Languages and automata (Chapters 2–4)

- A hierarchy of models of computation
- Nondeterminism
- Turing machines
- Problem set on automata (needs to be graded. . .)

Undecidability (Chapter 5)

- Definition of undecidability
- The Halting Problem
- Reduction proofs
- Problem set on undecidability proofs (due Wednesday after break)

NP-completeness (Chapters 6 and 7)

- The class $\mathcal{P}$, definition of tractability (§6.1)
- Problems: Reachability , Euler cycle, Hamiltonian cycle, Traveling Salesman, Independent Set, Clique, Node Cover, Integer Partition (§6.2)
- Boolean Satisfiability (§6.3)
- The class $\mathcal{NP}$, $\mathcal{NP}$-completeness, and proofs (§6.4)
- More problems, practice, and applications for $\mathcal{NP}$-completeness
- Problem set on $\mathcal{NP}$-completeness (Due Thurs, Dec 12)

Schedule (recent and imminent)

Date	**Reading**	**In class**
Fri, Nov 22	6 (whole chapter)	Sections 5.(4,6,7), ~~definitions from 6.(1 & 2)~~
Mon, Nov 25	Reread 6.1 Reread 6.2 through pg 282	6.1 Definition of class $\mathcal{P}$ etc 6.2 $\textsc{Reachability}$, $\textsc{HamCycle}$
Mon, Dec 2	Reread rest of 6.2 Reread 6.3	6.2 $\textsc{TSP}$, $\textsc{IndependentSet}$ $\textsc{Clique}$, $\textsc{Partition}$ 6.3 Boolean satisfiability
Wed, Dec 4	Reread 6.4 Read 7.2	6.4 The class NP 7.1 Polynomial-time reductions

Definition 6.1.1: A Turing machine M is **polynomially bounded** if

$\exists$ $p(n)$, a polynomial function such that
    $\forall$ $x \in \Sigma*$
        $\forall$ $C \in$ (set of configurations), either
            C is unreachable from $(s, \triangleright\underline{\sqcup}w)$, or
            $(s, \triangleright\underline{\sqcup}w) \vdash_M^k C$, where $k \leq p(|x|)$

A language is **polynomially decidable** if

$\exists$ M, a Turing machine that decides the language, such that
    $\exists$ $p(n)$, a polynomial function such that
        $\forall$ $x \in \Sigma*$
            $\forall$ $C \in$ (set of configurations), either
                C is unreachable from $(s, \triangleright\underline{\sqcup}w)$, or
                $(s, \triangleright\underline{\sqcup}w) \vdash_M^k C$, where $k \leq p(|x|)$

LP pg 276

Reachability. Given a graph G and vertices v_i and v_j, find a path from v_i to v_j.

Language version: Does there exist a path from v_i to v_j?

$$\{\kappa(G)\mathbf{b}(i)\mathbf{b}(j) \mid \exists \text{ path in } G \text{ from } v_i \text{ to } v_j\}$$

One of the main points that will emerge from the discussion that follows is that *the precise details of encodings rarely matter*.

Since it is easy to see that $m = O(n^3)$, this is yet another inconsequential inaccuracy, one that will not interfere with the issues that we deem important.

LP pg 280

Euler cycle. Given a graph G, is there a closed path (cycle) that uses each edge exactly once? (Repeated vertices are okay.)

$$\{\kappa(G) \mid \exists \text{ a cycle that uses each edge exactly once}\}$$

Euler's result: A graph has an Euler cycle if all non-isolated pairs are reachable and each node's in-degree equals its out-degree.

Hamiltonian Cycle. Given a graph G, is there a cycle that passes through each vertex exactly once? (Unused edges are okay.)

$$\{\kappa(G) \mid \exists \text{ a cycle that visits each vertex exactly once}\}$$

Despite the superficial similarity between the two problems, Euler Cycle and Hamiltonian Cycle, there appears to be a world of difference between them. After one and a half centuries of scrutiny by many talented mathematicians, no one has discovered a polynomial algorithm for Hamiltonian Cycle.

LP pg 282

Traveling Salesman. Given a complete weighted graph, find a simple cycle with with least weight.

Optimization version: Given $n \in \mathbb{N}$ and an $n \times n$ distance matrix $d_{i,j}$, and letting π range over permutations of $\{1, 2, \ldots n\}$, define $c(\pi) = \left(\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)}\right) + d_{\pi(n),\pi(1)}$ Find *pi* to minimize $c(\pi)$.

Budgeted version: Given $n \in \mathbb{N}$, an $n \times n$ distance matrix $d_{i,j}$, and $B \in \mathbb{W}$, and using π and $c(\pi)$ as above, find a permutation π such that $c(\pi) \leq B$.

