This lecture covers (part of) Chapter 10 of HMU: Intractable Problems

- Complexity Class \( \mathcal{P} \)
- Complexity Class \( \mathcal{NP} \)

Additional Reading: Chapter 10 of HMU.
What is Complexity?
So far, we focussed on decidability.

If a problem $P$ is decidable, how easy or hard is it?

We need a notion for hardness of an algorithm.

Computational complexity is the framework to classify (decision) problems.
Asymptotic Notation

\[ f(n) = O(g(n)) \]
| \( f(n) \) | ≤ \( c|g(n)| \) eventually

\[ f(n) = \Theta(g(n)) \]
| \( f(n) \) | ≤ \( \beta|g(n)| \) eventually
| \( f(n) \) | ≥ \( \alpha|g(n)| \) eventually

\[ f(n) = o(g(n)) \]
\[ \frac{f(n)}{g(n)} \xrightarrow{n\to\infty} 0 \]
What is Complexity?

Time Complexity of a Turing Machine

> A Turing Machine $M$ is said to have a **time complexity** of $T(n)$ (or equivalently, a **running time** of $T(n)$) if it halts after $T(n)$ moves for any input of $n$ input symbols.

> Given a TM, one simply has to count/estimate the number of moves to identify its time complexity.

**Time complexity of string reversal**

The TM takes $O(n^2)$ time steps to complete string reversal.
Complexity Class $\mathcal{P}$
The complexity class $\mathcal{P}$

- Let for each $i \in \mathbb{N}$, $\text{TIME}(n^i)$ denote the set of all problems that have a time complexity of $O(n^i)$.

- The complexity class $\mathcal{P}$ consists of all problems that are decidable in polynomial time by a deterministic Turing machine.

$$\mathcal{P} := \bigcup_{i \geq 0} \text{TIME}(n^i)$$

- Some problems in $\mathcal{P}$:
  - Path problem: Given a directed graph $G$, is there a path from $s$ to $t$?
  - Test relative primeness: Given $n, m \in \mathbb{N}$, test if $\gcd(n, m) = 1$. (Euclid’s algorithm)
  - Any decision problem equivalent to “Is $w \in L(G)$ for a given CFG $G$” (See Ch. 8 notes for algorithm)
  - Minimum-weight Spanning Tree (MWST) Problem
A spanning tree of an undirected graph $G(V, E)$ is any subgraph $G'(V, E')$ with the least number of edges that preserves connectivity:

Two nodes are connected in $G$ $\iff$ they are connected in $G'$. 

$G(V, E)$

$G'(V, E')$
Minimum Weight Spanning Tree (MWST)

- Given (undirected) graph \( G(V, E) \) and cost/weight function \( w : E \rightarrow \mathbb{N} \), identify a spanning tree such that the sum total of the weight is minimized.

\[ G(V, E) \]

1. Start with \( V' = V \) and \( E' = \emptyset \).
   Repeat 2 until ends of every edges in \( E'^c \) is in the same component.
2. Pick an(y) least-weight edge \( e \in E'^c \) with ends in different components.
   Update: \( E \leftarrow E \cup \{e\} \) and component of each node.

Kruskal’s Algorithm
Complexity of Kruskal’s Algorithm (Crude Estimate)

Given $G$ with $n$ nodes and $m$ edges:

- Each round of edge selection requires:
  - Identify an edge of least weight: $O(m)$ time steps.
  - Update the component of all nodes: $O(n)$ steps.
- There are at most $O(m)$ rounds.
- Kruskal’s algorithm requires $O(m(n + m))$ time steps.

Is this the complexity of the algorithm?

Not really.

We need to represent the graph and weights in a format a TM can interpret.

Each problem information/detail must be encoded in a **self-delineating** manner.
(To represent a list of numbers as input, the TM needs to know where the representation of one number ends and where next begins)

One possible coding scheme: Number nodes by integers, and represent edge $(i, j)$ and cost $w_{i,j} \in \mathbb{N}$ by $0^i10^j10^{w_{i,j}}$ and concatenate them with 11.

Input length: $\Theta(\max\{|V|, |E|, W\})$, where $W =$ maximum edge weight.

Kruskal’s algorithm is polynomial in the input length.
Complexity Class $\mathcal{NP}$
The complexity class $NP$

- Let for each $i \in \mathbb{N}$, $NTIME(n^i)$ denote the set of all problems (or TMs or languages) that are decidable in $n^i$ moves by a non-deterministic Turing machine.
  - All branches halt after $n^i$ moves.
  - Input is accepted if there is a branch that accepts.

- The complexity class $NP$ consists of all problems that are decidable in polynomial time by a non-deterministic Turing machine.

$$NP := \bigcup_{i \geq 0} NTIME(n^i)$$

- Some problems in $NP$:
  - Boolean Satisfiability Problem (SAT)
  - Dominating set problem
  - Knapsack problem
  - Travelling Salesman Problem
Travelling Salesman Problem

- Given an undirected graph $G(V, E)$, and edge weights $w : E \rightarrow \mathbb{N}$, is there a Hamiltonian circuit of weight $W$?
  - A Hamiltonian circuit is a cycle that visits each node exactly once.
  - We want the ‘salesman’ to visit all ‘cities’ without costing more than $W$ units.
- In general, a graph on $n$ vertices can contain $O(n!)$ Hamiltonian circuits, which is super-exponential!

TSP is in $\mathcal{NP}$

- We can design a multi-tape NTM $N$ that writes $n\lceil \log_2 n \rceil \in O(n^2)$ bits (nondeterministically) on one of its track.
- Each branch then has a string of $n\lceil \log_2 n \rceil$ bits that can be viewed as a list of $n$ vertices indexed by $\{0, \ldots, n - 1\}$.
- Every tuple of $n$ vertices is considered non-deterministically, since $n! < n^n = 2^{n\log_2 n}$.
- The cost of each Hamiltonian circuit can be non-deterministically computed in $O(n^2)$ time steps.
- NTM $N$ can then return the least cost of a Hamiltonian circuit in $O(n^2)$ time steps.
- A single-tape NTM can do the same work in $O(n^4)$ time steps.
$\mathcal{NP}$-Complete Problems
A Closer Look at Reductions

> **Reduction**: We say a (decision) problem $P$ reduces to (decision problem) $Q$, if there is a TM that translates every instance of $P$ to an instance of $Q$ such that a yes/no instance of $P$ maps to a yes/no instance of $Q$.

> When we look at the computational complexity of problems, we have to consider the time cost of a reduction; otherwise, we cannot meaningfully use reductions to translate complexity results from one problem to another.