Theory of NP-Completeness

- **NP-complete** is a class of problems with a certain property.
- Since it is a very theoretical area, it can also be complex.
- We won't get into the finer details of the theory, but will try to give an overall picture of what it is all about.
- Be aware that I will gloss over some important details/concepts, and make some simplifying assumptions.
- Before we talk about NP-complete problems, we first need to discuss several things, including
  - **decision problems**,
  - the class **P**, and
  - the class **NP**

Optimization/Decision Problems

- **Optimization Problems**
  - An **optimization problem** is one which asks "What is the optimal solution to problem X?"
  - **Examples:**
    * 0-1 knapsack/fractional knapsack
    * matrix chain multiplication
    * minimum spanning tree.
- **Decision Problems**
  - A **decision problem** is a problem that asks "Is there a solution to problem X satisfying property Y?"
  - **Examples:**
    * Does graph G have a minimum spanning tree of weight $\leq W$?
    * Can I multiply matrices $(A_1, \ldots, A_n)$ with $\leq M$ operations?

Optimization/Decision Example

- **Example**
  - **Problem:** 0-1 knapsack
  - **Instance:** A list of integer profits $P = (p_1, \ldots, p_n)$ and integer weights $W = (w_1, \ldots, w_n)$, and capacity M (an integer).
  - **Feasible Solution:** A vector $x = (x_1, x_2, \ldots, x_n)$, where $x_i \in \{0, 1\}$ for $1 \leq i \leq n$, and $\sum_{i=1}^{n} w_i x_i \leq M$.
  - **Optimal Solution:** A feasible solution which maximizes **profit**. That is, the quantity $P = \sum_{i=1}^{n} p_i x_i$.
  - **Question:** Is the optimal profit $\geq Q$?
- The optimization problem tries to find the optimal solution.
- The decision problem tries to answer the “yes/no” question.

Why Decision Problems

- When we discuss the various classes of problems (P,NP, NP-Complete), we restrict our attention to decision problems.
- This simplifies the study of complexity.
- It turns out that this restriction is not really much of a restriction.
- There are simple ways of using the solution to a decision problem to get a solution to a related optimization problem.
- **Example:** If we can solve the 0-1 knapsack decision problem, we can solve the 0-1 knapsack optimization problem by using a **binary search** type algorithm.
- Before we go further, we will see another important example of a decision problem.

Satisfiability (SAT)

- **Notation:** Let $\vee$ denote logical *or* of 2 boolean variables, and $\neg$ denote *negation* of a boolean variable. A **clause** is the logical *or* of boolean variables and/or their negations.
- **Problem:** Satisfiability (SAT)
  - **Instance:** A set of boolean variables $U = \{x_1, x_2, \ldots, x_m\}$, and a set of clauses, $C = \{c_1, c_2, \ldots c_s\}$.
  - **Question:** Is there a satisfying truth assignment? In other words, can boolean values be assigned to the variables x_i so that each of the clauses is true?

SAT Examples

- Let $U = \{u, v, w, x\}$, and

$$C = \{u \vee v \vee \neg x, \neg u \vee \neg v \vee w, u \vee \neg w \vee \neg x\}.$$

If we set $u = x = false$, and $v = w = true$, then it is easy to see that each of the clauses above is true, so the answer is "yes".

- Let $U = \{u, v, w\}$, and

$$C = \{u \vee v, \neg u \vee \neg v, u \vee w, \neg u \vee \neg w, v \vee w, \neg v \vee \neg w\}.$$

Then the answer is "no". But how do we know?

