CS 580: Algorithm Design and Analysis

Jeremiah Blocki
Purdue University
Spring 2018
Chapter 9

PSPACE: A Class of Problems Beyond NP
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 → Tokyo → Ottawa → Ankara → Amsterdam → Moscow → Washington → Nairobi → ...

**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.
9.1 PSPACE
PSPACE

P. Decision problems solvable in polynomial time.

PSPACE. Decision problems solvable in polynomial space.

Observation. $P \subseteq \text{PSPACE}$. \\
\[ \uparrow \]
\[ \text{poly-time algorithm can consume only polynomial space} \]
Binary counter. Count from 0 to $2^n - 1$ in binary.

Algorithm. Use $n$ bit odometer.

Claim. 3-SAT is in PSPACE.

Pf.
- Enumerate all $2^n$ possible truth assignments using counter.
- Check each assignment to see if it satisfies all clauses. ∎

Theorem. $\text{NP} \subseteq \text{PSPACE}$.

Pf. Consider arbitrary problem $Y$ in NP.
- Since $Y \leq_p 3$-SAT, there exists algorithm that solves $Y$ in poly-time plus polynomial number of calls to 3-SAT black box.
- Can implement black box in poly-space. ∎
9.3 Quantified Satisfiability
Quantified Satisfiability

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

$$\exists x_1 \ \forall x_2 \ \exists x_3 \ \forall x_4 \ldots \ \forall x_{n-1} \ \exists x_n \ \Phi(x_1, \ldots, x_n)$$

↑

assume $n$ is odd

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

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

**Yes.** Amy sets $x_1$ true; Bob sets $x_2$; Amy sets $x_3$ to be same as $x_2$.

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

**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.
Theorem. QSAT ∈ PSPACE.

Pf. Recursively try all possibilities.

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

\[
\begin{align*}
\therefore & \quad \because \\
x_1 = 0 & \quad \therefore & \quad \because \\
x_2 = 0 & \quad x_2 = 1 & \quad \therefore & \quad \because \\
x_3 = 0 & \quad x_3 = 1 & \quad \therefore & \quad \because \\
\Phi(0, 0, 0) & \quad \Phi(0, 0, 1) & \quad \Phi(0, 1, 0) & \quad \Phi(0, 1, 1) & \quad \Phi(1, 0, 0) & \quad \Phi(1, 0, 1) & \quad \Phi(1, 1, 0) & \quad \Phi(1, 1, 1)
\end{align*}
\]
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.

![Diagram](image-url)

initial configuration

move 12

goal configuration
Planning Problem

**Conditions.** Set $C = \{ C_1, \ldots, C_n \}$.

**Initial configuration.** Subset $c_0 \subseteq C$ of conditions initially satisfied.

**Goal configuration.** Subset $c^* \subseteq C$ of conditions we seek to satisfy.

**Operators.** Set $O = \{ O_1, \ldots, O_k \}$.

- 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?

**Examples.**
- 15-puzzle.
- Rubik’s cube.
- Logistical operations to move people, equipment, and materials.
Planning Problem: 8-Puzzle

Planning example. Can we solve the 8-puzzle?

Conditions. $C_{ij}, 1 \leq i, j \leq 9$. $c_{ij}$ means tile $i$ is in square $j$

Initial state. $c_0 = \{C_{11}, C_{22}, ..., C_{66}, C_{78}, C_{87}, C_{99}\}$.

Goal state. $c^* = \{C_{11}, C_{22}, ..., C_{66}, C_{77}, C_{88}, C_{99}\}$.

Operators.
- Precondition to apply $O_i = \{C_{11}, C_{22}, ..., C_{66}, C_{78}, C_{87}, C_{99}\}$.
- 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!
**Diversion: Why is 8-Puzzle Unsolvable?**

**8-puzzle invariant.** Any legal move preserves the parity of the number of pairs of pieces in reverse order (inversions).

![Diagram showing 8-puzzle invariant with examples](image)

- **3 inversions**
  - Original state: 1-3, 2-3, 7-8
  - Move: 7-8
  - Result: 1-3, 2-3, 7-8, 5-8, 5-6

- **0 inversions**
  - Original state: 0 inversions
  - Move: 7-8
  - Result: 1 inversion: 7-8
Planning Problem: Binary Counter

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

Conditions. \( C_1, \ldots, C_n \). \( \leftarrow C_i \) corresponds to bit \( i = 1 \)

Initial state. \( c_0 = \emptyset \). \( \leftarrow \) all 0s

Goal state. \( c^* = \{C_1, \ldots, C_n\} \). \( \leftarrow \) all 1s

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. \( \leftarrow \) set bit \( i \) to 1
- After invoking \( O_i \), conditions \( C_1, \ldots, C_{i-1} \) become false.

Solution. \( \{\} \Rightarrow \{C_1\} \Rightarrow \{C_2\} \Rightarrow \{C_1, C_2\} \Rightarrow \{C_3\} \Rightarrow \{C_3, C_1\} \Rightarrow \ldots \)

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 $c''$.

**PLANNING.** Is there a path from $c_0$ to $c^*$ in configuration graph?

**Claim.** PLANNING is in EXPTIME.
**Pf.** 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 $= 2^n - 1$.

↑

binary counter
Theorem. PLANNING is in PSPACE.

Pf.

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

boolean hasPath($c_1$, $c_2$, $L$) {
   if ($L \leq 1$) return correct answer
   enumerate using binary counter
   foreach configuration $c'$ {
      boolean $x$ = hasPath($c_1$, $c'$, $L/2$)
      boolean $y$ = hasPath($c'$, $c_2$, $L/2$)
      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.

