

PSPACE

Rangaballav Pradhan
ITER, SOADU

PSPACE

- **PSPACE** includes the set of Decision problems solvable in polynomial space.
- i.e., an algorithm that uses an amount of space that is polynomial in the size of the input.
- **Observation.** $P \subseteq \text{PSPACE}$.
- Space can be reused during a computation in ways that time, by definition, cannot.
- For example, an algorithm that just counts from 0 to $2^n - 1$ in base-2 notation. It simply needs to implement an n-bit counter. This algorithm runs for an exponential amount of time, and then halts; in the process, it has used only a polynomial amount of space.

PSPACE

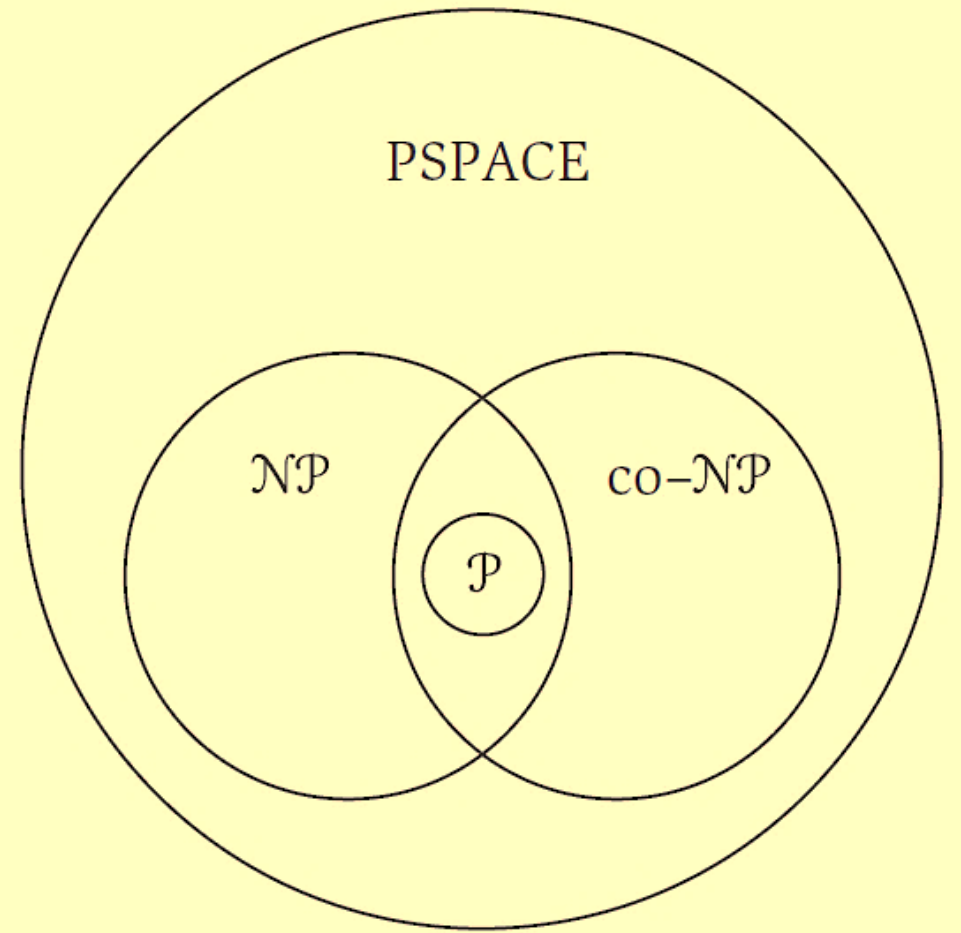
- **Claim.** 3-SAT \in PSPACE.
- **Pf.** We have n variables x_1, x_2, \dots, x_n .
- Enumerate all 2^n possible truth assignments using counter.
- Check each assignment to see if it satisfies all clauses.
- Every truth assignment will require exactly n bits at a time, then the same n bits can be reused for the next truth assignment and so on.
- So, the space requirement is polynomial. ▀

PSPACE

- **Theorem.** $\text{NP} \subseteq \text{PSPACE}$.
- **Pf.** Consider arbitrary problem $Y \in \text{NP}$.
- As $Y \leq_P \text{3-SAT}$, there is an algorithm that solves Y using a polynomial number of steps plus a polynomial number of calls to a black box for 3-SAT.
- Using the polynomial-space algorithm to implement this black box, we obtain an algorithm for Y that uses only polynomial space.

