CS 580
ALGORITHM DESIGN AND ANALYSIS

Beyond NP

Vassilis Zikas

SUMMARY

- Class so far:
  - Efficient algorithms
  - Probably hard problems (NP-complete)
- Today:
  - A glimpse beyond: PSPACE

PSPACE

- So far, we've been thinking of *time* as the valuable resource, so we focused on polynomial time algorithms
- P: set of problems solvable in polynomial time
- This lecture: treat *space* as the fundamental resource

PSPACE

- PSPACE: set of problems solvable in polynomial space

- Observation: $P \subseteq PSPACE$
    - Proof: a polynomial time algorithm can "consume" at most a polynomial amount of space

PSPACE

- What can we do with polynomial space?
- Different mentality than “time”
- Count from 0 to 2^n-1
  - Only needs n bits!
- Not super exciting as an algorithm, but highlights that space can be re-used!
  - Time, of course, cannot

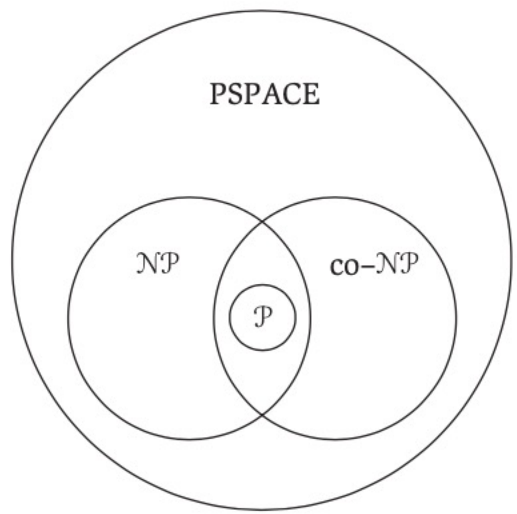
PSPACE

- Claim: 3-SAT is in PSPACE
- Proof:
  - Enumerate all possible assignments
  - For each assignment check if it satisfies all clauses

- Theorem: $NP \subseteq PSPACE$
- Proof:
  - Consider an arbitrary problem $X \in NP$
  - $X \leq_P$ 3-SAT, by definition, so there is a poly-time algorithm that solves X by calling an oracle for 3-SAT
  - This algorithm (and oracle) can be implemented in polynomial space

PSPACE

- PSPACE is closed under complements
- Therefore, co-NP ⊆ PSPACE



HARD PROBLEMS IN PSPACE: QUANTIFICATION

- QSAT: Let ϕ be a 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

- 3-SAT is just

$$\exists x_1 \exists x_2, \ldots, \exists x_n \; \phi(x_1, \ldots, x_n)$$

HARD PROBLEMS IN PSPACE: QUANTIFICATION

- QSAT: Let ϕ be a CNF formula. Is the following propositional formula true?

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

- Intuition: Alice picks value for x_1, the Bob picks value for x_2, the Alice picks x_3, and so on. Can Alice satisfy ϕ no matter what Bob does?

HARD PROBLEMS IN PSPACE: QUANTIFICATION

- QSAT: Let ϕ be a CNF formula. Is the following propositional formula true?

