Gentle Introduction to Computational Complexity

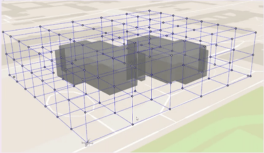
Francois Schwarzentruber

ENS Rennes, France

April 4, 2019

Motivation: naive methodology

Define a problem



e.g. coverage by a connected multi-drone system

→

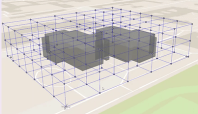
Directly design an algo

Issues

- you may design an overly complicated algorithm...
  ...whereas the problem is intrinsically simpler
- you may try to design a simple algorithm...
  ...whereas the problem is intrinsically harder;

Motivation: computational complexity comes in

Define a problem

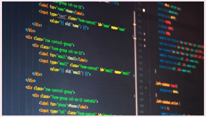


Study its intrinsic complexity

Theorem (Charrier et al., 2017)

The coverage by a connected multi-drone system is PSPACE*-complete.*

Design an algo



We learned:

- No polynomial-time algorithm exists, unless $\mathrm{P} = \mathrm{PSPACE}$;
- Algorithmic techniques corresponding to PSPACE are suitable.

Motivation: algorithmic techniques

Complexity class	Algorithmic techniques
P	Greedy algorithm, dynamic programming, algorithms on graphs
NP	Backtracking, Backjumping, Branch-and-bound, SAT, SAT modulo theories, Linear programming
PSPACE	Model checkers, planners, Monte-Carlo tree search, QBF solvers

Outline

1. Complexity classes defined with deterministic algorithms
   - Problems
   - Definition of an algorithm
   - Time and space (memory) as resources
   - Definition of complexity classes
2. Abstracting the combinatorics
3. Proving hardness
4. PSPACE
5. Big theorems in computational complexity

Outline

Example: traveler salesman problem (TSP)

Definition (Problem)

Defined in terms of input/output.

Example

TSP

- input: a weighted graph $G = (V, E, w)$;

  G described by an adjacency list and weights are written in binary

- output: a minimal tour;

Outline

Algorithm

Definition (Deterministic algorithm)

Standard algorithm, but no random choices.

Example

function tsp(G)
$\quad$ $bestWeight := +\infty$
$\quad$ $bestTour = -$
$\quad$ **for** all tours t **do**
$\quad\quad$ **if** $weight(t) < bestWeight$ **then**
$\quad\quad\quad$ $bestWeight := weight(t)$
$\quad\quad\quad$ $bestTour := t$
$\quad$ **return** $bestTour$

Outline

1. Complexity classes defined with deterministic algorithms
   - Problems
   - Definition of an algorithm
   - Time and space (memory) as resources
   - Definition of complexity classes
2. Abstracting the combinatorics
3. Proving hardness
4. PSPACE
5. Big theorems in computational complexity

Cobham's thesis

Cobham's thesis

Polynomial in the size n of the input = efficient.

Reasons:

- Many real algorithms are $O(n^3)$ at most;
- *algo* efficient $\Rightarrow$ **for** $i := 1..n$ **do** *algo* efficient;
- Do not depend so much on a computation model.

Very liberal concerning the complexity analysis

$O(\log n)$	logarithmic
$O(poly(n))$	polynomial
$2^{O(poly(n))}$	exponential
$2^{2^{O(poly(n))}}$	double-exponential

Membership in EXPTIME

Proposition

TSP *is in* EXPTIME.

Proof.

- tsp solves TSP.
- tsp runs in $2^{poly(|G|)}$.

```
function tsp(G)
    bestWeight := +∞
    bestTour = −
    for all tours t do
        if weight(t) < bestWeight then
            bestWeight := weight(t)
            bestTour := t
    return bestTour
```

□

Outline

Definition of complexity classes

Definition

EXPTIME is the class of problems for which *there is* an algorithm that solves it in exponential-time.