PSPACE

- A problem X is in PSPACE if and only if its complementary problem X' is in PSPACE as well.
- Thus we can conclude that $\text{co-NP} \subseteq \text{PSPACE}$.
- Given that PSPACE is an enormously large class of problems, containing both NP and co-NP, it is very likely that it contains problems that cannot be solved in polynomial time.



PSPACE-Complete

- A problem X is PSPACE-complete if,
 - (i) it belongs to PSPACE; and
 - (ii) for all problems Y in PSPACE, we have $Y \leq_P X$.
- PSPACE-Complete problems are the hardest problems in the class.

Planning Problem

- A PSPACE-Complete Problem
- Planning problems seek to capture, in a clean way, the task of interacting with a complex environment to achieve a desired set of goals.
- Example: Disaster management, factory setup, planning games such as 15-puzzle, Rubik's Cube, etc.
- There are a number of conditions to achieve and a set of allowable operators that we can apply to achieve these conditions.
- Thus we model the environment by a set $C = \{C_1, \dots, C_n\}$ of conditions: A given state of the world is specified by the subset of the conditions that currently hold. We interact with the environment through a set $\{O_1, \dots, O_k\}$ of operators.

Planning Problem

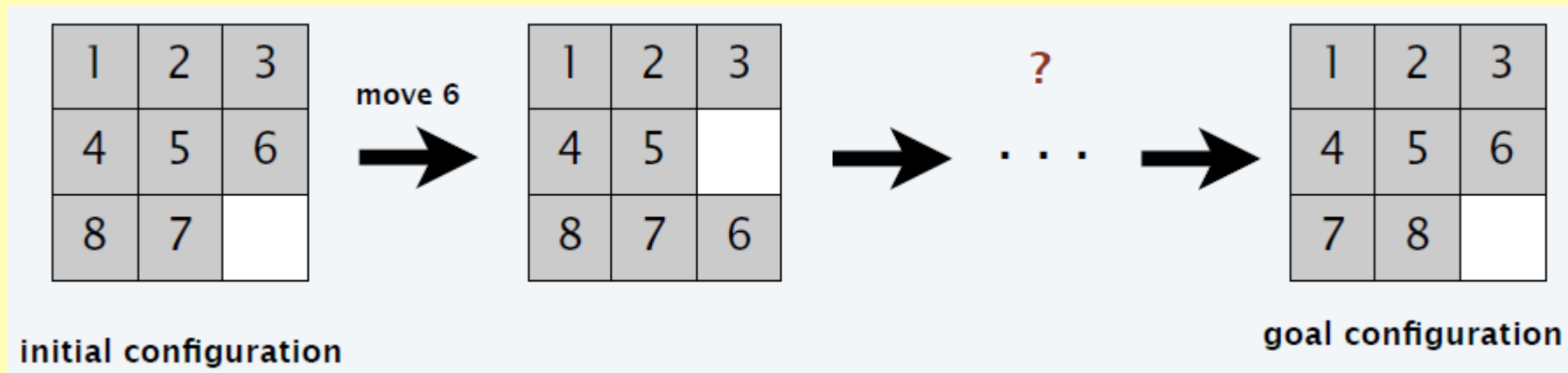
- Each operator O_i is specified by a **prerequisite list**, containing a set of conditions that must hold for O_i to be invoked; an **add list**, containing a set of conditions that will become true after O_i is invoked; and a **delete list**, containing a set of conditions that will cease to hold after O_i is invoked.
- For example, we could model the fifteen-puzzle by having a condition for each possible location of each tile, and an operator to move each tile between each pair of adjacent locations; the prerequisite for an operator is that its two locations contain the designated tile and the hole.

Planning Problem

- The Planning problem can be defined as follows: Given a set C_0 of initial conditions, and a set C^* of goal conditions, is it possible to apply a sequence of operators beginning with C_0 so that we reach a situation in which precisely the conditions in C^* (and no others) hold?

8-Puzzle

- 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.



8-Puzzle

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

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

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

Operators.