SAT Examples Continued

- Let $U = \{u, v, w\}$, and

$$C = \{u \vee v, \neg u \vee \neg v, u \vee w, \neg u \vee \neg w, v \vee w, \neg v \vee \neg w\}.$$

It seem that the only way to know that there is no solution is to try them all:

u	0	0	0	0	1	1	1	1
v	0	0	1	1	0	0	1	1
w	0	1	0	1	0	1	0	1
$u \vee v$	0	0	1	1	1	1	1	1
$\neg u \vee \neg v$	1	1	1	1	1	1	0	0
$u \vee w$	0	1	0	1	1	1	1	1
$\neg u \vee \neg w$	1	1	1	1	1	0	1	0
$v \vee w$	0	1	1	1	0	1	1	1
$\neg v \vee \neg w$	1	1	1	0	1	1	1	0

YES versus NO

- Notice that in the case of **SAT**:
  - answering “yes” is easy–we just find one assignment that works, but
  - answering “no” requires an exhaustive computation of all possible solutions.
- This is true for many decision problems.
- This is the basis of the theory of NP-completeness.
- We shall see more about this shortly.

The Class P

- A **polynomial-time algorithm** is one which runs in time $O(n^k)$, for some constant k, where n is the *size of the input*.
- A **deterministic algorithm** is (essentially) one which always returns the correct answer.
- The algorithms we have studied in this course are all deterministic, and most of them are polynomial-time.
- **Definition: P** denotes the collection of decision problems which have *deterministic polynomial-time algorithms*.
- **Examples:**
  - Is the largest element in the array A larger than m?
  - Does the graph G have a spanning tree with weight at most w?

Example of an problem in P

- **Example**
  - **Problem:** Fractional knapsack
  - **Instance:** A list of integer profits $P = (p_1, \ldots, p_n)$ and integer weights $W = (w_1, \ldots, w_n)$, and capacity M (an integer).
  - **Feasible Solution:** A vector $x = (x_1, x_2, \ldots, x_n)$, where $0 \leq x_i \leq 1$ for $1 \leq i \leq n$, and $\sum_{i=1}^{n} w_i x_i \leq M$.
  - **Optimal Solution:** A feasible solution which maximizes **profit**. That is, the quantity $P = \sum_{i=1}^{n} p_i x_i$.
  - **Question:** Is the optimal profit $\geq Q$?
- **Solution:** Solve optimization problem with the polynomial-time greedy algorithm, and test whether the optimal profit is larger than Q.

Certificates

- A **certificate** is a set of data representing a solution to an instance of a (decision) problem.
- If a particular instance of a problem is a “yes” instance, then there is some certificate (set of data) that meets the requirements.
- Thus, certificates can be used to prove that a particular instance of a problem is a “yes” instance.
- A certificate is **valid** if it provides proof that the particular instance of a decision problem is a “yes” instance.
- To prove that a particular instance of a decision problem is a “yes” instance, one can generate certificates until a valid one is found.
- This may or may not yield an efficient algorithm.

Certificate Example

- For **SAT**, a certificate is a truth assignment for the variables.
  - We saw that when $U = \{u, v, w, x\}$, and

$$C = \{u \vee v \vee \neg x, \neg u \vee \neg v \vee w, u \vee \neg w \vee \neg x\}.$$

    the certificate $\{u = x = true, v = w = false\}$ is a proof that $\{U, C\}$ is a "yes" instance of **SAT**.

  - We also saw that if $U = \{u, v, w\}$, and

$$C = \{u \vee v, \neg u \vee \neg v, u \vee w, \neg u \vee \neg w, v \vee w, \neg v \vee \neg w\},$$

    then none of the 8 certificates are valid. Since $\{U, C\}$ is a "no" instance of **SAT**, this is what we should expect.

Certificate Example

- For the 0-1 knapsack problem, a certificate is a list of integers $(x_1, x_2, \ldots, x_n)$ such that for $1 \leq i \leq n$, $x_i \in \{0, 1\}$, and

$$\sum_{i=1}^{n} w_i x_i \leq M.$$

  – A certificate is valid if it has profit $\geq Q$.

Non-Deterministic Algorithms

- To prove that an instance of a decision problem is a “yes” instance, we only need to find one valid certificate.
- A **non-deterministic algorithm N** is one which
  - Guesses a certificate (The nondeterministic stage)
  - Checks the validity of the certificate (The deterministic stage)
  - Returns “yes” if the certificate is valid and “no” otherwise
- Notice that if a non-deterministic algorithm produces the answer “false” it *does not necessarily* mean that the instance of the problem is a “no” instance.
- It only means that the certificate it guessed was not a valid certificate.