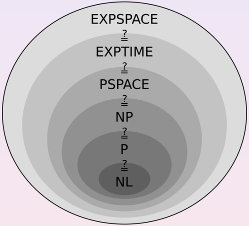
	LOGSPACE
P	PSPACE
EXPTIME	EXPSPACE
2EXPTIME	2EXPSPACE

Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
   - Decision problems
   - One-player algo
   - Definition of NP
3. Proving hardness
4. PSPACE
5. Big theorems in computational complexity

New fine-grained complexity classes

TSP is in EXPTIME...



Methodology

Define new fine-grained complexity classes by means of *games*.

Outline

Decision problems

Definition (decision problem)

Problem whose output is yes/no.

Example ()

TSP

- input: a weighted graph $G = (V, E, w)$;

  G described by an adjacency list and weights are written in binary

- output: a minimal tour.

TSP reformulated as a decision problem

Example

TSP

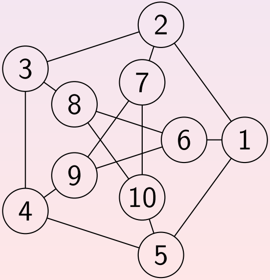
- input:
  - a weighted graph $G = (V, E, w)$;
  - a threshold $c \in \mathbb{N}$;
- output:
  - yes, if there is a tour in G of weight $\leq c$;
  - no otherwise.

Examples

Example

GRAPH COLORING

- Input: an undirected graph $G = (V, E)$;
- Output: yes if there is a coloring of vertices using ●●●, than assigns different colors to adjacent vertices; no otherwise.

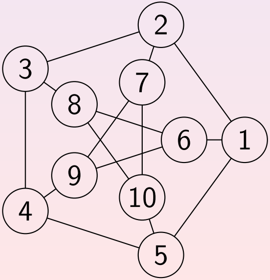


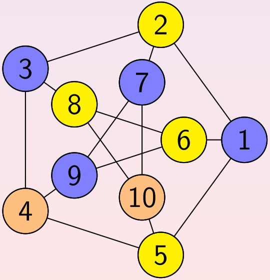
Examples

Example

GRAPH COLORING

- Input: an undirected graph $G = (V, E)$;
- Output: yes if there is a coloring of vertices using ●●●, than assigns different colors to adjacent vertices; no otherwise.





Examples

Example

SAT

- Input: a Boolean formula φ;
- Output: yes if there are values for Boolean variables that make φ true; no otherwise.

$$(p \vee q) \wedge (r \rightarrow \neg p) \wedge r \wedge (r \rightarrow \neg s) \wedge (s \rightarrow \neg q)$$

Examples

Example

HALT

- input: a program π;
- output:
  - yes, if the execution of π halts;
  - no otherwise.

Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
   - Decision problems
   - One-player algo
   - Definition of NP
3. Proving hardness
4. PSPACE
5. Big theorems in computational complexity

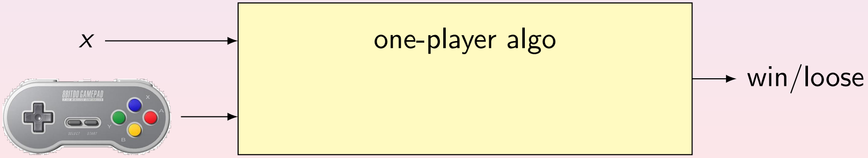
One-player algo

Definition (One-player algo)

A *one-player algo* is an algorithm that may use special instructions

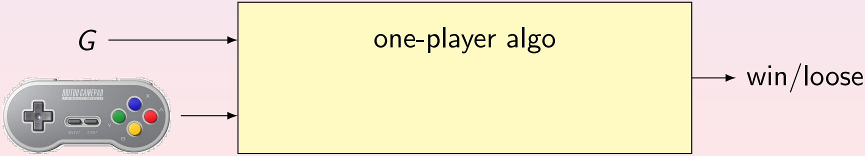
$$\textbf{choose } b \in \{0, 1\}$$

and that ends with instruction **win** or instruction **loose**.