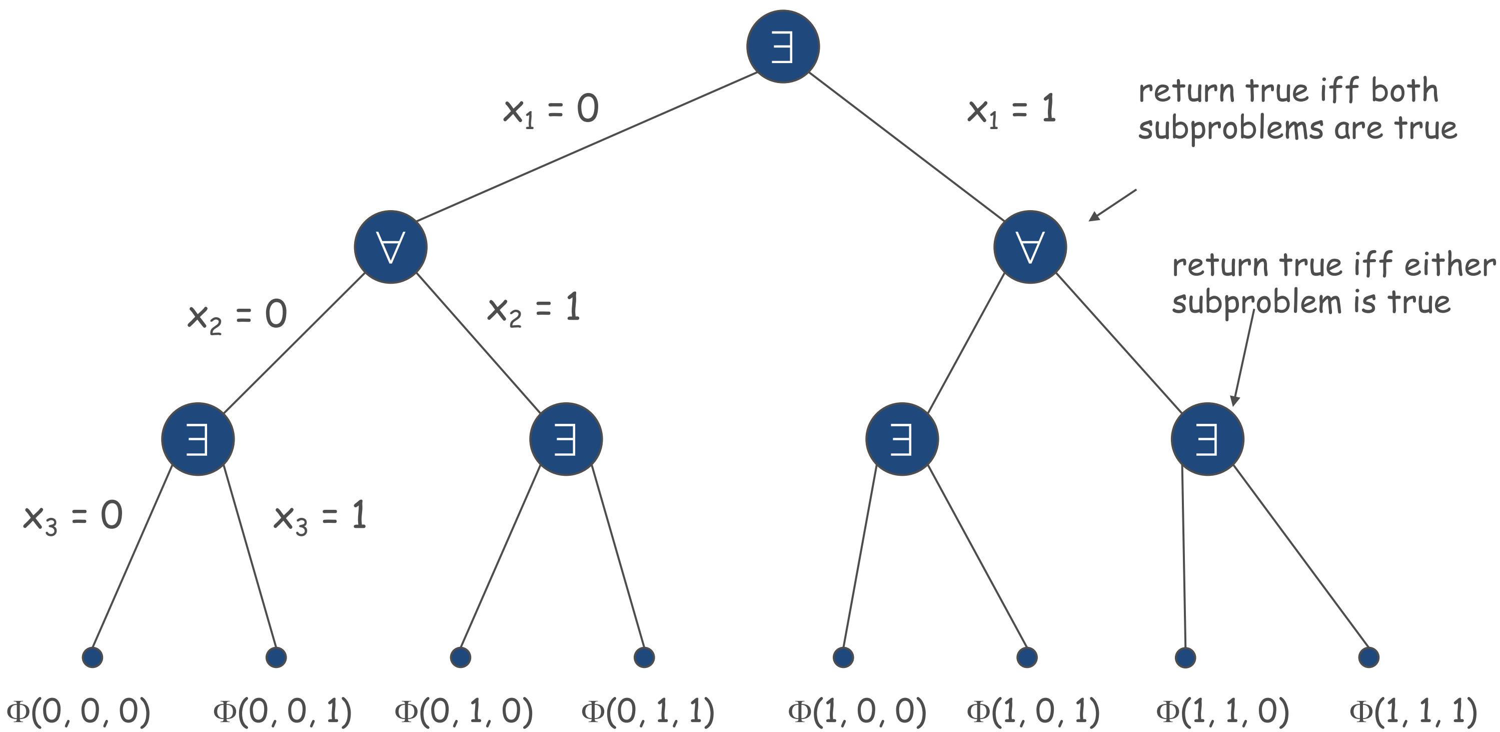
$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots \forall x_{n-1} \exists x_n \; \phi(x_1, \dots, x_n)$$

- $(x_1 \vee x_2) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$
- Yes: Alice sets $x_1 = 1$. Bob sets x_2. Alice sets $x_3 = x_2$ and all clauses are satisfied

- $(x_1 \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$
- No:
  - If Alice sets $x_1 = 1$, Bob could set $x_2 = 1$. Alice loses.
  - If Alice sets $x_1 = 0$, Bob could set $x_2 = 0$. Alice loses

