Lecture 15: P and NP

Lorenzo De Stefani
Fall 2020
Outline

• The class P
• Towards the definition of NP
• Verifiability
• The class NP
• Examples of NP problems
• Co-NP

From Sipser Chapter 7.2
Recap

- Relation of single-tape TM time and Multi-tape
  
  \[ \text{TIME}(t(n)) \subseteq \text{MTIME}(t(n)) \]
  
  \[ \text{MTIME}(t(n)) \subseteq \text{TIME}(2^{O(t(n))}) \]

- Relation between deterministic and non-deterministic TM running time
  
  \[ \text{TIME}(t(n)) \subseteq \text{NTIME}(t(n)) \]
  
  \[ \text{NTIME}(t(n)) \subseteq \text{TIME}(2^{O(t(n))}) \]
Consequences

These results show an important difference:

• The difference between a single and multi-tape TM is at most a square, or polynomial, difference

• Moving to a nondeterministic TM yields an exponential difference
• With respect to time complexity, polynomial differences are considered small and exponential ones large
  – Exponential functions grow incredibly fast, much faster than polynomial function
  – Differences between degrees of polynomial are still extremely important when analyzing algorithms:
    • Different polynomials do grow much faster than!
    • $O(n \log n)$ sorting much better than $O(n^2)$.
    • $O(n)$ and $O(n^3)$ not even comparable in practice!
• We still want to assuming polynomial time equivalence towards highlighting the difference with exponential time algorithms
Background

- Exponential time algorithms usually involve exhaustively searching a space of possible solutions using \textit{brute force (exhaustive) search}.
- Polynomial time algorithms are more efficient in exploring the possible solutions space.
- All reasonable computational models are \textit{polynomial-time equivalent}.
  - If we view all polynomial complexity algorithms as equivalent then the specific computational model does not matter.
P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine.

\[ P = \bigcup_{k=1}^{\infty} \text{TIME}(n^k) \]
The Class P

P is of great importance in the theory of computation

- P is invariant for all models of computation that are polynomial equivalent to the deterministic single-tape Turing machine (Multitape, RAM)

- P roughly corresponds to the class of problems that are realistically solvable on a computer
  - Still some exponential algorithms can be solved on realistic machines although they have extremely poor scaling
P and decidable problems

If something is decidable, then there is a method to compute/solve it

– It can be in P, in which case there is an algorithm to solve it, whiteout exploring the space of solutions brute-force

– It may not be in P in which case it can be solved by brute force

– If it is not in P then it can only be solved via brute force searching

– Languages corresponding to problems not in P are interesting as they are not efficiently decidable

– Note: NP does not mean “not in P” (as we shall soon see). In fact every problem in P is in NP (but not vice versa).
Problems in P

• When we present a polynomial time algorithm, we give a high level description without reference to a particular computational model
• We describe the algorithms in stages
• When we analyze an algorithm to show that it belongs to P, we need to:
  – Provide a polynomial upper bound, usually using big-O notation, on the number of stages in terms of an input of length $n$
  – We need to examine each stage to ensure that it can be implemented in polynomial time on a reasonable deterministic model
  – Note that the composition of a polynomial with a polynomial is a polynomial, so we get a polynomial overall
    • We choose the stages to make it easy to determine the complexity associated with each stage
The “PATH” problem is to determine if in a directed graph $G$ whether there is a path from node $s$ to node $t$

- $\text{PATH} = \{<G,S,t>| G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$
- Prove $\text{PATH} \in P$
- Brute force method is required exploration of all the exponential possible directed paths
  - Total number of possible paths is around $m^m$
    - Think about strings of length $m$. Each symbol represents a node. Actually less than $m^m$ since no node can repeat.
    - Some of these paths should actually not be considered as paths with cycles are of no interest. Assuming $m$ is the number of nodes in $G$, the path cannot be longer than $m$. 
A breadth-first algorithm runs in polynomial time:

- \( M = \) On input \(<G, s, t>\) where \( G \) is a directed graph with nodes \( s \) and \( t \) (and there are \( m \) nodes):
  1. Place a mark on node \( s \)
  2. Repeat until no additional nodes are marked
     3. Scan all edges of \( G \). If an edge \((a,b)\) is found from a marked node \( a \) to an unmarked node \( b \), mark node \( b \)
  4. If \( t \) is marked, accept. Otherwise, reject.

- Analysis:
  - Stages 1 and 4 executed exactly 1 time each. Stage 3 runs at most \( m \) times since each time a node is marked. There are at most \( m+2 \) stages, which is polynomial in the size of \( G \).
  - Stages 1 and 4 are easily implemented in polynomial time on any reasonable model. Stage 3 involves a scan of the input and a test of whether certain nodes are marked, which can easily be accomplished in polynomial time. Proof complete and \( \text{PATH} \in \mathbb{P} \).
The Class NP

• We have seen that in some cases we can do better than brute force and come up with polynomial-time algorithms

• In some cases polynomial time algorithms are not known
  – Do they exist but we have not yet found the polytime solution?
  – Or does one not exist?
  – Core of the P vs. NP question!