One-player algo for graph coloring

one-player-algo graphColoring (G)
  for all vertices v of G **do**
    choose a color among for v
  if no adjacent vertices have the same color **then**
    win
  else
    loose



One-player game for graph coloring

```
one-player-algo graphColoring (G)
    for all vertices v of G do
        choose a color among ●●● for v
    if no adjacent vertices have the same color then
        win
    else
        loose
```

Proposition

G is a positive instance of GRAPH COLORING
iff
the player has a winning strategy in graphColoring (G)

$\rightarrow$ we say that graphColoring decides GRAPH COLORING.

One-player game

```
one-player-algo tsp(G, c)
    t := empty tour
    for i = 1..nb vertices in G do
        choose a non-already chosen successor for t
        extend t with that successor
    if t is a tour and weight(t) ≤ c then win else loose
```

Proposition

(G, c) is a positive instance of TSP
iff
the player has a winning strategy at the game tsp(G, c)

$\rightarrow$ we say that tsp decides TSP.

Outline

Execution-time of a one-player algo

Definition (poly-time one-player algo)

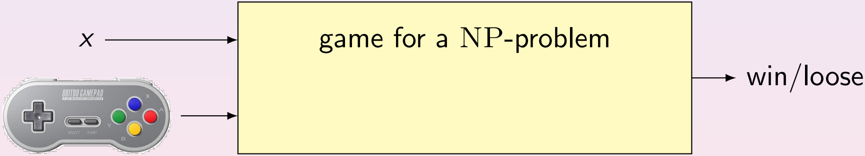
A one-player algo $algo(x)$ is in polynomial-time in $|x|$ if the length of all runs is $poly(|x|)$.

```
one-player-algo graphColoring (G)
    for all vertices v of G do
        choose a color among ●●● for v
    if no adjacent vertices have the same color then
        win
    else
        loose
```

Definition of NP

Definition

NP = class of decision problems such that there is a one-player game that decides it in polynomial-time.



Example (of decision problems that are in NP)

TSP Graph coloring SAT Shortest path (is in P)
Real linear programming (is in P) Clique in graphs
Integer linear programming

Terminology

- one player game = non-deterministic algorithm
- N = non-deterministic;
- choice of a move = a non-deterministic choice/guess
- the list of moves = a certificate

	LOGSPACE		NLOGSPACE
P	PSPACE	NP	NPSPACE
EXPTIME	EXPSPACE	NEXPTIME	NEXPSPACE
2EXPTIME	2EXPSPACE	NEXPTIME	NEXPSPACE

Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
3. Proving hardness
    - Easier than
    - Intrinsic hardness
    - NP-complete problems
4. PSPACE
5. Big theorems in computational complexity

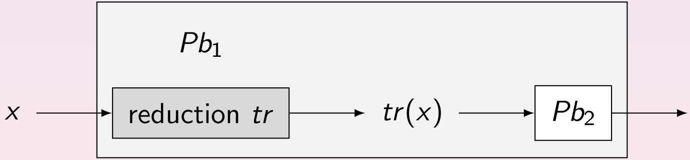
Outline

Easier than

Definition

Problem Pb_1 is *easier than* problem Pb_2 if there is a poly-time deterministic algorithm tr such that:

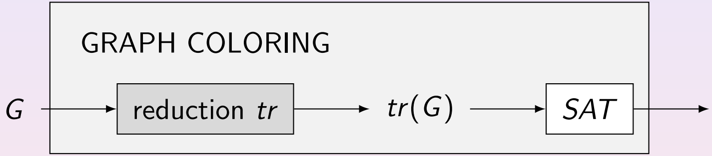
x is positive instance of Pb_1 iff $tr(x)$ is a positive instance of Pb_2



Terminology

- *Pb_1 reduces to Pb_2 in poly-time*
- *tr is called a poly-time reduction from Pb_1 to Pb_2*

Example: Graph coloring is easier than SAT



$tr(G) :=$ a Boolean formula expressing that G is colorable.

