COMP2210: Theory of Computing

Lecture 22: The Class NP

Corina Cîrstea
Objectives for Today

- non-deterministic Turing machines and their expressiveness
- time complexity of (non-deterministic) Turing machines
- the class NP
- examples of NP problems
Recap: On the HAMPATH problem

Some observations:

- brute-force algorithm for PATH can be adapted to HAMPATH
- we don’t know if a polynomial time algorithm for HAMPATH exists...
- ...but if we are given a candidate for a Hamiltonian path, we can verify whether it is a valid one in polynomial time!
Non-deterministic Turing machines

Definition

A non-deterministic Turing machine is a tuple

\[ M = (Q, \Sigma, \Gamma, \sqcup, \sqsubset, \delta, s, t, r) \]

where

- \( Q \) is a finite set of states,
- \( \Sigma \) is the input alphabet,
- \( \Gamma \) is the tape alphabet,
- \( \sqcup \) is the blank symbol,
- \( \sqsubset \in \Gamma - \Sigma \) is the left endmarker,
- \( \sqsubset \in \Gamma - \Sigma \) is the right endmarker,
- \( \delta : (Q - \{t, r\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}) \) is the transition function,
- \( s \in Q \) is the start state,
- \( t \in Q \) is the accept state,
- \( r \in Q \) is the reject state, \( r \neq t \).
Non-deterministic Turing machines

- A non-deterministic machine can be thought of as making all possible transitions from a given configuration in parallel;
- The computation of a non-deterministic Turing machine is a tree, whose branches are all traversed in parallel!
  - The nodes of the tree are configurations.

Deterministic machine:  Non-deterministic machine:

• Note: this is a *non-physical* model of computation!
Non-deterministic Turing machines

Definition

A non-deterministic Turing machine

- accepts an input if some branch of the computation leads to the accept state,
- rejects an input if all branches of the computation lead to the reject state.

A non-deterministic Turing machine is called a decider if all branches halt on all inputs.

That is, the machine accepts an input if there exists a sequence of "choices" which ends in the accepting state.

A decider will either accept an input or reject it.
Expressiveness of Non-deterministic Turing Machines

Theorem

For any non-deterministic TM $N$, there exists a deterministic TM $D$ accepting the same language.

Proof idea: A non-deterministic TM $N$ can be simulated by a deterministic (multi-tape) TM $D$ which tries all branches of $N$’s computation (breadth-first):

- one tape stores the input,
- one tape used to simulate a branch up to a given depth,
- one tape used to remember the branch being simulated.

If $D$ ever finds the accept state on one of the branches, it accepts. Otherwise $D$ will loop.

(See pp. 152-153 of Sipser for details.)

Corollary

A language is recursively enumerable if and only if some non-deterministic TM accepts it.
We can modify the proof of the previous theorem so that if \( N \) halts on all branches then \( D \) halts:

- remember the branches that halt (e.g. on a fourth tape) and do not explore them again;
- if no branch left to explore, reject.

**Theorem**

*Any non-deterministic TM has an equivalent deterministic TM.*

**Corollary**

*A language is recursive if and only if some non-deterministic TM decides it.*
**Time complexity of Turing Machines**

For Turing machine problems, we take

- the size of a problem instance to be the number of symbols on the input tape,

- the algorithm cost (measure of time taken) to be the number of Turing machine steps executed.

**Definition**

The running time, or time complexity of a decider is the function $f : \mathbb{N} \rightarrow \mathbb{N}$ where $f(n)$ is the maximum number of steps the machine uses on any branch of its computation, on any input of length $n$. 
Comparing Computational Models

Theorem

Let \( t(n) \) be a function, where \( t(n) \geq n \).

1. Every \( t(n) \) time multi-tape (deterministic) Turing machine has an equivalent \( O(t^2(n)) \) time single-tape (deterministic) Turing machine.

2. Every \( t(n) \) time non-deterministic Turing machine has an equivalent \( 2^{O(t(n))} \) time deterministic Turing machine.

N.B. "\( f(n) \) is \( 2^{O(t(n))} \)" means \( f(n) \leq 2^{c \cdot t(n)} \) for some constant \( c > 0 \) and all \( n \geq N \), with \( N \) fixed.
Comparing Computational Models

Proof sketch:

1. If a multi-tape machine $M$ runs in $t(n)$ time, it can only use $t(n)$ tape cells on each tape.
   - The simulation of each of $M$'s steps by a single-tape machine uses $O(t(n))$ steps – need to scan the tape to find the tape heads before simulating each step.