Language version:

$$\{(n, d_{i,j}, B) \mid \exists\ \pi \text{ such that } c(\pi) \leq B\}$$

Independent Set. Given an undirected graph $G = (V, E)$, find a maximal set of vertices $C \subseteq V$ such that for all $v_i, v_j \in C$, $(v_i, v_j) \notin E$.

Language version: Does an independent set of a given *goal* size exist?

$$\{(\kappa(G), K) \mid \exists\ C \subseteq V \ \text{ such that } \ |C| \geq K \text{ and } \forall\ v_i, v_j \in C,\ (v_i, v_j) \notin E\}$$

. . . yet another simply stated problem for which, despite prolonged and intense interest by researchers, no polynomial-time algorithm has been found. *LP pg 283*

Clique. Given an undirected graph $G = (V, E)$, find a maximal set of vertices $C \subseteq V$ such that for all $v_i, v_j \in C$, $(v_i, v_j) \in E$.

Language version: Does a clique of a given *goal* size exist?

$$\{(\kappa(G), K) \mid \exists\ C \subseteq V \ \text{ such that } \ |C| \geq K \text{ and } \forall\ v_i, v_j \in C,\ (v_i, v_j) \in E\}$$

Node Cover. Given an undirected graph $G = (V, E)$, find a minimal set of vertices $C \subseteq V$ such that for every edge in E, at least one endpoint of E is in C.

Language version: Does a cover of a give *budget* size exist?

$$(\kappa(G), B) \mid \exists\ C \subseteq V \text{ such that } |C| \leq B \text{ and } C \text{ covers all edges in } E\}$$

We can think of the [vertices] of an undirected graph as the rooms of a museum, and each edge as a long straight corridor that joins two rooms. Then the Node Cover problem may be useful in assigning as few guards as possible to the rooms, so that all corridors can be seen by a guard. *LP pg 284*

Integer Partition. Given a set of whole numbers $\{a_1, a_2, \ldots a_n\}$, find a subset indexed by $P \subseteq \mathbb{N}_n$ such that $\sum_{i \in P} a_i = \sum_{i \in \mathbb{N}_n - P} a_i$.

Language version: Does a two-set partition of a given set exist with equal sums?

$$\{S \mid \exists\ A \subseteq S \text{ such that } \sum_{a \in A} a = \sum_{b \in S - A} b\}$$

An algorithm exists (pg 285), but is it polynomial?

Boolean Satisfiability (SAT) and family

A **literal** is an occurrence of a variable or its negation: x or $\sim x$

A **clause** is a disjunction of literals: $x_1 \vee x_2 \vee \sim x_3$

A **formula** is a conjunction of clauses: $(x_1 \vee \sim x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee \sim x_5)$

A **truth assignment** is a mapping from variables to $\{\top, \bot\}$

A truth assignment **satisfies** a formula is $\forall$ clauses $\exists$ a true literal

The **Satisfiability** problem is, given a formula, does a satisfying truth assignment exist?

2-SAT: Given a formula in which each clause has no more than two literals . . .

3-SAT: Given a formula in which each clause has no more than three literals . . .

Claim: This algorithm produces a satisfying truth assignment iff one exists.

Proof. *($\Rightarrow$) [If the algorithm returns a truth assignment, that assignment indeed satisfies the given formula.]*

In the original/initial call to `purge`, any individual variable assignment that results must be part of any satisfying truth assignment, since the formula cannot be satisfied without the variable assignments done by `purge`.

Invariant *for the main loop: The (partial) assignment to the variables is part of a satisfying (complete) truth assignment, iff one exists.*

Initialization: *Implied by what is said above.*

Maintenance: *Suppose the partial assignment at the beginning of an iteration is part of a complete satisfying truth assignment. This iteration assigns to one variable. If that assignment were not part of a CSTA that also includes the current partial assignment, it would be rejected by the call to purge. Hence the updated partial assignment is also part of a CSTA.*

Termination. *There is at most one iteration for each variable. Since there are a finite number of variables, the loop terminates. When the loop terminates, all variables are assigned, and, by the loop invariant, that assignment is "part of" a CSTA. There fore the assignment is a CSTA.*

($\Leftarrow$) [If a truth assignment exists, the algorithm returns one.]

Suppose a truth assignment exists, and suppose the algorithm doesn't find one. From that we can derive a contradiction. $\square$

Why don't the polynomial-time algorithms for 2-SAT work for 3-SAT?

The `purge` routine is based on the premise that if a guess doesn't fail, then it is safe.

$$(\sim x_1 \vee x_2 \vee \sim x_3) \wedge (\sim x_1 \vee \sim x_2 \vee \sim x_3) \wedge (\sim x_1 \vee x_2 \vee x_3) \wedge (\sim x_1 \vee \sim x_2 \vee x_3)$$

What if we guess $x_{:} = \top$?