Examples

- **SAT**
  - Let $|U| = n$, and $|C| = m$.
  - If we are given a truth assignment for the n variables in U, we can check whether or not each of the m clauses is true or not in $O(n)$ time in the worst-case.
  - The total time to do the checking is $O(nm)$.
  - If each of the clauses is true, we output “true”.
  - Otherwise we output “false”.
- **Fractional knapsack**
  - Given an assignment $(x_1, x_2, \ldots, x_n)$, we can compute the profit P in $O(n)$ steps, and compare it to Q in constant time.
  - If $P \geq Q$, output “true”.
  - Otherwise we output “false”.

The Class NP

- **Definition: NP** denotes the collection of decision problems which have polynomial-time non-deterministic algorithms to solve them.
- In other words, a problem is in **NP** if every "yes" instance has some certificate that can be validated in polynomial time.
- Notice that a problem in **NP** does not necessarily have a deterministic polynomial algorithm to solve it.
- This is because the definition only assumes we can check the validity of certificates. It does not give us a method for finding valid certificates.
- For certain problems, there is no known method of finding valid certificates in polynomial-time.

Problems in NP

- From our earlier example, it is clear that **SAT** and the fractional knapsack problem are both **NP**.
- The famous **traveling salesman problem** is in **NP**. Can you argue this?
  - **Problem:** Traveling salesman problem
  - **Instance:** n cities, and a cost $c(i, j)$ to travel from city i to city j.
  - **Feasible Solution:** A route that takes the salesman through every city.
  - **Optimal Solution:** A feasible solution with minimal cost.
  - **Question:** Is there a route with cost $\leq C$?
- As we will prove (indirectly), most of the algorithms we have discussed are in **NP**.

More on NP

- As we just stated, for some problems in **NP**, there is no known way of producing a valid certificate in polynomial time.
- For instance, there is no known polynomial-time deterministic algorithm, or polynomial-time algorithm for finding a valid certificate for, the **traveling salesman problem**.
- In other words, it is unknown whether **NP**$\subseteq$**P** or not.
- It can be shown, however, that problems in **NP** can be solved in worst-case by exponential-time algorithms.
- As will be seen shortly, **P**$\subseteq$**NP**.
- It is important to remember that **NP** *does not* stand for *non-polynomial*.
- **NP** stands for **non-deterministic polynomial**, which is much different.

$\mathbf{P} \subseteq \mathbf{NP}$

- It is not very difficult to argue that **P**$\subseteq$**NP**
- Since a problem in **P** has a polynomial-time deterministic algorithm **A**, we can compute a valid certificate C in polynomial time.
- We use this fact to write a nondeterministic algorithm **N** as follows
  - Guess a certificate in the nondeterministic stage
  - Ignore the certificate in the deterministic stage
  - Instead, run **A**, which returns “yes” or “no,” and return that value.
- Since **A** is polynomial, so is the nondeterministic algorithm.
- If an instance is a “no” instance, **A** (and hence **N**) will return “no,” the correct answer.
- If an instance is a “yes” instance, **A** (and hence **N**) will return “yes,” the correct answer.

