$? (for a minimization problem)
  - Is there a solution whose value is at least $B$? (for a maximization problem)
An optimization problem: traveling salesman

• Given:
  – A finite set $C=\{c_1, \ldots, c_m\}$ of cities,
  – A distance function $d(c_i, c_j)$ of nonnegative numbers.
• Find the length of the **minimum** distance tour which visits every city exactly once and comes back to the starting city
A decision problem for traveling salesman (TS)

• Given a finite set $C=\{c_1, \ldots, c_m\}$ of cities, a distance function $d(c_i, c_j)$ of nonnegative numbers and a bound $B$

• Is there a tour of all the cities (in which each city is visited exactly once) with total length at most $B$?

• There is no known polynomial bound algorithm for TS.
Relation between an optimization problem and the decision problem

- If we have a solution to the optimization problem, we can compare the solution to the bound and answer “yes” or “no”.

- Therefore, if the optimization problem is tractable so is the decision problem.

- If the decision problem is “hard” the optimization problem is also “hard”
  - If the optimization was easy then the decision problem is easy.
The class $P$

- $P$ is the class of decision problems that are polynomial bounded

- Is the following problem in $P$?
  - Given a weighted graph $G$, is there a spanning tree of weight at most $B$?

- The decision versions of problems such as shortest distance path and minimum spanning tree belong to $P$
The goal of verification algorithms

• The goal of a verification algorithm is to verify a “yes” answer to a decision problem’s input (i.e., if the answer is “yes” the verification algorithm verifies this answer)

• The inputs to the verification algorithm are:
  – the original input (problem instance) and
  – a certificate (possible solution)
Verification Algorithms

• A verification algorithm, takes a problem instance x and answers “yes”, if there exists a certificate y such that the answer for x with certificate y is “yes”

• Consider HAMILTONIAN-CYCLE
• A problem instance x lists the vertices and edges of G: (\{1,2,3,4\}, \{(3,2), (2,4), (3,4), (4,1), (1, 3)\})
• There exists a certificate y = (3, 2, 4, 1, 3) for which the verification algorithm answers “yes”
Polynomial bound verification algorithms

- Given a decision problem \( d \).

- A verification algorithm for \( d \) is *polynomial bound* if given an input \( x \) to \( d \), there exists a certificate \( y \), such that \(|y| = O(|x|^c)|\) where \( c \) is a constant, and a polynomial bound algorithm \( A(x, y) \) that verifies an answer “yes” for \( d \) with input \( x \).

Note: \(|y|\) is the size of the certificate, \(|x|\) is the size of the input.
The problem PATH

- PATH denotes the decision problem version of shortest path.
- PATH: Given a graph $G$, a start vertex $u$, and an end vertex $v$. Does there exist a path in $G$, from $u$ to $v$ of length at most $k$?

- The instance is: $G=\{\text{A, B, C, D}\}$, $\{(\text{A, C, 2}), (\text{A, D, 15}), (\text{C,D, 3}), (\text{D, B, 1})\}$ $k=6$
- A certificate $y=(\text{A, C, D, B})$
A verification algorithm for PATH

- Verification algorithm:
  - Given the problem instance $x$ and a certificate $y$
    - Check that $y$ is indeed a path from $u$ to $v$.
    - Verify that the length of $y$ is at most $k$

- Is the verification algorithm for PATH polynomial bound?
- Is the size of $y$ polynomial in the size of $x$?
- Is the verification algorithm polynomial bound?
Example: A verification algorithm for TS

• Given a problem instance $x$ for TS and a certificate $y$
  – Check that $y$ is indeed a cycle that includes every vertex exactly once
  – Verify that the length of the cycle is at most $B$

• Is the size of $y$ polynomial in the size of $x$?
• Is the verification algorithm polynomial?
The class NP

• NP is the class of decision problems for which there is a polynomial bounded verification algorithm

• It can be shown that:
  – all decision problems in P, and
  – decision problems such as traveling salesman, knapsack, bin packing, are also in NP
The relation between P and NP

- $P \subseteq NP$

- If a problem is solvable in polynomial time, a polynomial time verification algorithm can easily be designed that *ignores the certificate* and answers “yes” for all inputs with the answer “yes”.

NP 26
The relation between $P$ and $NP$

- It is not known whether $P = NP$.

- Problems in $P$ can be *solved* “quickly”

- Problems in $NP$ can be *verified* “quickly”.

- It is easier to verify a solution than solving a problem.

- Some researchers believe that $P$ and $NP$ are not the same class.
Polynomial reductions

• **Motivation**: The definition of NP-completeness uses the notion of *polynomial reductions* of one problem $A$ to another problem $B$, written as

$$A \preceq B$$

• Let $tran$ be a function that converts any input $x$ for decision problem $A$ into input $tran(x)$ for decision problem $B$
Polynomial reductions


tran is a polynomial reduction from A to B if:
1. tran can be computed in polynomial bounded time
2. The answer to A for input x is yes if and only if the answer to B for input tran(x) is yes.

Algorithm for B

“yes” or “no”

Algorithm for A
Two simple problems

- **A**: Given n Boolean variables with values $x_1, \ldots, x_n$, does at least one variable have the value True?
- **B**: Given n integers $i_1, \ldots, i_n$ is $\max\{i_1, \ldots, i_n\} > 0$?