- Precondition to apply $O_i = \{C_{11}, C_{22}, \dots, C_{66}, C_{78}, C_{87}, C_{99}\}.$
- After invoking O_i , conditions C_{79} and C_{97} become *true*.
- After invoking O_i , conditions C_{78} and C_{99} become *false*.

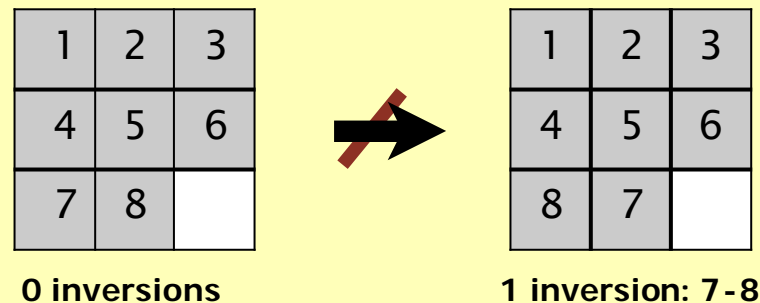
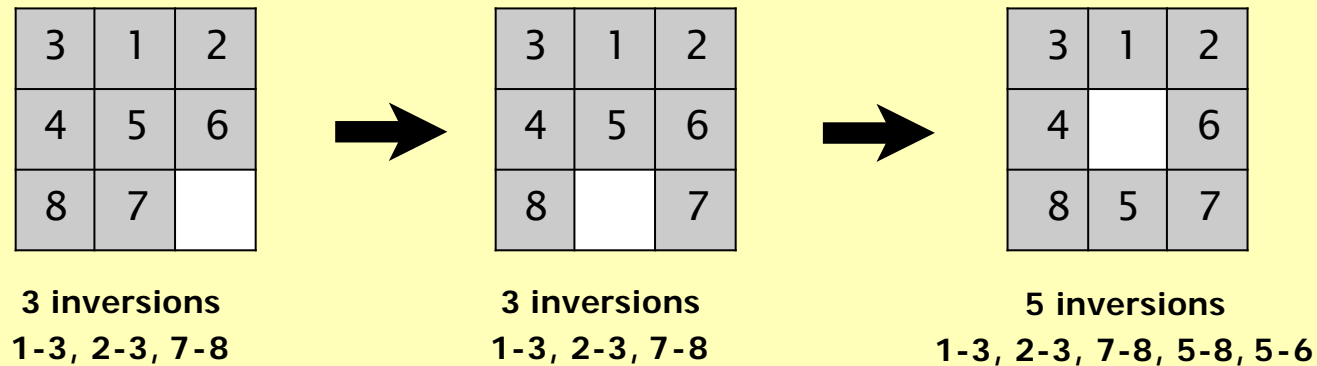
1	2	3
4	5	6
8	7	9



1	2	3
4	5	6
8	9	7

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).



Quantification

- In the 3-SAT problem, the goal is to determine whether a set of disjunctive clauses can be simultaneously satisfied.
- If we add quantifiers, the problem appears to become even more difficult.
- Let $\varphi(x_1, x_2, \dots, x_n)$ be a Boolean formula of the form $C_1 \wedge C_2 \wedge \dots \wedge C_k$ where each C_i is a disjunction of three terms. Suppose we ask,

$$\exists x_1 \forall x_2 \dots \exists x_{n-2} \forall x_{n-1} \exists x_n \Phi(x_1, \dots, x_n)?$$

- Here, we wish to know whether there is a choice for x_1 , so that for both choices of x_2 , there is a choice for x_3 , and so on, so that φ is satisfied. We refer to this decision problem as **Quantified 3-SAT** (or, **QSAT**).

Quantification

- The original 3-SAT problem, by way of comparison, simply asked

$$\exists x_1 \exists x_2 \cdots \exists x_{n-2} \exists x_{n-1} \exists x_n \Phi(x_1, \dots, x_n)?$$

- In 3-SAT it was sufficient to look for a single setting of the Boolean variables.
- Here's an example to illustrate the kind of reasoning that underlies an instance of QSAT. Suppose that we have the formula

$$\Phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

- And we ask,

$$\exists x_1 \forall x_2 \exists x_3 \Phi(x_1, x_2, x_3)?$$

Quantification

$$\Phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

- The answer to this question is yes: We can set x_1 so that for both choices of x_2 , there is a way to set x_3 so that φ is satisfied. Specifically, we can set $x_1 = 1$; then if x_2 is set to 1, we can set x_3 to 0 (satisfying all clauses); and if x_2 is set to 0, we can set x_3 to 1 (again satisfying all clauses).