HARD PROBLEMS IN PSPACE: QUANTIFICATION

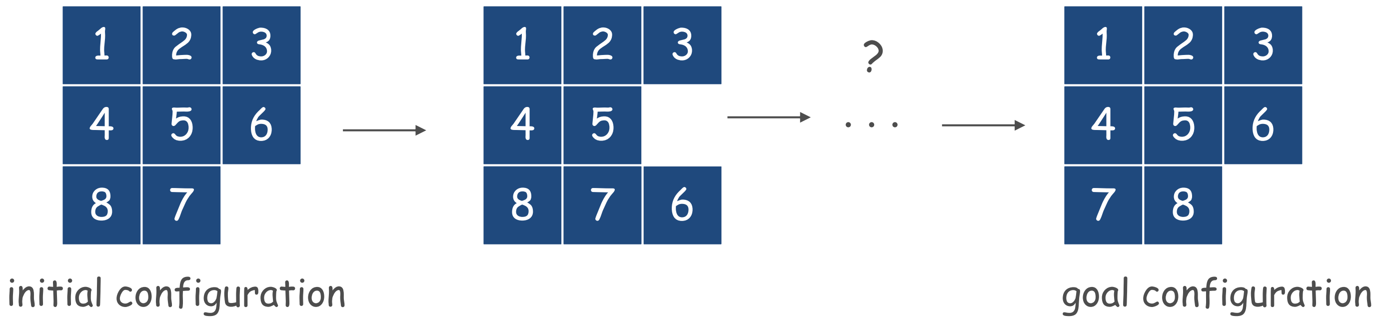
- Theorem: $QSAT \in PSPACE$
- Proof
  - Recursively try all possibilities
  - Only need one bit of information from each sub-problem
  - Amount of space proportional to the call stack



PLANNING PROBLEM

8-PUZZLE

- 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.

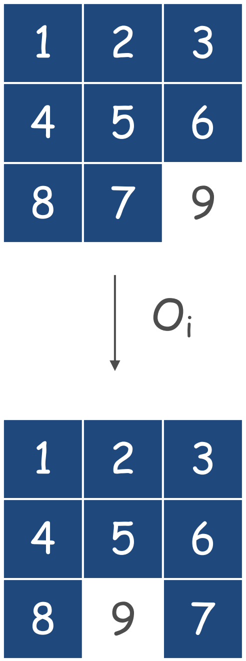


PLANNING PROBLEM

- Input:
  - 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 we must satisfy some prereq conditions
    - After we invoke operator O_i some conditions become true and other become false
- Question: Is it possible to apply sequence of operators to get from initial configuration to goal configuration?

PLANNING PROBLEM

- Example: 8 – puzzle
- Conditions: $C_{i,j}$: $1 \leq i, j \leq 9$
  - $C_{i,j}$ means that tile i is in square j
- Initial state: $c_0 = \{C_{1,1}, C_{2,2}, \dots, C_{7,8}, C_{8,7}, C_{9,9}\}$
- Goal state: $c^* = \{C_{1,1}, \dots, C_{9,9}\}$
- Operators:
  - Conditions to apply $O_i = \{\mathrm{C}_{1,1}, \dots, \mathrm{C}_{7,8}, \mathrm{C}_{8,7}, \mathrm{C}_{9,9}\}$
  - After invoking O_i, conditions $C_{7,9}$ and $C_{9,8}$ become true
  - After invoking O_i, conditions $C_{7,8}$ and $C_{9,9}$ become false



AN ASIDE

- How to check if an 8-puzzle is solvable: Any legal move preserves the parity of the number of inversions: pairs of numbers in the wrong order

3	1	2
4	5	6
8	7	

3 inversions
1-3, 2-3, 7-8

$\longrightarrow$

3	1	2
4	5	6
8		7

3 inversions
1-3, 2-3, 7-8

$\longrightarrow$

3	1	2
4		6
8	5	7

5 inversions
1-3, 2-3, 7-8, 5-8, 5-6

1	2	3
4	5	6
7	8	

0 inversions

$\not\longrightarrow$

1	2	3
4	5	6
8	7	

1 inversion: 7-8

- Not really an algorithm, but still pretty cool

PLANNING PROBLEM

- Think of planning as a very very large graph
- There is a node for each of the 2^n possible configurations (T/F values for conditions)
- There is a directed edge from configuration c to configuration c', if one of the operators can take us from c to c'
- Question: is there a directed path from c_0 to c^*?