**Algorithm** for B:

Check the integers one after the other.
If one is positive stop and answer “yes”;
otherwise (if none is positive) stop and answer “no”.

Example:
n=4.
Given integers: -1, 0, 3, and 20.
Algorithm for B answers “yes”.
Given integers: -1, 0, 0, and 0.
Algorithm for B answers “no”.
Is there a transformation?

• Can we transform an instance of $A$ into an instance of $B$?
• Yes.

\[
\text{tran}(x) \\
\text{for } (j = 1; j <= n; j++) \\
\text{if } (x_j == \text{true}) \\
\quad i_j = 1 \\
\text{else } // x_j = \text{false} \\
\quad i_j = 0
\]

\[T(\text{false, false, true, false}) = 0, 0, 1, 0\]

• Is this transformation polynomial bounded? yes
Does it satisfy all the requirements?

- Can we show that when the answer for an instance $x_1,\ldots,x_n$ of $A$ is “yes” the answer for the transformed instance $\text{tran}(x_1,\ldots,x_n) = i_1,\ldots,i_n$ of $B$ is also “yes”?

- If the answer for the given instance $x_1,\ldots,x_n$ of $A$ is “yes”, there is some $x_j = \text{true}$.

- The transformation assigns $i_j = 1$.

- Therefore the answer for problem $B$ is also “yes” (the maximum is positive)
The other direction

- Can we also show that when the answer for problem B with input \( tran(x_1, \ldots, x_n) = i_1, \ldots, i_n \) is "yes", the answer for the instance \( x_1, \ldots, x_n \) of A is also "yes"?

- If the answer for problem B is "yes", it means that there is an \( i_j > 0 \) in the transformed instance.

- \( i_j \) is either 0 or 1 in the transformed instance. So \( i_j = 1 \), and therefore \( x_j = \text{true} \).

- So the answer for A is also "yes"
Polynomial reductions

**Theorem:**

If $A \preceq B$ and $B$ is in $P$, then $A$ is in $P$

If $A$ is not in $P$ then $B$ is also not in $P$
NP-complete problems

• A problem $A$ is **NP-complete** if
  1. It is in NP and
  2. For every other problem $A'$ in NP, $A' \propto A$

• A problem $A$ is **NP-hard** if
  For every other problem $A'$ in NP, $A' \preceq A$
Examples of NP-Complete problems

• Cook’s theorem
  – Satisfiability is NP-complete
• This was the first problem shown to be NP-complete

• Other problems
  – the decision version of 0-1 knapsack,
  – the decision version of traveling salesman
Coping with NP-Complete Problems

• To solve use approximations, heuristics, etc.

• Sometimes we need to solve only a restricted version of the problem.

• If the restricted problem is tractable, design an algorithm for the restricted problem
NP-complete problems: Theorem

If any NP-complete problem is in P, then $P = NP$.

If any NP-complete problem is not polynomial bound, then all NP-Complete problems are not polynomial bound.

The trivial decision problem that always answers “yes” in here
NP-completeness and Reducibility

• The existence of NP-complete problems leads us suspect that \( P \neq NP \).

• If HAMILTONIAN CYCLE could be solved in polynomial time, every problem in NP can be solved in polynomial time.

• If HAMILTONIAN CYCLE could not be solved in polynomial time, every NP-complete problem can not be solved in polynomial time.
The Satisfiability problem

- First, Conjunctive Normal Form (CNF) will be defined
- Then, the Satisfiability problem will be defined
- Finally, we will show a polynomial bounded verification algorithm for the problem
Conjunctive Normal Form (CNF)

- A *logical (Boolean) variable* is a variable that may be assigned the value *true* or *false* (p, q, r and s are Boolean variables)
- A *literal* is a logical variable or the negation of a logical variable (p and ¬q are literals)
- A *clause* is a disjunction of literals ( (p∨q∨s) and (¬q ∨ r) are clauses)
Conjunctive Normal Form (CNF)

• A logical (Boolean) expression is in Conjunctive Normal Form if it is a conjunction of clauses.
• The following expression is in conjunctive normal form:
  \[(p \lor q \lor s) \land (\neg q \lor r) \land (\neg p \lor r) \land (\neg r \lor s) \land (\neg p \lor \neg s \lor \neg q)\]
The Satisfiability problem

• Is there a truth assignment to the n variables of a logical expression in Conjunctive Normal Form which makes the value of the expression true?
• For the answer to be “yes”, all clauses must evaluate to true
• Otherwise the answer is “no”
The Satisfiability problem

• \( p=T, \ q=F, \ r=T \) and \( s=T \) is a truth assignment for:
\[
(p \lor q \lor s) \land (\neg q \lor r) \land (\neg p \lor r) \land (\neg r \lor s) \land (\neg p \lor \neg s \lor \neg q)
\]
• Note that if \( q=F \) then \( \neg q=T \)
• Each clause evaluates to true
A verification algorithm for Satisfiability

1. Check that the certificate s is a string of exactly n characters which are T or F.
2. \textbf{while} (there are unchecked clauses) {
   \quad select next clause
   \quad \textbf{if} (clause evaluates to false) \textbf{return} ("no") }
3. \textbf{return} ("yes")

- Is verification algorithm polynomial bound?
- Satisfiability is in NP since there exists a polynomial bound verification algorithm for it