`http://people.irisa.fr/Francois.Schwarzentruber/reductioncatalog/`

Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
3. Proving hardness
   - Easier than
   - Intrinsic hardness
   - NP-complete problems
4. PSPACE
5. Big theorems in computational complexity

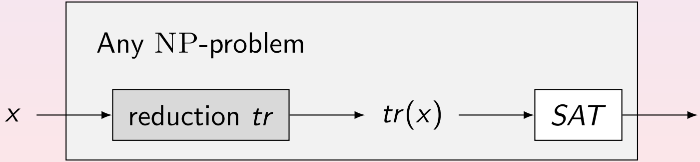
SAT is NP-hard

Terminology

A problem is NP-hard if any NP-problem is easier than it.

Theorem

Cook's theorem SAT is NP-hard.



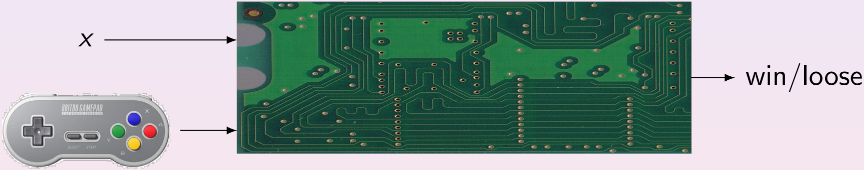
Any NP-problem is easier than SAT

x

one-player poly-time algo for that NP-problem

win/loose

Any NP-problem is easier than SAT



$tr(x) :=$ Boolean formula saying ‘the game run on x is winning’

NP-hard problems

Theorem

$$\left.\begin{array}{l} \bullet\ \ Pb_1 \textit{ is } \mathrm{NP}\textit{-hard} \\ \bullet\ \ Pb_1 \textit{ easier than } Pb_2 \end{array}\right\} \textit{implies } Pb_2 \textit{ is } \mathrm{NP}\textit{-hard.}$$

$$\begin{tikzcd}
 & & \text{HAMILTONIAN CYCLE} \arrow[r, dashed, "\text{easier than}"] & \text{TSP} \\
\text{any } \mathrm{NP}\text{-problem} \arrow[r, dashed, "\text{easier than}"] & \text{SAT} \arrow[r, dashed, "\text{easier than}"] & \text{3SAT} \arrow[u, dashed, "\text{easier than}"'] \arrow[d, dashed, "\text{easier than}"'] & \\
 & & \text{COLORING} &
\end{tikzcd}$$

Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
3. Proving hardness
   - Easier than
   - Intrinsic hardness
   - NP-complete problems
4. PSPACE
5. Big theorems in computational complexity

NP-complete problems

Definition

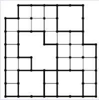
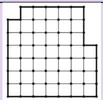
A problem is NP-complete if:

- it is in NP;
- it is NP-hard.

$$
\begin{tikzcd}
 & \text{HAMILTONIAN CYCLE} \arrow[r, dashed, "\text{easier than}"] & \text{TSP} \\
\text{SAT} \arrow[r, dashed, "\text{easier than}"] & \text{3SAT} \arrow[u, dashed, "\text{easier than}"'] \arrow[d, dashed, "\text{easier than}"'] & \\
 & \text{COLORING} &
\end{tikzcd}
$$

Understand where the 'complexity' is

Complexity of TSP restrictions

NP-complete	even for 2D grid graphs [Itai, Papadimitriou, Szwarcfiter, 1982]	
in P	for solid grid graphs [Arkin, Bender, Demaine, Fekete, Mitchell, Sethia, 2001]	

Parameterized complexity

identify a parameter (e.g. diameter of graphs, etc.) on inputs that sums up the complexity.

Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
3. Proving hardness
4. PSPACE
   - Two-player poly-time games
   - One-player poly-space games
   - PSPACE-complete problems
5. Big theorems in computational complexity

Outline

Motivation

- Bounded planning versus the environment
- querying $(\exists\forall)^*$-properties (e.g. SQL, etc.)

$$\exists x, \forall y, (R(x,y) \rightarrow \exists z, p(f(x,y,z))$$

$\rightarrow$ winning strategy for player one in a two-player poly-time game

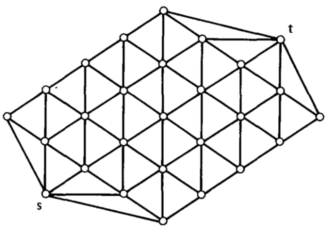
[Arora, Barak, chap. 4.2.2 (≪ The essence of PSPACE: optimum strategies for game-playing ≫)]

Generalized Hex

Example (Even, Tarjan, 1976)

HEX

- Input: A graph G, a source s, a target t;
- Output: yes if player 1 has a winning strategy to the game:
  - by turn, player i select a non-selected vertex in $G \setminus \{s, t\}$;
  - player 1 wins if there is a $s - t$-path made up of 1-vertices.



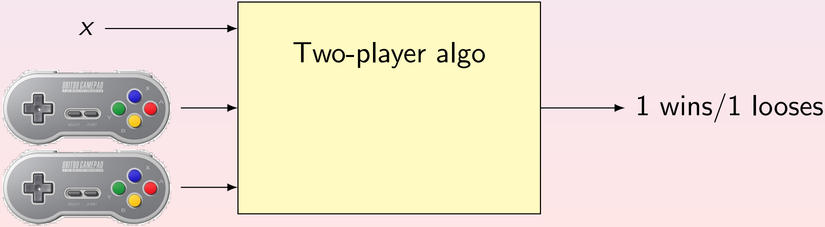
Two-player algo

Definition (Two-player algo)

A *two-player algo* is an algorithm that may use special instructions

$$\textbf{player one chooses } b \in \{0,1\}$$
$$\textbf{player two chooses } b \in \{0,1\}$$

and that ends with **player one wins** or **player one looses**.



Two-player poly-time algos

two-player-algo hex(G, s, t)
  while some non-selected vertex in G **do**
    player one chooses a non-selected vertex in $G \setminus \{s, t\}$
    player two chooses a non-selected vertex in $G \setminus \{s, t\}$
    if there is a $s - t$-path made up of 1-vertices **then**
      player one wins
  player one looses

Definition

- A strategy for a player tells her/him which move to take at each time.
- A winning strategy for player one makes player one win, whatever the moves of the other player.

A two-player algo decides a decision problem

Proposition

(G, s, t) is a positive instance of HEX
iff
the first player has a winning strategy in hex(G, s, t)

$\rightarrow$ We say that hex decides HEX!

Alternating poly-time: AP

Definition

AP is the class of decision problems such that there is a two-player algo that decides it in poly-time.

‘

Theorem

HEX in AP*.*

Proof.

hex decides HEX in poly-time. □

Quantified binary formulas

Example

QBF-SAT

- Input: a closed quantified binary formula φ;
- Output: yes if φ is true; no otherwise.

$$\exists p, \forall q, \forall r, \exists s (p \to (q \wedge r \to s))$$

Theorem

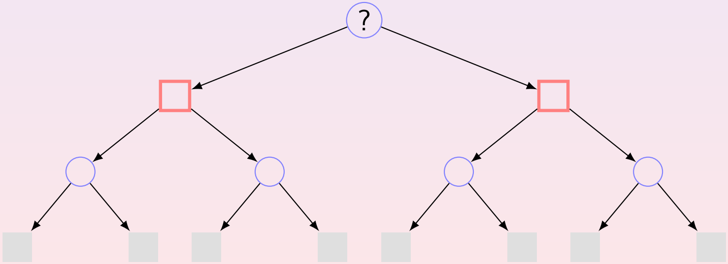
QBF-SAT in AP.

In PSPACE!

Theorem

$\mathrm{AP} \subseteq \textsc{Pspace}$.

Deterministic backtracking algorithm (min-max style)!

