Objectives

- Review the main ideas of computational “Tractability”, including:
  - Equivalence of search & decision problems
  - Formulation of optimization problems as search/decision problems
  - P, NP
  - Polynomial-time Turing Reduction
  - NP-Complete
  - P = NP?
- What to do when you are faced with a hard problem

Decision Problems

- A decision problem is a question in some formal system with a yes-or-no answer, depending on the values of some input parameters.
- Decision algorithm C, a verifier!
  - Takes two inputs:
    - instance $I$
    - proposed solution $S$
  - We say $S$ is a solution to $I$ iff $C(I,S) = \text{yes}$
- Examples:
  - Given a variable assignment and an expression in 3-CNF, is the expression satisfied by the assignment?
  - Given a Hamiltonian circuit and a budget, does the cost of the Hamiltonian circuit come under budget?

Search Problems

- A search problem requires finding a solution, possibly subject to some constraint(s).
- Given an instance $I$
  - Input data specifying the problem instance at hand
  - Goal: to find a solution $S$
    - An object that meets a particular specification
    - If no such solution exists, then say so
  - $S$ must be able to be quickly checked for correctness
    - $S$ must be concise (length bounded by polynomial in $|I|$)
    - i.e., there exists a polynomial-time algorithm with arguments $(I,S)$ that decides whether or not $S$ is a solution of $I$
  - Any search problem can be represented by its corresponding decision problem.

Optimization Problems

- An optimization problem requires finding the best solution from all feasible solutions.
- More formally, an optimization problem $A$ is a 4-tuple $(I, f, m, g)$, where
  - $I$ is a set of instances
  - For an instance $x \in I$, $m(x)$ is the set of feasible solutions to $x$
  - For an instance $x$ and a feasible solution $y$, $m(x,y)$ denotes the measure of $y$, which is usually a positive real number
  - $g$ is the goal function: either minimization or maximization
- The goal is then to find for a given instance $x$ an optimal solution $y$:
  - Optimal measure $A(x) = g(y^*)$
  - Optimal solution $y^* = \arg\max_{y \in m(x)} g(y)$
  - Note: $m(x,y) = A(x)$

Combinatorial Optimization Problems

- A special case of optimization problems.
- Where the set of feasible solutions is discrete
  - or can be reduced to a discrete one
**Traveling Salesman Problem**

- **Rudrata or Hamiltonian Cycle**
  - Cycle in the graph that passes through each vertex exactly once

- **Least Cost or “shortest”**


---

**TSP**

- How to cast the TSP as an optimization problem?
  - **TSP-OPT:**
    - Input: A matrix of distances between cities
    - Output: The shortest tour passing through all cities
  - As a search problem? (Hint: use a budget)
    - **TSP-SEARCH:**
      - Input: A matrix of distances AND a budget b
      - Output: A tour passing through all cities and having length $\leq b$, if such a tour exists
  - As a decision problem? (Hint: use a budget)
    - **TSP-CHECK:**
      - Input: A matrix of distances, a budget b, a tour
      - Output: YES/NO, does the tour pass through all cities and have length $\leq b$?

- Relationships?

---

**P**

- P is the set of all search problems that can be solved in polynomial time using a deterministic Turing Machine (TM)
  - Equivalently:
    - All languages decidable in polynomial time
  - Efficiency:
    - All polynomial time algorithms are efficient.
    - $O(n), O(n^2), O(n^{1000})$, etc. are all efficient!

---

**NP**

- NP is the set of all search problems that can be solved in polynomial time on a non-deterministic TM
  - Equivalently:
    - All search problems with polynomial time decision algorithms
    - All languages decidable in non-deterministic polynomial time

---

**NP (continued)**

- One method for solving a problem in NP by deterministic means:
  - Explore all the computational paths of the non-deterministic TM
- What does that look like? A sketch:
  - Requires exponential time (using a deterministic TM)
  - Remember: this was just a sketch

---

Height of the tree is polynomial in the input size
P, NP

- **P** = problems that can be solved in polynomial time on a deterministic Turing Machine.
- **NP** = problems that can be solved in polynomial time on a non-deterministic Turing machine.

NP-Complete

- **Definition:** A decision problem $A$ is **NP-complete** iff
  - $A \in NP$
  - For all $B \in NP$, $B \leq^p A$
    - i.e., Every problem in NP can be reduced to $A$ by a polynomial-time Turing reduction
    - $A$ is at least as hard as anything in NP
    - An algorithm for $A$ can be used as a subroutine to solve any problem in NP (by way of reduction)

Efficiency Terminology

- **Efficient** means “polynomial time”
- **Tractable** is a synonym for efficient
- NP-complete problems have no known efficient solutions/algorithms
- We call a problem “hard” or “intractable” iff that problem has no polynomial solution.

*Caveat: if P $\neq$ NP

NP-Complete vs. P

<table>
<thead>
<tr>
<th>Hard problems (NP-complete)</th>
<th>Easy problems (in P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3SAT</td>
<td>3SAT, HORN SAT</td>
</tr>
<tr>
<td>TRAVELING SALESMAN PROBLEM</td>
<td>MINIMUM SPANNING TREE</td>
</tr>
<tr>
<td>LONGEST PATH</td>
<td>SHORTEST PATH</td>
</tr>
<tr>
<td>3D MATCHING</td>
<td>BIPARTITE MATCHING</td>
</tr>
<tr>
<td>KNAPSACK $(\alpha/M)$</td>
<td>UNARY KNAPSACK $(\alpha)$</td>
</tr>
<tr>
<td>INDEPENDENT SET</td>
<td>INDEPENDENT SET ON TREES</td>
</tr>
<tr>
<td>INTEGER LINEAR PROGRAMMING</td>
<td>LINEAR PROGRAMMING</td>
</tr>
<tr>
<td>HUBRATA PATH</td>
<td>EULER PATH</td>
</tr>
<tr>
<td>BALANCED CUT</td>
<td>MINIMUM CUT</td>
</tr>
</tbody>
</table>

P, NP, NP-Complete

- **P** = problems that can be solved in polynomial time on a deterministic Turing Machine.
- **NP** = problems that can be solved in polynomial time on a non-deterministic Turing machine.
- **NP-Complete** = problems that can be reduced to any other problem in NP by a polynomial-time Turing reduction.

Reductions

- $A \leq^p B$ means: “$A$ is polynomial-time Turing reducible to $B$”

Examples?

- **Algorithm for $A$**
- **Algorithm for $B$**
- **Solution for $f(i)$**
- **No solution to $f(i)$**
NP-Complete

- Recall our Definition: A decision problem A is NP-complete iff
  - A ∈ NP
  - For all B ∈ NP, B ≤ₚ A

- Can you use this definition directly to prove that a problem is in NP-Complete?
- Why not?

Proving a Problem is NP-Complete

- Let X be an NP-Complete problem
  - e.g., 3CNF SAT
  - Consider a decision problem Z in NP such that X ≤ₚ Z
  - Then?

Thus, Z is also NP-Complete

To prove Z is NP-Complete:
- 1. Show that Z is in NP
- Easy
- 2. Show that a known NP-Complete problem can be reduced to Z!
  - More fun!

P = NP?

- What if we could identify just one problem in NP-Complete with an efficient solution?
  - Then there is an efficient solution to all problems in NP.
  - Then NP ⊆ P
  - We already know P ⊆ NP
  - So we would conclude P=NP!

Where Next?

- Approximation Algorithms!
  - Intelligent Search
  - Branch and Bound
  - Randomized (Probabilistic) Algorithms
- These approaches are general
  - Many can give approximate answers before they run to completion.