PLANNING PROBLEM

- The configuration graph can have 2^n nodes and the shortest path can be of length $2^n - 1$

- Concrete example:
  - $c_0 = (0,0,0, \dots, 0)$, i.e. all conditions 0
  - $c^* = (1,1,1, \dots, 1)$
  - To invoke O_i must satisfy $C_1, \dots, C_{i-1}$
  - After invoking, C_i becomes true and $C_1, \dots, C_{i-1}$ become false
  - Solution: $\{\} \rightarrow^{O_1} \{C_1\} \rightarrow^{O_2} \{C_2\} \rightarrow^{O_1} \{C_1, C_2\} \dots$
  - Number of steps: $2^n - 1$

PLANNING PROBLEM

- Claim: PLANNING is in EXPTIME
- Proof:
  - Run BFS on the configuration graph

PLANNING PROBLEM

- Theorem: PLANNING is in PSPACE
- Proof:
  - Intuition: BFS and DFS are not aggressive enough in terms of space re-usage
    - Need to go around this
  - 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 of length $\leq L/2$
  - Enumerate all possible midpoints!
  - Recurse. Depth of recursion $\log_2 L$

PLANNING PROBLEM

enumerate using binary counter

```
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
}
```

PSPACE-COMPLETENESS (9.5 IN KT)

PSPACE COMPLETE

- Definition: A problem X is PSPACE-hard if for every problem Y in PSPACE $Y \leq_P X$
- Definition: A problem X is PSPACE-complete if (1) X is in PSPACE, (2) X is PSPACE-hard.

- Theorem [Stockmeyer-Meyer 1973]: QSAT is PSPACE-complete
- Corollary: PSPACE $\subseteq$ EXPTIME
  - Proof: We gave an exponential time algorithm for QSAT, and QSAT is PSPACE-complete.

COMPLEXITY

- What we know:

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$$

and

$$P \subset EXPTIME$$

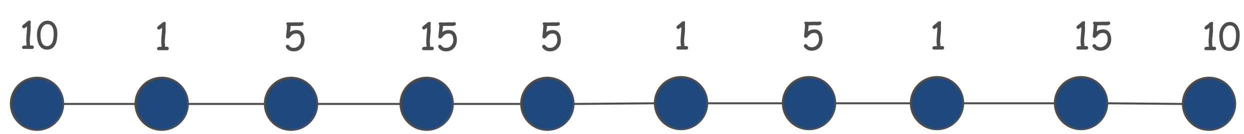
- What we don't know:
  - Which of the above inclusions are strict?
  - We think all of them
  - We know it should be at least one of them
  - We have proofs for zero of them

MORE PSPACE-COMPLETE PROBLEMS

- Competitive facility location
- Generalizations of games
  - Othello, Hex, Shanghai, Sokoban, etc
  - Note: generalized chess and go and EXPTIME-complete
- Given a memory restricted Turing Machine does it terminate in at most k steps?
- Do two regular expressions describe different languages?
- Is a deadlock state possible within a system of communicating processors?

COMPETITIVE FACILITY LOCATION

- Input: Graph with positive node weights and a target B
- Game: Two competing players alternate in selecting nodes. Not allowed to select a node if any of its neighbors has been selected.
- Question: Can the second player guarantee at least B units of profit?



Yes if B = 20; no if B = 25.