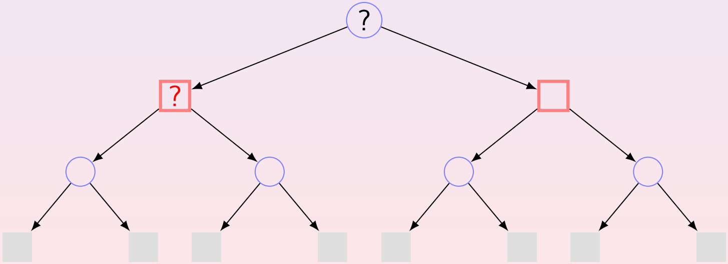


In PSPACE!

Theorem

$\mathrm{AP} \subseteq \mathrm{PSPACE}$.

Deterministic backtracking algorithm (min-max style)!

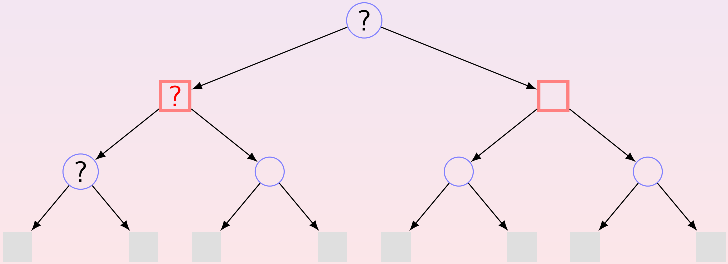


In PSPACE!

Theorem

AP $\subseteq$ PSPACE.

Deterministic backtracking algorithm (min-max style)!

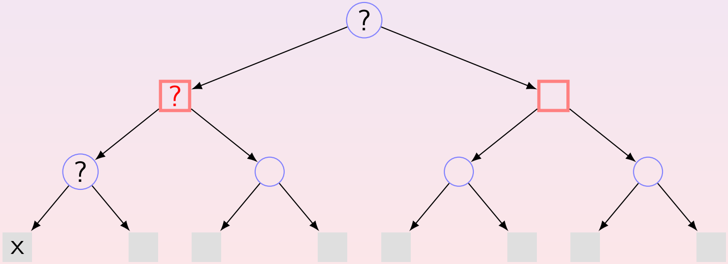


In PSPACE!

Theorem

AP $\subseteq$ PSPACE.

Deterministic backtracking algorithm (min-max style)!

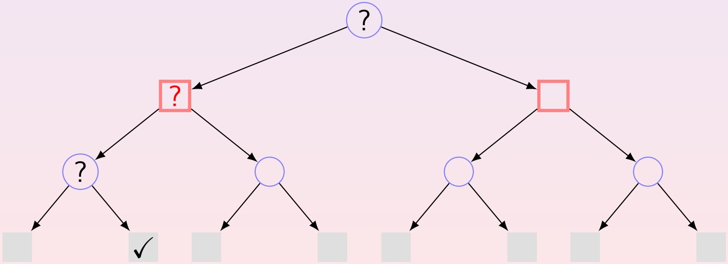


In PSPACE!

Theorem

AP $\subseteq$ PSPACE.

Deterministic backtracking algorithm (min-max style)!

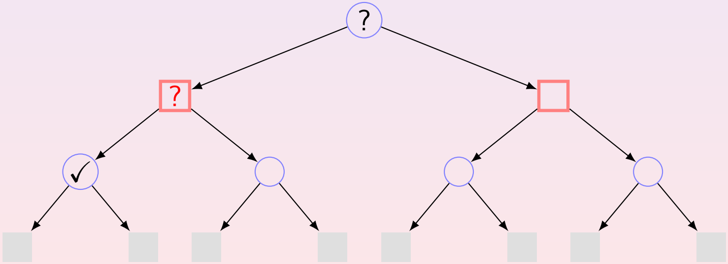


In PSPACE!

Theorem

AP $\subseteq$ PSPACE.

Deterministic backtracking algorithm (min-max style)!

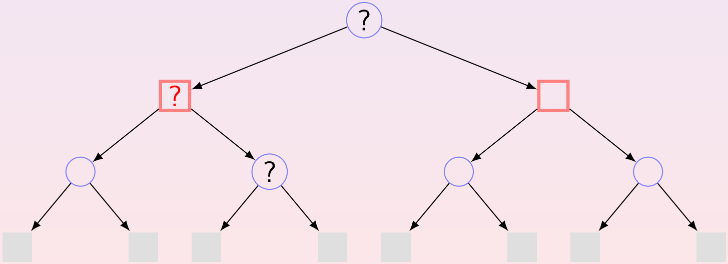


In PSPACE!

