Chapter 9 PSPACE: A Class of Problems Beyond NP

Jeremiah Blocki
Purdue University
Spring 2018

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 → Tokyo → Ottawa → 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.

PSPACE

P. Decision problems solvable in polynomial time.

PSPACE. Decision problems solvable in polynomial space.

Observation. P ⊆ PSPACE.

poly-time algorithm can consume only polynomial space

PSPACE

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. NP ⊆ PSPACE.

Pf. Consider arbitrary problem Y in NP.

Since Y ≤_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

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

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

Quantified Satisfiability

Theorem. QSAT $\in$ 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.

9.4 Planning Problem

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 preconditions.
- 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.

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.
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 \( O_i \equiv (C_{11}, C_{22}, \ldots, C_{66}, C_{78}, C_{87}, C_{99}) \).
- 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-zeroes state to the all-ones state?

Conditions. \( C_1, \ldots, C_n \).

Initial state. \( c_0 = \emptyset \).

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

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.

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: In Polynomial Space

Theorem. PLANNING is in PSPACE.

Proof.

- Suppose there is a path from \( c_i \) to \( c_j \) of length \( L \).
- Path from \( c_i \) to midpoint and from midpoint to \( c_j \) are each \( \leq L/2 \).
- Enumerate all possible midpoints.
- Apply recursively. Depth of recursion \( = \log_2 L \).

---

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. PSPACE \subseteq EXPTIME.

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

\[ \text{Summary. } P \subseteq NP \subseteq PSPACE \subseteq \text{EXPTIME.} \]

\[ \text{It is known that } P \neq \text{EXPTIME, but whether which inclusion is strict is unknown.} \]

PSPACE-Complete Problems

More PSPACE-complete problems.
- Competitive facility location.
- Natural generalizations of games.
  - Othello, Hex, Geography, Rush Hour, Instant Insanity
  - Shanghai, go-moku, Sokoban
- Given a 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 a 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?

\[ \text{Yes if } B = 20, \text{ no if } B = 25. \]

\[ \text{Claim. COMPETITIVE-FACILITY is PSPACE-complete.} \]

Pf.
- To solve in poly-space, use recursion like QSAT, but at each step there are at most \( k \) 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.

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 \( c : k \mapsto 2^i \), and put weight \( c \) on literal \( x_i \) and its negation.
- Set \( B = c^{n-2} + c^{n-3} + \ldots + c^4 + c^2 + 1 \).
- Ensures variables are selected in order \( x_n, x_{n-1}, \ldots, x_1 \).
- As is, player 2 will lose by 1 unit: \( c^{n-2} + c^{n-3} + \ldots + c^4 + c^2 \).

\[ \text{Construction. Given instance } \Phi(x_1, \ldots, x_n) = C_1 \land C_2 \land \ldots \land C_k \text{ of QSAT.} \]

- 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.

\[ \text{Construction. Given instance } \Phi(x_1, \ldots, x_n) = C_1 \land C_2 \land \ldots \land C_k \text{ of QSAT.} \]

- 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.

\[ \text{Construction. Given instance } \Phi(x_1, \ldots, x_n) = C_1 \land C_2 \land \ldots \land C_k \text{ of QSAT.} \]

- 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.
### 11.1 Load Balancing

#### 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!

---

#### Load Balancing

**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!

---

#### Load Balancing: List Scheduling

**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 $m$
  - $L_i \leftarrow 0$
  - $J(i) \leftarrow \emptyset$

- For $j = 1$ to $n$
  - $i = \arg\min_k L_k$
  - $J(i) \leftarrow J(i) \cup \{j\}$
  - $L_i \leftarrow L_i + t_j$

- 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 $C^* \leq \sum_j t_j$.
- **PF.** Some machine must process the most time-consuming job.

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

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

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

$$L_i - t_j \leq L_k$$

Lemma 1.

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, $L_\text{opt} \geq 2t_{m+1}$.

Proof.

1. Consider first $m+1$ jobs $t_1, \ldots, t_{m+1}$.
2. Since the $t_i$'s are in descending order, each takes at least $t_{m+1}$ time.
3. 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.

$$L_i = (L_i - t_j) + t_j \leq 2L_\text{opt}$$

Lemma 2.
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 $m$. 
