Midterm 2

- When?
  - April 3rd from 8PM to 10PM (2 hours)
- Where?
  - EE 170
- What can I bring?
  - 3x5 inch index card with your notes (double sided)
  - No electronics (phones, computers, calculators etc.)

Chapter 9

PSPACE: A Class of Problems Beyond NP

9.1 PSPACE

Geography Game

Geography. Alice names capital city $c$ of country she is in. Bob names a capital city $c'$ that starts with the letter on which $c$ ends. Alice and Bob repeat this game until one player is unable to continue. Does Alice have a forced win?

Ex. Budapest $\rightarrow$ Tokyo $\rightarrow$ Ottawa $\rightarrow$ Ankara $\rightarrow$ Amsterdam $\rightarrow$ Moscow $\rightarrow$ Washington $\rightarrow$ Nairobi $\rightarrow$ ...

Geography on graphs. Given a directed graph $G = (V, E)$ and a start node $s$, two players alternate turns by following, if possible, an edge out of the current node to an unvisited node. Can first player guarantee to make the last legal move?

Remark. Some problems (especially involving 2-player games and AI) defy classification according to P, EXPTIME, NP, and NP-complete.
PSPACE

P. Decision problems solvable in polynomial time.
PSPACE: Decision problems solvable in polynomial space.

Observation. \( P \subseteq \text{PSPACE} \).

Poly-time algorithms can consume only polynomial space

---

9.3 Quantified Satisfiability

QSAT. Let \( \Phi(x_1, \ldots, x_n) \) be a Boolean CNF formula. Is the following propositional formula true?

\[
\begin{align*}
\exists x_1 & \quad \forall x_2 \quad \exists x_3 \quad \ldots \quad \exists x_n \quad \Phi(x_1, \ldots, x_n)
\end{align*}
\]

Intuition. Amy picks truth value for \( x_1 \), then Bob for \( x_2 \), then Amy for \( x_3 \), and so on. Can Amy satisfy it no matter what Bob does?

Ex. \( (x_1 \lor x_2) \land (x_1 \lor \overline{x_2}) \land (x_2 \lor \overline{x_3} \lor x_3) \)

Yes. Amy sets \( x_1 \) true; Bob sets \( x_2 \); Amy sets \( x_3 \) to be same as \( x_2 \).

Ex. \( (x_1 \lor x_2) \land (x_1 \lor \overline{x_2}) \land (x_2 \lor \overline{x_3} \lor x_3) \)

No. If Amy sets \( x_1 \) false; Bob sets \( x_2 \) false; Amy loses.

If Amy sets \( x_1 \) true; Bob sets \( x_2 \) true; Amy loses.

---

QSAT is in PSPACE

Theorem. QSAT \( \in \text{PSPACE} \).

Proof. Recursively try all possibilities.

- Only need one bit of information from each subproblem.
- Amount of space is proportional to depth of function call stack.

---

9.4 Planning Problem
15-Puzzle

8-puzzle, 15-puzzle. [Sam Loyd 1870s]

- Board: 3-by-3 grid of tiles labeled 1-8.
- Legal move: slide neighboring tile into blank (white) square.
- Find sequence of legal moves to transform initial configuration into goal configuration.

![Initial configuration: 1 2 3 4 5 6 7 8 goal configuration: 1 2 3 4 5 6 7 8](image)

Planning Problem: 8-Puzzle

Planning example. Can we solve the 8-puzzle?

### Conditions
- \( C_{ij}, 1 \leq i, j \leq 9 \)

### Initial state
- \( c_0 = \{C_{11}, C_{22}, \ldots, C_{66}, C_{78}, C_{87}, C_{99}\} \)

### Goal state
- \( c^* = \{C_{11}, C_{22}, \ldots, C_{66}, C_{77}, C_{88}, C_{99}\} \)

### Operators
- Precondition to apply \( O_i = \{C_{11}, C_{22}, \ldots, C_{66}, C_{78}, C_{87}, C_{99}\} \)
- After invoking \( O_i \), conditions \( C_{79} \) and \( C_{97} \) become true.
- After invoking \( O_i \), conditions \( C_{78} \) and \( C_{99} \) become false.

### Solution
- No solution to 8-puzzle or 15-puzzle!

### Planning Problem: Binary Counter

Planning example. Can we increment an n-bit counter from the all-zeros state to the all-ones state?