2. Each branch of a $t(n)$ time non-deterministic machine $N$ has length at most $t(n)$.
   - Each node in the computation tree of $N$ can have up to $b$ children.
   - Hence there are at most $b^{t(n)}$ leaves, and $O(b^{t(n)})$ nodes.
   - Then the running time of the multi-tape deterministic machine simulating $N$ is $O(t(n) \times b^{t(n)})$, hence $2^{O(t(n))}$.
   - Simulating this with a single-tape deterministic machine requires $(2^{O(t(n))})^2 = 2^{O(2 \times t(n))} = 2^{O(t(n))}$ steps.

(See pages 258–260 of Sipser for details.)
The Class NP

Definition

The class NP is the set of all decision problems that can be solved by a non-deterministic Turing machine in polynomial time.

IMPORTANT: NP = Non-deterministic Polynomial
Examples of NP Problems

- Anything in P is in NP. If a problem can be solved in polynomial time by a deterministic machine, then also by a non-deterministic one.
NP Problems: SAT

- **SAT**: Boolean satisfiability

Given a boolean expression, e.g.

$$(x \lor y \lor \neg z) \land (w \lor \neg z) \land (\neg w \lor x)$$

does there exist an assignment of 0s and 1s to the variables $(w, x, y, z)$ such that the formula is satisfied?
NP Problems: SAT

Instances of SAT:

- \((x \lor y) \land (\neg x \lor \neg y) \land (x \lor z)\)
  - “yes”: e.g. \(x = 1, y = 0, z = 0\)

- \(x \land y \land (\neg x \lor z) \land (\neg y \lor \neg z)\)
  - “no”; always \text{FALSE}

Are these satisfiable?
NP Problems: SAT

- **SAT** is in NP: we can non-deterministically generate all possible assignments to the variables, then substitute each of them in the formula to see if it is satisfied.

- This can be done in non-deterministic polynomial (linear) time.

- We don’t know if **SAT** is in P! We have no polynomial time algorithm for it, but no one has been able to prove there isn’t one.
HAMPATH: Given a directed graph, does there exist a directed path connecting two given nodes that goes through each node exactly once?
NP Problems: HAMPATH

HAMPATH is in NP.

We can non-deterministically generate all sequences of nodes of length equal to the size of the graph $n$, and then check for any such sequence that:

- no repetitions are found in the sequence,
- the start and end of the sequence coincide with the required nodes,
- the sequence defines a directed path through the graph.

We don’t know if HAMPATH is in P.
• **TSP(D):** Travelling Salesperson Problem (Decision version)

Given a number $D$ and $n$ cities $c_1, \ldots, c_n$ with lists of distances between them: $d(c_i, c_j) = d_{i,j}$, is there a route leading round all the cities and back to the start, with total distance at most $D$?
NP Problems: TSP(D)

Is there a route with distance $\leq 43$?

TSP(D) is in NP: the non-deterministic machine can just “guess” an order in which to visit the cities, and then add up the distances to see if they are at most $D$.

We don’t know if TSP(D) is in P.
NP Problems: 3COL

- **3COL**: 3-colourability

Given a graph $G$, is it possible to colour the vertices with at most 3 colours such that no two adjacent vertices have the same colour?
NP Problems: 3COL
NP Problems: 3COL

“yes”

“no”
NP Problems: 3COL

- **3COL** is in NP: we can “guess” the colour of each vertex and check that the resulting colouring is valid.

- We don’t know if **3COL** is in P.
The class NP - An alternative Definition

These examples show another way of looking at NP:

NP is the set of all decision problems for which a solution can be checked (by a deterministic TM) in polynomial time.

(a solution := a boolean assignment, a tour of cities, a possible colouring)

**Note:** Sipser takes (a formalisation of) the above observation as the definition of the class NP (see section 7.3)

- input to the deterministic machine (the verifier) consists of a problem instance and a ”certificate”, or ”proof” that the given instance is a yes instance

- size of certificate can be no more than polynomial in the size of the problem instance, if the certificate can be checked in polynomial time!
A Problem not in NP

Presburger arithmetic:

Consider the simple theory of addition on natural numbers, where statements are closed formulas of a first order predicate calculus that uses just 0, 1, + and = (in addition to boolean operators and quantifiers).

Some examples of formulas:

\[ \forall x \exists y (x + y = x + 1) \]

\[ \exists x \forall y \exists z (x + z = x + y + z) \]

\[ \forall x \forall y \exists z (\neg (x = y) \Rightarrow (x + z = y)) \]

The problem: Given a formula of the above form, is it true? This problem is known to be decidable [Presburger, 1930]. It is also known not to be in NP [Fischer and Rabin, 1974].
There are lots of NP problems like SAT, TSP(D) and 3COL for which the best known algorithm is exponential.

But no one has been able to prove that the only algorithms for these problems are exponential!

In fact, we don’t know whether P and NP are equal or not! Do there exist problems in NP which are not in P? We suspect “yes”, but can’t prove it!