Quantified 3-SAT (QSAT)

Theorem. Q-SAT \in PSPACE.

Brute-force approach. For the first quantifier $\exists x_1$, consider both possible values for x_1 in sequence. i.e., first set $x_1 = 0$ and check, recursively, whether the remaining portion of the formula evaluates to 1.

- Then set $x_1 = 1$ and check, recursively, whether the remaining portion of the formula evaluates to 1.
- The full formula evaluates to 1 if and only if *either* of these recursive calls yields a 1—that's simply the definition of the \exists quantifier.
- This is a divide-and-conquer algorithm, which, given an input with n variables, spawns two recursive calls on inputs with $n - 1$ variables each.
- If we save all the work done in both these recursive calls, our space usage $S(n)$ would satisfy the recurrence $S(n) \leq 2S(n - 1) + p(n)$, where $p(n)$ is a polynomial function. This would result in an exponential bound, which is too large.

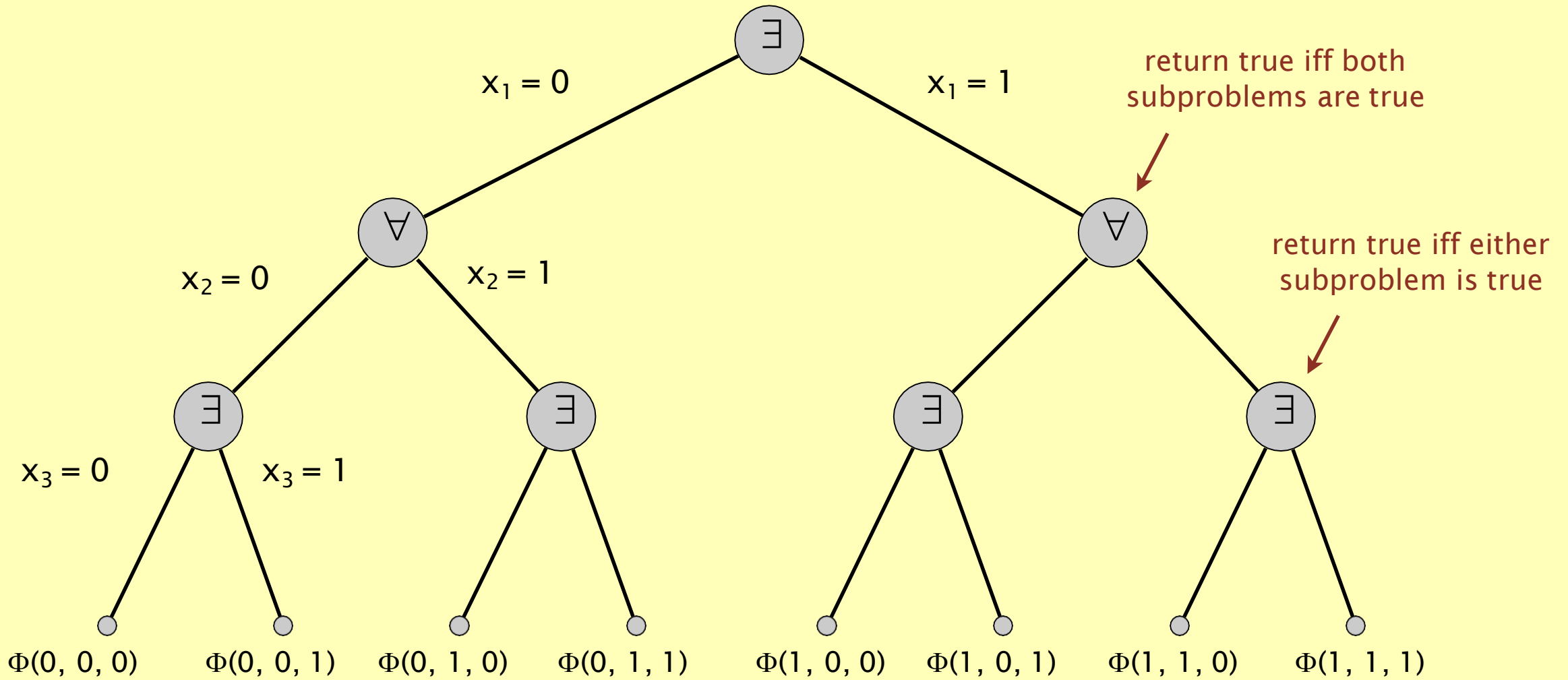
Quantified 3-SAT (QSAT)