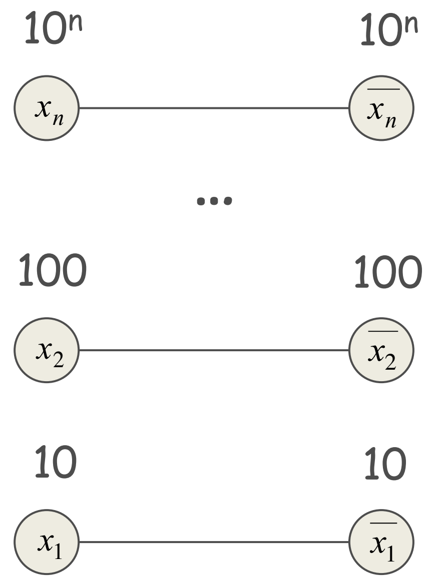
COMPETITIVE FACILITY LOCATION

- Claim: COMPETITIVE FACILITY LOCATION is PSPACE-complete
- Proof:
    - To show that it is in PSPACE, we can use almost the same algorithm as QSAT
        - Recursion step has n choices instead of 2
    - To show that it is PSPACE-hard, we reduce QSAT to it. I.e. given an instance of QSAT we construct a game such that player 2 can get utility B (win) iff the QSAT formula is false

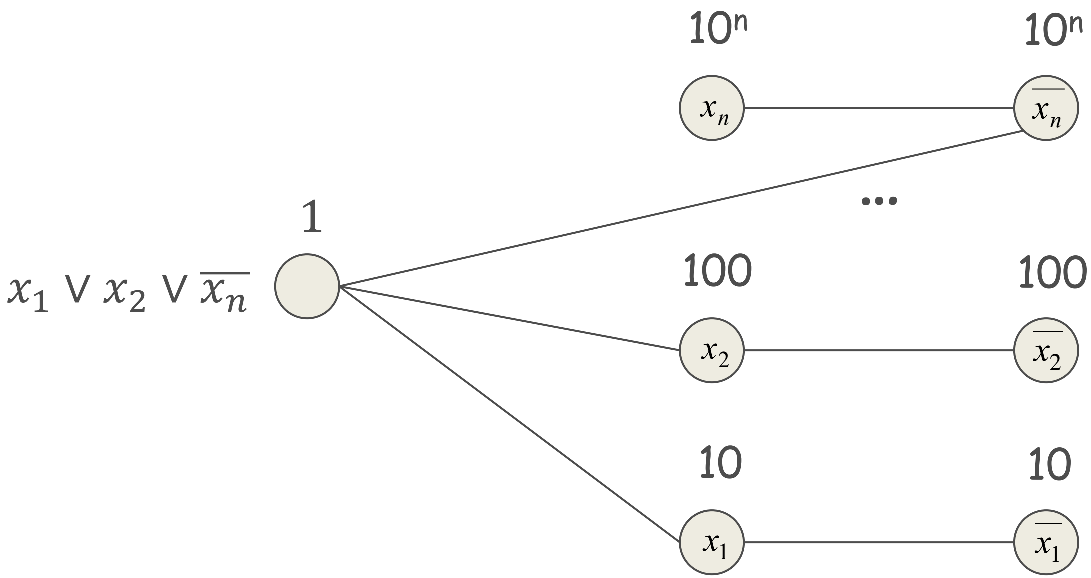
COMPETITIVE FACILITY LOCATION

- Construction:
  - We are given a formula $\phi(x_1, \dots, x_n) = C_1 \wedge \cdots \wedge C_k$ of QSAT
  - Include a node for each literal and its negation, and connect them
    - At most one of x_i and $\overline{x_i}$ can be selected
  - Choose $c \geq k + 2$ and put weight c^i on literal x_i and its negation
  - Set $B = c^{n-1} + c^{n-3} + \cdots + c^4 + c^2 + 1$
    - Ensures variables are selected in order $x_n, x_{n-1}, \dots$
  - So far, player 2 will lose by 1
    - The max they can get is $c^{n-1} + c^{n-3} + \cdots + c^2$

COMPETITIVE FACILITY LOCATION



COMPETITIVE FACILITY LOCATION



- Give player 2 one more chance!
- Add a node with weight 1 for each clause C_j (and connect with the corresponding literal nodes)
- Player 2 can take this extra move if and only if there exists some clause that is left unsatisfied

SUMMARY

- A glimpse beyond NP