***The* Question: Is P=NP?**

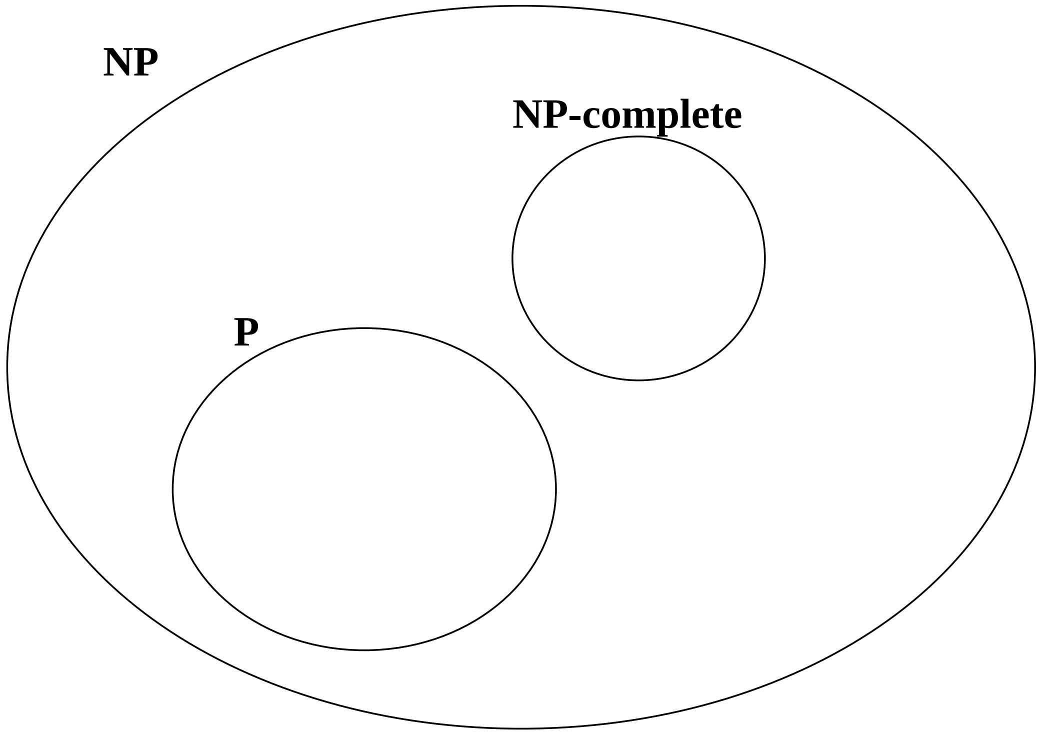
- The most important open question in theoretical computer science is the question "Is **P=NP**?"
- Why is this so important?
  - If it can be proved that indeed **P=NP**, then we will have polynomial-time algorithms to solve thousands of problems for which the best-known algorithms are exponential.
  - If it can be shown that **P**$\neq$ **NP**, then we will know that no polynomial-time algorithm exists for certain problems. That is, problems in **NP-P**.
- The question of which problems belong to the set **NP-P**, if any, has been studied extensively for several decades.
- This is where the class of **NP-complete** problems comes into play.

NP-complete

- There is a lot more to the theory of NP-Completeness than we can cover here.
- It is important, however, to have a basic understanding of why **NP-complete** problems are so important.
- A problem D is said to be **NP-complete** if
  - $D \in$ **NP**, and
  - For all $D' \in \mathbf{NP}$, there exists a polynomial time algorithm that maps "yes" ("no") instances of D' to "yes" ("no") instances of D.
- In other words, a solution to an **NP-complete** problem yields a solution to *any other* problem in **NP**. Why?
- Thus, if one can find a polynomial-time algorithm to solve an **NP-complete** problem, then it can be used to solve *all* problems in **NP**, and **P=NP**.

NP-complete Summary

- Although it is not known whether or not **P=NP**, most people think **P** $\neq$ **NP**.
- If $\mathbf{P} \neq \mathbf{NP}$, the **NP-complete** problems are the most likely problems to be in **NP-P**.
- Thus, the most likely scenario is as follows:



- If you can prove that **P=NP** or that **P** $\neq$ **NP**, you will be famous.

A Final Note on NP-completeness

- There is one more important use of the theory of **NP-completeness**.
- If you are asked by your boss to write an efficient algorithm for some problem, there are several possibilities:
  - You will find a polynomial-time algorithm to solve the problem, and everyone will be happy.
  - You will be unable to find a polynomial-time algorithm, and your boss will fire you, and you will both be unhappy.
  - You will be unable to find a polynomial-time algorithm, but you will be able to show that the problem is **NP-complete**. Then your boss will be unhappy, but he can't fire you, because you have shown that nobody else can come up with a good solution either, so you will be happy.