Theorem. Q-SAT \in PSPACE.

Optimized approach. a simple optimization to the brute force approach greatly reduces the space usage.

- When we're done with the case $x_1 = 0$, all we really need to save is the single bit that represents the outcome of the recursive call; we can throw away all the other intermediate work.
- This is another example of “reuse”—we're reusing the space from the computation for $x_1 = 0$ in order to compute the case $x_1 = 1$.

Quantified 3-SAT (QSAT)



Quantified 3-SAT (QSAT)

If the first quantifier is $\exists x_i$ then

Set $x_i=0$ and recursively evaluate the quantified expression
over the remaining variables

Save the result (0 or 1) and delete all other intermediate work

Set $x_i=1$ and recursively evaluate the quantified expression
over the remaining variables

If either outcome yielded an evaluation of 1, then

return 1

Else return 0

Endif

If the first quantifier is $\forall x_i$ then

Set $x_i=0$ and recursively evaluate the quantified expression
over the remaining variables

Save the result (0 or 1) and delete all other intermediate work

Set $x_i=1$ and recursively evaluate the quantified expression
over the remaining variables

If both outcomes yielded an evaluation of 1, then

return 1

Else return 0

Endif

Endif

Quantified 3-SAT (QSAT)

Theorem. Q-SAT \in **PSPACE**.

Optimized approach. $S(n)$ would satisfy the recurrence $S(n) \leq S(n - 1) + p(n)$, where $p(n)$ is a polynomial function.

- Iterating this recurrence, we get,

$$S(n) \leq p(n) + p(n - 1) + p(n - 2) + \cdots + p(1) \leq n \cdot p(n) \quad \text{which}$$
is polynomial.

- So, Q-SAT \in **PSPACE**

Planning problem

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

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 $O = \{ O_1, \dots, O_k \}$.

- To invoke operator O_i , must satisfy certain prerequisite conditions.
- After invoking O_i certain conditions become true (Add list), and certain conditions become false (Delete list).

PLANNING. Is it possible to apply sequence of operators to get from initial configuration to goal configuration?

Planning problem: Solution

Modelling. An instance of Planning problem can be viewed as a giant, implicitly defined, directed graph G with a vertex for each of the 2^n possible configurations (i.e., each possible subset of C); and there is an edge of G from configuration C' to configuration C'' if, in one step, one of the operators can convert C' to C'' .

- In terms of this graph G , the Planning Problem has a very natural formulation:
 - Is there a path in G from C^0 to C^* ?
- Such a path corresponds precisely to a sequence of operators leading from C^0 to C^* .
- **Observation 1.** There are instances of the Planning Problem with n conditions and k operators for which there exists a solution, but the shortest solution has length $2^n - 1$.

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, \dots, C_n . \longleftarrow C_i corresponds to bit $i = 1$

Initial state. $c_0 = \varnothing$. \longleftarrow all 0s

Goal state. $c^* = \{C_1, \dots, C_n\}$. \longleftarrow all 1s

Operators. O_1, \dots, O_n .

- To invoke operator O_i , must satisfy C_1, \dots, C_{i-1} . \longleftarrow $i-1$ least significant bits are 1
- After invoking O_i , condition C_i becomes true. \longleftarrow set bit i to 1
- After invoking O_i , conditions C_1, \dots, C_{i-1} become false. \longleftarrow $i-1$ least significant bits are 0

Solution. $\{ \} \Rightarrow \{C_1\} \Rightarrow \{C_2\} \Rightarrow \{C_1, C_2\} \Rightarrow \{C_3\} \Rightarrow \{C_3, C_1\} \Rightarrow \dots$

Planning problem: binary counter

Observation 1. There are instances of the Planning Problem with n conditions and k operators for which there exists a solution, but the shortest solution has length $2^n - 1$.