**Theorem.** $\text{PSPACE} \subseteq \text{EXPTIME}$.

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

**Summary.** $\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$.

it is known that $\text{P} \neq \text{EXPTIME}$, but unknown which inclusion is strict; conjectured that all are
More PSPACE-complete problems.

- **Competitive facility location.**
- Natural generalizations of games.
  - Othello, Hex, Geography, Rush-Hour, Instant Insanity
  - Shanghai, go-moku, Sokoban
- Given a memory restricted Turing machine, does it terminate in at most $k$ steps?
- Do two regular expressions describe different languages?
- Is it possible to move and rotate complicated object with attachments through an irregularly shaped corridor?
- Is a deadlock state possible within a system of communicating processors?
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?

Yes if $B = 20$; no if $B = 25$. 
Competitive Facility Location

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 true.
Competitive Facility Location

Construction. Given instance $\Phi(x_1, \ldots, x_n) = C_1 \land C_1 \land \ldots C_k$ 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;
  set $B = c^{n-1} + c^{n-3} + \ldots + c^4 + c^2 + 1$.
  - ensures variables are selected in order $x_n, x_{n-1}, \ldots, x_1$.
- As is, player 2 will lose by 1 unit: $c^{n-1} + c^{n-3} + \ldots + c^4 + c^2$.
Construction. Given instance $\Phi(x_1, \ldots, x_n) = C_1 \land C_1 \land \ldots \land C_k$ of QSAT.

- Give player 2 one last move on which she can try to win.
- For each clause $C_j$, add node with value 1 and an edge to each of its literals.
- Player 2 can make last move iff truth assignment defined alternately by the players failed to satisfy some clause. □
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
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.

```
List-Scheduling(m, n, t_1, t_2, ..., t_n) {
    for i = 1 to m {
        L_i ← 0 ← load on machine i
        J(i) ← ∅ ← jobs assigned to machine i
    }

    for j = 1 to n {
        i = argmin_k L_k ← machine i has smallest load
        J(i) ← J(i) ∪ {j} ← assign job j to machine i
        L_i ← L_i + t_j ← update load of machine i
    }

    return J(1), ..., J(m)
}
```

Implementation. $O(n \log m)$ using a priority queue.
Load Balancing: List Scheduling Analysis

**Theorem.** [Graham, 1966] Greedy algorithm is a 2-approximation.
- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan $L^*$. 

**Lemma 1.** The optimal makespan $L^* \geq \max_j t_j$.
**Pf.** Some machine must process the most time-consuming job. ★

**Lemma 2.** The optimal makespan $L^* \geq \frac{1}{m} \sum_j t_j$
**Pf.**
- The total processing time is $\sum_j t_j$.
- One of $m$ machines must do at least a $1/m$ fraction of total work. ★
Theorem. Greedy algorithm is a 2-approximation.

Pf. 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 $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ for all $1 \leq k \leq m$. 
Theorem. Greedy algorithm is a $2$-approximation.

Pf. 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 $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.
- Sum inequalities over all $k$ and divide by $m$:

$$L_i - t_j \leq \frac{1}{m} \sum_{k=1}^{m} L_k$$

$$= \frac{1}{m} \sum_{k=1}^{n} t_k \leq L^*$$

Now $L_i = (\overline{L_i - t_j}) + \underbrace{t_j}_{\leq L^*} \leq 2L^*$. 

\[ \mathbf{Lemma~2} \]
Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

\[
\begin{array}{c|c}
\text{m=10} & \text{machine 2 idle} \\
& \text{machine 3 idle} \\
& \text{machine 4 idle} \\
& \text{machine 5 idle} \\
& \text{machine 6 idle} \\
& \text{machine 7 idle} \\
& \text{machine 8 idle} \\
& \text{machine 9 idle} \\
& \text{machine 10 idle} \\
\end{array}
\]

list scheduling makespan = 19
Q. Is our analysis tight?
A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

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.

\[
\text{LPT-List-Scheduling}(m, n, t_1, t_2, \ldots, t_n) \{
\text{Sort} \text{ jobs so that } t_1 \geq t_2 \geq \ldots \geq t_n \\
\text{for } i = 1 \text{ to } m \{ \\
L_i \leftarrow 0 \quad \text{load on machine } i \\
J(i) \leftarrow \emptyset \quad \text{jobs assigned to machine } i \\
\} \\
\text{for } j = 1 \text{ to } n \{ \\
i = \text{argmin}_k L_k \quad \text{machine } i \text{ has smallest load} \\
J(i) \leftarrow J(i) \cup \{j\} \quad \text{assign job } j \text{ to machine } i \\
L_i \leftarrow L_i + t_j \quad \text{update load of machine } i \\
\} \\
\text{return } J(1), \ldots, J(m) \\
\}
\]
**Load Balancing: LPT Rule**

**Observation.** If at most m jobs, then list-scheduling is optimal.

**Pf.** Each job put on its own machine. ▪

**Lemma 3.** If there are more than m jobs, $L^* \geq 2t_{m+1}$.

**Pf.**
- Consider first $m+1$ jobs $t_1, \ldots, t_{m+1}$.
- 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.

**Pf.** Same basic approach as for list scheduling.

$$L_i = \left( L_i - t_j \right) + t_j \leq L^* \leq \frac{3}{2} L^*. $$

Lemma 3
(by observation, can assume number of jobs > m)
Load Balancing: LPT Rule

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

**Theorem.** [Graham, 1969] LPT rule is a 4/3-approximation.
**Pf.** More sophisticated analysis of same algorithm.

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, ..., 2m-1 and one job of length m.