Chapter 9

PSPACE: A Class of Problems Beyond NP

9.1 PSPACE

PSPACE

- Decision problems solvable in polynomial space.

Observation. $P \subseteq PSPACE$.

Claim. $3$-SAT is in $PSPACE$.

Proof.

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

Theorem. $NP \subseteq PSPACE$.

Proof. Consider arbitrary problem $Y$ in $NP$.

1. Since $Y \subseteq 3$-SAT, there exists algorithm to solve $Y$ in poly-time.
2. Can implement black box in poly-space.
9.3 Quantified Satisfiability

**Theorem.** QSAT is in 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.

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

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

**Ex.** No. If Amy sets $x_1$ false; Bob sets $x_2$ false; Amy loses.

**Ex.** No. if Amy sets $x_1$ true; Bob sets $x_2$ true; Amy loses.

9.4 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. \( C_{ij}, 1 \leq i, j \leq 9 \)

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 \( O_i \) is \( (C_{11}, C_{22}, \ldots, C_{66}, C_{78}, C_{87}, C_{99}) \).
- 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!

Def.: \( C_{ij} \) means tile \( i \) is in square \( j \)

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

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 \( 2^n - 1 \).

Planning Problem: In Polynomial Space

Theorem. PLANNING is in PSPACE.

Pf.

- Suppose there is a path from \( c_0 \) to \( c* \) of length \( L \).
- Path from \( c_0 \) to midpoint and from midpoint to \( c* \) 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. P \subseteq \text{EXPTIME}.

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

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

It is known that P \neq \text{EXPTIME}, but whether which inclusion is strict is conjectured to be 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 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 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?

Yes if B \geq 20; no if B \geq 25.

Construction. Given instance \phi(x_1, \ldots, x_n) = C_1 \land C_2 \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.

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 prove that it is 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.
Chapter 11  
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.

- ρ-approximation algorithm.  
  - Guaranteed to run in poly-time.  
  - Guaranteed to solve arbitrary instance of the problem.  
  - Guaranteed to find solution within ratio ρ of true optimum.  
  
Challenge. Need to prove a solution’s value is close to optimum,  
without even knowing what optimum value is!

Approximation Algorithms

Load Balancing

Input: m identical machines; n jobs, job j has  
processing time tj.  
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 Li = ∑j∈J(i) tj.

Def. The makespan is the maximum load on any  
machine L = max i Li.

Load balancing. Assign each job to a machine to  
minimize makespan.

Load Balancing: List Scheduling

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, t1,t2, … ,tn) {  
for i = 1 to m {  
    Li ← 0  
    J(i) ← ∅  
}  
for j = 1 to n {  
    i = argmin k Lk  
    J(i) ← J(i) ∪ {j}  
    Li ← Li + tj  
}  
return J(1), …, 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

Load Balancing: List Scheduling Analysis

First worst-case analysis of an approximation algorithm.  
Need to compare resulting solution with optimal makespan L*.

Lemma 1. The optimal makespan C ≥ ∑j tj.  
PF. Some machine must process the most time-consuming  
job. -

Lemma 2. The optimal makespan C ≥ 1/2 ∑j tj.  
PF.  
The total processing time is ∑j tj.  
One of machines must load at least the fraction of  
total work. -

Implementation. O(n log m) using a priority queue.
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 \( i \), it had smallest load. Its load before assignment is \( L_i - t_j \) \( \leq L_k \) for all \( 1 \leq k \leq m \).

Now \( L_i = \sum (L_i - t_j) \leq L_k \) for all \( 1 \leq k \leq m \).

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

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

Load Balancing: LPT Rule

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 \( t_1, \ldots, t_{m+1} \).
- 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.

Proof. Some basic approach as for list scheduling
\[ t_i = (t_i - t_j) + t_j \leq 2t_j \]

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

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

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.