- The graph G has 2^n nodes, and if there is a path from C^0 to C^* , then the shortest such path does not visit any node more than once. As a result, the shortest path between any pair of nodes in general has a length $= (|V|-1) = 2^n - 1$.

Observation 2. If a Planning instance with n conditions has a solution, then it has one using at most $2^n - 1$ steps.

- The graph G has 2^n nodes, and if there is a path from C^0 to C^* , then the shortest such path does not visit any node more than once. As a result, the shortest path can take at most $2^n - 1$ steps after leaving C^0 .

Planning problem: binary counter

Observation 2. If a Planning instance with n conditions has a solution, then it has one using at most $2^n - 1$ steps.

- We can prove this using induction on n .
- **Base.** If $n = 1$, the number of steps to reach C^1 is $2^1 - 1 = 1$ which is to apply O_1
- **I.H.** Assume that it is true for $n = k-1$. i.e., steps to achieve $C^{k-1} = 2^{k-1} - 1$
- **I.S.** Now, for $n=k$, we have the goal to achieve C^k , which can be set to 1 by applying O_k
- O_k can be applied for a prerequisite list $\{C_1, \dots, C_{k-1}\}$ which we achieved for $n = k-1$ in $2^{k-1} - 1$ steps.
- After applying O_k , C_k becomes 1 but all of $\{C_1, \dots, C_{k-1}\}$ are 0. To achieve C^k with all of the $\{C_1, \dots, C_k\}$ as 1, we need to apply another $2^{k-1} - 1$ steps (I.H.).
- So, Total steps $= 2^{k-1} - 1 + 1 + 2^{k-1} - 1 = 2^k - 1$

Planning problem \in EXPTIME

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 \in EXPTIME.

Pf. Run BFS/DFS 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

Theorem. PLANNING \in PSPACE.

Pf.

- Suppose there is a path from C_1 to C_2 of length L .
- Path from C_1 to midpoint (C') and from midpoint to C_2 are each of length $\leq L / 2$.
- Enumerate all possible midpoints C' .
- Apply recursively.
- Depth of recursion = $\log_2 L$. ▪

Planning problem

```
Path( $\mathcal{C}_1, \mathcal{C}_2, L$ )
  If  $L = 1$  then
    If there is an operator  $\mathcal{O}$  converting  $\mathcal{C}_1$  to  $\mathcal{C}_2$  then
      return 'yes'
    Else
      return 'no'
    Endif
  Else ( $L > 1$ )
    Enumerate all configurations  $\mathcal{C}'$  using an  $n$ -bit counter
    For each  $\mathcal{C}'$  do the following:
      Compute  $x = \text{Path}(\mathcal{C}_1, \mathcal{C}', \lceil L/2 \rceil)$ 
      Delete all intermediate work, saving only the return value  $x$ 
      Compute  $y = \text{Path}(\mathcal{C}', \mathcal{C}_2, \lceil L/2 \rceil)$ 
      Delete all intermediate work, saving only the return value  $y$ 
      If both  $x$  and  $y$  are equal to 'yes', then return 'yes'
    Endfor
    If 'yes' was not returned for any  $\mathcal{C}'$  then
      Return 'no'
    Endif
  Endif
```

Planning problem: Analysis

Claim. $\text{Path}(C_1, C_2, L)$ returns “yes” if and only if there is a sequence of operators of length at most L leading from C_1 to C_2 . Its space usage is polynomial in n , k , and $\log L$.

- Aside from the space spent inside recursive calls, each invocation of Path involves an amount of space polynomial in n , k , and $\log L$.
- But at any given point in time, only a single recursive call is active, and the intermediate work from all other recursive calls has been deleted. Thus, for a polynomial function p , the space requirement $S(n, k, L)$ satisfies the recurrence

$$S(n, k, L) \leq p(n, k, \log L) + S(n, k, \lceil L/2 \rceil)$$

$$S(n, k, 1) \leq p(n, k, 1).$$

- If we put $L = 2^n$, We have $s(n, k, L) = p(n, k)$.