• The complexities of many problems are linked
  – If you solve one in polynomial time then many others are also solved
Towards the definition of NP 1/2

$NP_1$ is the class of languages that are decidable in polynomial time on a non-deterministic Turing machine.

$NP_1 = \bigcup_{k=1}^{\infty} NTIME(n^k)$

• NP stands for non-deterministic polynomial!
• Recall that $TIME(t(n)) \subseteq NTIME(t(n))$
• Clearly $P \subseteq NP_1$
An Hamiltonian path in a directed graph $G$ is a directed path that goes through every node exactly once

- Consider the problem of whether two specific nodes in $G$ are connected with a Hamiltonian path
- $\text{HAMPATH} = \{<G,s,t>|G\text{ is a directed graph with a Hamiltonian path from }s\text{ to }t\}$

- No polynomial-time algorithms are known for this problem
Polynomial Verifiability

• Even though we do not know of a fast (polytime) way to determine if a Hamiltonian path exists, if we discover such a path (e.g., with exponential brute-force method), then we can verify it easily (in polytime)
  • We can verify the solution just “presenting it”
  • This can be done in polynomial time
    – At worst in time $O(n^3)$ since paths at most $n$ long and graph has at most $n^2$ edges.

• Verifiability is often much easier than coming up with a solution
• Such property is referred as polynomial verifiability
Definition of Verifier

• Let $A$ be a language. A **verifier $V$** for $A$ is a TM such that it recognizes the language

$$ A = \{w | V \text{ accepts} < w, c > \text{ for some string } c \} $$

• A verifier for a language $A$ is an algorithm $V$, which uses **additional information**, represented by the symbol $c$, to verify that a string $w$ is a member of $A$

• This information is called a **certificate (or proof) of membership** in $A$

• If a verifier runs in **polynomial time** with respect to $|w|$, it is called a **polynomial verifier**

• A language $A$ is polytime verifiable if it has a polytime verifier
Another Polynomial Verifiable Problem

• A natural number is a **composite** if it is the product of two integers greater than 1
  – A composite number is **not a prime number**
  – COMPOSITES = \{x | x = pq, for integers p,q > 1\}

• Is it hard to check that a number is a composite if we are given the solution?
  – No, we must multiply the numbers. This is in **polynomial time verifiable**

• Interesting mathematical trivia:
  – Is there a brute-force method for checking primality of m?
    • Yes: try all integers from 2 to square root of n.
    • This runs in exponential time (w.r.t. the representation of the integer)
  – Is there a polytime algorithm for checking primality?
    • For a long time we had a probabilistic polynomial time algorithms
    • The existence of a deterministic polytime algorithm was proven in 2002 (AKS primality test, 2006 Gödel prize)
• For HAMPATH, what is the certificate for a string $<G,s,t> \in \text{HAMPATH}$?
  – Answer: it is the Hamiltonian path

• For the COMPOSITES problem, what is the certificate?
  – Answer: it is one of the two divisors

• In both cases we can check that the input is in the language in polynomial time given the certificate.
Some Problems are Not Polynomial Time Verifiable

• Consider $\text{HAMPATH}^c$, the complement of $\text{HAMPATH}$
• Even if we know there is not an Hamiltonian path between two nodes, we do not know how to verify it without going through the same number of exponential steps to determine whether one exists
• Thus, this problem is not polynomial time verifiable
Towards the definition of NP 2/2

\( \text{NP}_2 \text{ is the class of languages that have a polynomial-time verifier} \)

\[ \text{NP}_2 = \{ L \mid L \text{ has a polynomial time verifier} \} \]

- What is the relation between \( \text{NP}_1 \) and \( \text{NP}_2 \)?

NP₁ = NP₂

Proof: NP₁ ⊆ NP₂

– That is, if there is a non-deterministic algorithm which solves a problem in polynomial-time is possible to verify a solution in polynomial time

– For any \( L \in NP₁ \), by definition there exist a Polynomial time NTM \( N \) which decides \( L \)

– Construct a verifier \( V \) for \( L \):
  
  • On input \( <w,c> \)
  
  • Simulate \( N \) on \( w \) using \( c \) to pick the transitions at each step
  
  • If \( N \) accepts (rejects) so does \( V \)
\[ \text{NP}_1 = \text{NP}_2 \]

Proof: \( \text{NP}_2 \subseteq \text{NP}_1 \)