### Conditions
- \( C_i, \ldots, C_n \)
- \( C_i \) corresponds to bit \( i = 1 \)

### Initial state
- \( c_0 = \phi \)

### Goal state
- \( c^* = \{C_1, \ldots, C_n\} \)

### Operators
- \( O_1, \ldots, O_n \)
  - To invoke operator \( O_i \), must satisfy \( C_1, \ldots, C_{i-1} \)
  - After invoking \( O_i \), condition \( C_i \) becomes true.
  - After invoking \( O_i \), conditions \( C_1, \ldots, C_{i-1} \) become false.

### Solution
- \( \{\} \rightarrow \{C_1\} \rightarrow \{C_2\} \rightarrow \{C_3\} \rightarrow \ldots \)

### Observation
- Any solution requires at least \( 2^n - 1 \) steps.

Planning Problem: In Exponential Space

Configuration graph \( G \).
- Include node for each of \( 2^n \) possible configurations.
- Include an edge from configuration \( c' \) to configuration \( c'' \) if one of the operators can convert from \( c' \) to \( c'' \).

### Claim
- PLANNING is in \( \text{EXPTIME} \).

### Proof
- Run BFS to find path from \( c_0 \) to \( c^* \) in configuration graph.

### Note
- Configuration graph can have \( 2^n \) nodes, and shortest path can be of length \( 2^n - 1 \).

Planning Problem: 15-Puzzle

### Conditions
- \( C = \{C_1, \ldots, C_9\} \)

### Initial configuration
- Subset \( c_0 \subseteq C \) of conditions initially satisfied.

### Goal configuration
- Subset \( c^* \subseteq C \) of conditions we seek to satisfy.

### Operators
- \( O = \{O_1, \ldots, O_9\} \)
  - To invoke operator \( O_i \), must satisfy certain prereq conditions.
  - After invoking \( O_i \), certain conditions become true, and certain conditions become false.

### PLANNING
- Is it possible to apply sequence of operators to get from initial configuration to goal configuration?
**Planning Problem: In Polynomial Space**

**Theorem.** PLANNING is in PSPACE.

**Pf.**
- Suppose there is a path from $c_1$ to $c_2$ of length $L$.
- Path from $c_1$ to midpoint and from midpoint to $c_2$ are each $\leq L/2$.
- Enumerate all possible midpoints.
- Apply recursively. Depth of recursion = $\log_2 L$.

```
boolean hasPath(c1, c2, L) {
  if (L ≤ 1) return correct answer
  foreach configuration c' {
    boolean x = hasPath(c1, c', L/2)
    boolean y = hasPath(c', c2, L/2)
    if (x and y) return true
  }
  return false
}
```

---

**9.5 PSPACE-Complete**

**PSPACE-Complete**

PSPACE: Decision problems solvable in polynomial space.

**PSPACE-Complete.** Problem $Y$ is PSPACE-complete if (i) $Y$ is in PSPACE and (ii) for every problem $X$ in PSPACE, $X \leq_p Y$.

**Theorem.** [Stockmeyer-Meyer 1973] QSAT is PSPACE-complete.

**PSPACE ⊆ EXPTIME.**

**Pf.** Previous algorithm solves QSAT in exponential time, and QSAT is PSPACE-complete.

**Summary.** $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$.

---

**Competitive Facility Location**

**Input.** Graph with positive edge weights, and target $B$.

**Game.** Two competing players alternate in selecting nodes. Not allowed to select a node if any of its neighbors has been selected.

**Competitive facility location.** Can second player guarantee at least $B$ units of profit? (Player one might play vindictively to minimize $B$’s profit)

**Claim.** COMPETITIVE-FACILITY is PSPACE-complete.

**Pf.**
- To solve in poly-space, use recursion like QSAT, but at each step there are up to $n$ choices instead of 2.
- To show that it's complete, we show that QSAT polynomial reduces to it. Given an instance of QSAT, we construct an instance of COMPETITIVE-FACILITY such that player 2 can force a win iff QSAT formula is false.
Competitive Facility Location

Construction. Given instance \( \phi(x_1, \ldots, x_n) = C_1 \land C_2 \land \ldots \land C_k \) of QSAT:
- Include a node for each literal and its negation and connect them.
- at most one of \( x_i \) and its negation can be chosen.
- Choose \( c \geq k+2 \), and put weight \( c_i \) on literal \( x_i \) and its negation;
- ensure variables are selected in order \( x_n, x_{n-1}, \ldots, x_1 \).
- As is, player 2 will lose by 1 unit: \( c_n + c_{n-3} + \ldots + c_4 + c_2 + 1 \).

Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you’re unlikely to find a poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem to optimality.
- Solve problem in poly-time.
- Solve arbitrary instances of the problem.

\( \rho \)-approximation algorithm.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem.
- Guaranteed to find solution within ratio \( \rho \) of true optimum.

Challenge. Need to prove a solution’s value is close to optimum, without even knowing what optimum value is.

11.1 Load Balancing

Input. \( m \) identical machines: \( n \) jobs, job \( j \) has processing time \( t_j \).
- Job \( j \) must run contiguously on one machine.
- A machine can process at most one job at a time.

Def. Let \( J(i) \) be the subset of jobs assigned to machine \( i \). The load of machine \( i \) is \( L_i = \sum_{j \in J(i)} t_j \).

Def. The makespan is the maximum load on any machine \( L = \max_i L_i \).

Load balancing. Assign each job to a machine to minimize makespan.
**List-Scheduling Algorithm**

- Consider $n$ jobs in some fixed order.
- Assign job $j$ to machine whose load is smallest so far.

**Implementation**

Given $m$, $n$, $t_1, t_2, \ldots, t_m$:

1. Initialize arrays $L_i$ and $J_i$ for each machine $i$.
2. For $i = 1$ to $m$:
   - $L_i = 0$,
   - $J_i = \emptyset$.
3. For $j = 1$ to $n$:
   - $i = \text{argmin}_k L_k$,
   - $J_i = J_i \cup \{j\}$,
   - $L_i = L_i + t_j$.
4. Return $J(1), \ldots, J(m)$.

**Load Balancing: List Scheduling Analysis**

**Theorem.** Greedy algorithm is a 2-approximation.

**Proof.** Consider load $L_i$ of bottleneck machine $i$.

- Let $j$ be last job scheduled on machine $i$.
- When job $j$ assigned to machine $i$, $i$ had smallest load. Its load before assignment is $L_i - t_j = L_i$ for all $1 \leq k \leq m$.

**Lemma 1**

$$L_i = \sum_{k=1}^{i} t_k$$

**Lemma 2**

$$L_i \leq L'^*$$

**Example:**

- $m$ machines, $m(m-1)$ jobs length 1 jobs, one job of length $m$.
- Machine 2 idle,
- Machine 3 idle,
- Machine 4 idle,
- Machine 5 idle,
- Machine 6 idle,
- Machine 7 idle,
- Machine 8 idle,
- Machine 9 idle,
- Machine 10 idle.

**List Scheduling Makespan:** 10

**Optimal Makespan:** 10
Load Balancing: LPT Rule

Longest processing time (LPT). Sort \( n \) jobs in descending order of processing time, and then run list scheduling algorithm.

```plaintext
LPT-List-Scheduling(m, n, t_1, t_2, ..., t_n) {
  Sort jobs so that \( t_1 \geq t_2 \geq ... \geq t_n \)
  for i = 1 to m {
    \( L_i \) = \( 0 \)
    \( J(i) \) = \( \phi \)
  }
  for j = 1 to n {
    \( i = \text{argmin}_k L_k \)
    \( J(i) \) = \( J(i) \cup \{ j \} \)
    \( L_i \) = \( L_i + t_j \)
  }
  return \( J(1), \ldots, J(m) \)
}
```

- jobs assigned to machine \( i \)
- load on machine \( i \)
- machine \( i \) has smallest load
- assign job \( j \) to machine \( i \)
- update load of machine \( i \)

Observation. If at most \( m \) jobs, then list-scheduling is optimal.
Proof. Each job put on its own machine.

Lemma 3. If there are more than \( m \) jobs, \( L^* \geq 2t_{m+1} \).
Proof. Consider first \( m+1 \) jobs \( t_1 \geq t_2 \geq ... \geq t_{m+1} \).
Since the \( t_i \)'s are in descending order, each takes at least \( t_{m+1} \) time.
There are \( m+1 \) jobs and \( m \) machines, so by pigeonhole principle, at least one machine gets two jobs.

Theorem. LPT rule is a 3/2 approximation algorithm.
Proof. Same basic approach as for list scheduling.

\[ \frac{L_i}{L^*} \leq \frac{3}{2} \]

Lemma 3 (by observation, can assume number of jobs > \( m \)).

Q. Is our 3/2 analysis tight?
A. No.


Q. Is Graham’s 4/3 analysis tight?
A. Essentially yes.

Ex. \( m \) machines, \( n = 2m+1 \) jobs, 2 jobs of length \( m+1, m+2, \ldots, 2m-1 \) and one job of length \( m \).