PSPACE-complete

PSPACE. Decision problems solvable in polynomial space.

PSPACE-complete. Problem $Y \in \mathbf{PSPACE}$ -complete if


- (i) $Y \in \mathbf{PSPACE}$ and
- (ii) for every problem $X \in \mathbf{PSPACE}$, $X \leq_p Y$.

Theorem. [Stockmeyer–Meyer 1973] $\text{QSAT} \in \mathbf{PSPACE}$ -complete.

Theorem. $\mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$.

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

Summary. $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$.



The diagram shows four complexity classes arranged horizontally: P, NP, PSPACE, and EXPTIME. Below each class is a red upward-pointing arrow. A red line connects the first arrow (under P) to the second arrow (under NP). Another red line connects the second arrow (under NP) to the third arrow (under PSPACE). A third red line connects the third arrow (under PSPACE) to the fourth arrow (under EXPTIME). This visualizes the chain of inclusions: P ⊆ NP ⊆ PSPACE ⊆ EXPTIME.

it is known that $\mathbf{P} \neq \mathbf{EXPTIME}$, but unknown
which inclusion is strict; conjectured that all are

PSPACE-complete problems

More PSPACE-complete problems.

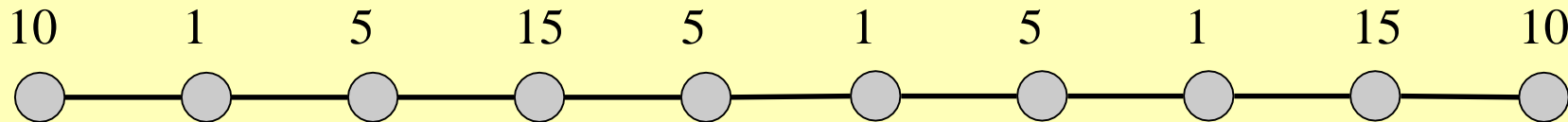
- Competitive facility location.
- Natural generalizations of games.
 - Othello, Hex, Geography, Rush-Hour, Instant Insanity
 - Shanghai, go-moku, Sokoban
- Given a memory restricted Turing machine, does it terminate in at most k steps?
- Do two regular expressions describe different languages?
- Is it possible to move and rotate 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 $G = (V, E)$ 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-LOCATION \in **PSPACE**- complete.

Pf.

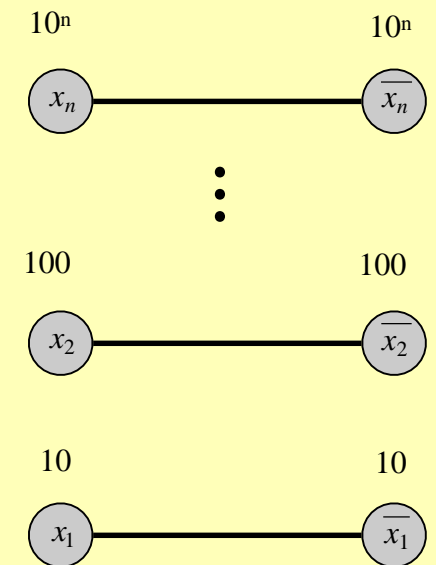
- To solve in poly-space, use recursion like Q-SAT, but at each step there are up to n choices instead of 2.
- To show that it's complete, we show that Q-SAT polynomial reduces to it.
- Given an instance of Q-SAT, we construct an instance of COMPETITIVE- FACILITY-LOCATION so that player 2 can force a win iff Q-SAT formula is *true*.

Competitive facility location

Construction. Given instance $\Phi(x_1, \dots, x_n) = C_1 \wedge C_1 \wedge \dots C_k$ of Q-SAT.

← assume n is odd

- 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 $B = c^{n-1} + c^{n-3} + \dots + c^4 + c^2 + 1$.
- (ensures variables are selected in order x_n, x_{n-1}, \dots, x_1)
- As is, player 2 will lose by 1 unit: $c^{n-1} + c^{n-3} + \dots + c^4 + c^2$.



Competitive facility location

Construction. Given instance $\Phi(x_1, \dots, x_n) = C_1 \wedge C_1 \wedge \dots C_k$ of Q-SAT.

- Give player 2 one last move on which he/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. ■