- For any \( L \in \text{NP}_2 \), by definition there exist a Polynomial time verifier \( V \) for \( L \)

- Construct a Non-deterministic TM decider \( N \) for \( L \):
  - On input \(<w>\>
  - Non deterministically pick a certificate \( c \) of length polynomial in \( |w| \)
  - Simulate \( V \) on \(<w,c>\>
  - If \( V \) accepts (rejects) so does \( R \)
Definition of NP

- \( NP \triangleq NP_1 = NP_2 \)

- NP is class of languages that have polytime verifiers
  - NP comes from Nondeterministic Polynomial time
  - Alternate formulation: languages for which there exists polynomial time nondeterministic TM decider

- The class NP is important because it contains many problems of practical importance
  - HAMPATH and COMPOSITES \( \in \) NP
  - COMPOSITES \( \in \) P, but the proof is difficult

- Clearly if something is in P it is in NP. Why?
  - Because if you can find the solution in polytime then certainly can verify it in polytime (just run same algorithm)
  - So, \( P \subseteq NP \)
The P Versus NP Question

• So $P = NP$ or $P \neq NP$?

....maybe the most important open problem in CS!

• $P =$ the class of languages for which membership can be decided quickly
  – you must be able to determine whether a certificate exists or not in polynomial time

• $NP =$ the class of languages for which membership can be verified quickly
  – you are given the certificate and must check it (verify its correctness) in polynomial time
Example of Problem in NP: CLIQUE

A clique in an undirected graph is a subgraph, where each pair of two nodes are connected by an edge.

– A k-clique is a clique that contains k-nodes
– The graph below has a 5 clique
The clique problem is to determine whether a graph contains a clique of a specified size

- $CLIQUE: \{< G, k > \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

- Note that $k$ is a parameter. Thus the problem of deciding whether there is a 3-clique or a 10-clique is not the CLIQUE problem

- This is important because we will see that CLIQUE is NP-complete
Prove that CLIQUE ∈ NP

• We need to show that the language can be verified in polynomial time
• We just need to prove that the clique is a certificate
• We build a verifier V for CLIQUE:
  • V = On input <<G,k>, c>:
    – Test whether c is a set of k nodes in G
    – Test whether G contains all edges connected nodes in c
      » This requires you to check at most $k^2$ edges
    – If both pass, accept; else reject
  – If you prefer to think of NTM method:
    • N = On input <G, k> where G is a graph
      – Nondeterministically select a subset c of k nodes of G
      – Test whether G contains all edges connected nodes in c
      – If yes, accept, otherwise reject
  – Each step runs in polynomial time (in second case in nondeterministic polynomial time)
• HAMPATH and CLIQUE $\in$ NP
• We do not know if they belong to P
• Verifiability seems much easier than decidability, so we would probably expect to have problem in NP but not P
• No one has ever found a single language that has proved to be in NP but not P
• I believe $P \neq NP$
  – People have tried to prove that many problems belong to P (e.g., Traveling Salesman Problem) but have failed
  – Best methods for solving some languages in NP deterministically use exponential time
  – $NP \subseteq \text{EXPTIME} = \bigcup_k \text{TIME}(2^{n^k})$
    • We do not know if NP is contained in a smaller deterministic time complexity class
The SUBSET-SUM problem:

- Given a collection of numbers $S = \{x_1, ..., x_n\}$ and a target number $t$
- Determine whether $S$ contains a subset that sums to $t$
- SUBSET-SUM = $\{<S, t>| S = \{x_1, ..., x_n\}$ and for some $\{y_1, ..., y_l\} \subseteq \{x_1, ..., x_n\}$, we have $\sum y_i = t\}$
  - Note that these sets are actually multi-sets and can have repeats
- Does $\langle\{4,11,16,21,27\}, 25\rangle \in$ SUBSET_SUM?
  - Yes since $21+4 = 25$
Prove that \( \text{SUBSET-SUM} \in \text{NP} \)

• Show that the subset is a certificate
• Proof:
  – The following is a verifier for \( \text{SUBSET-SUM} \)
    • \( V = \) On input \( <<S,t>,c>:\)
      – Test whether \( c \) is a collection of numbers that sum to \( t \)
      – Test whether \( S \) contains all the numbers in \( c \)
      – If both pass, accept; otherwise, reject.
    • Clearly in polynomial time with respect to the size of the input (numbers in \( S \))
Consider the languages which are the complements of CLIQUE and SUBSET-SUM

- E.g., $\text{CLIQUE}^c$: $\{<G,k>|G$ is an undirected graph with no $k$-clique$\}$
- It is very challenging to show that they are in NP
- Verifying that something is not present seems to be more difficult than verifying that it is present
- The class $\text{coNP}$ contains the languages that are complements of languages in NP
- We do not know if $\text{coNP}$ is different from NP