Theorem

AP $\subseteq$ PSPACE.

Deterministic backtracking algorithm (min-max style)!

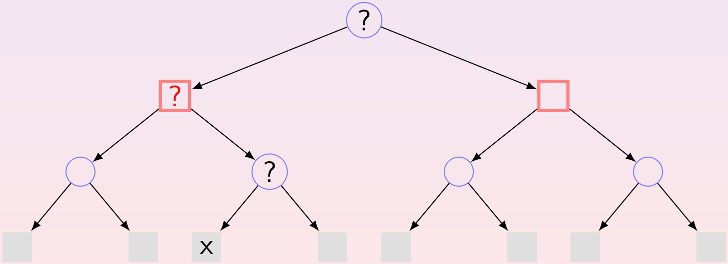


In PSPACE!

Theorem

AP $\subseteq$ PSPACE.

Deterministic backtracking algorithm (min-max style)!

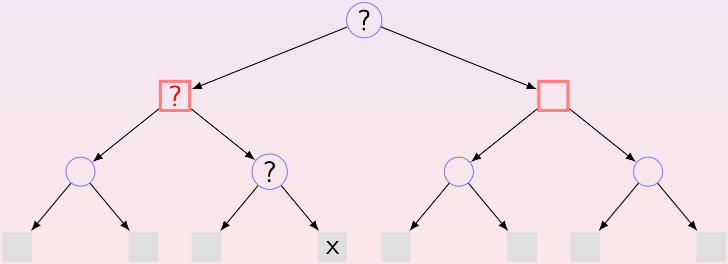


In PSPACE!

Theorem

AP $\subseteq$ PSPACE.

Deterministic backtracking algorithm (min-max style)!

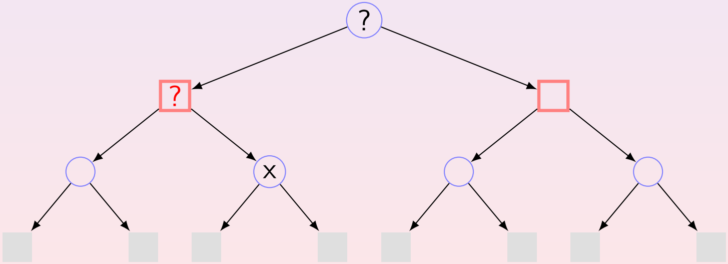


In PSPACE!

Theorem

AP $\subseteq$ PSPACE.

Deterministic backtracking algorithm (min-max style)!

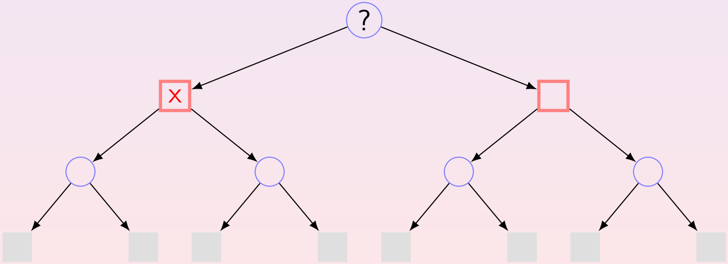


In PSPACE!

Theorem

AP $\subseteq$ PSPACE.

Deterministic backtracking algorithm (min-max style)!



Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
3. Proving hardness
4. PSPACE
   - Two-player poly-time games
   - One-player poly-space games
   - PSPACE-complete problems
5. Big theorems in computational complexity

Definition

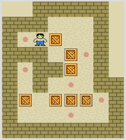
Definition

NPSPACE is the class of decision problems such that there is a one-player algo that decides it in polynomial-space.

Example

SOKOBAN:

- Input: a Sokoban position;
- Output: yes if the player can win from that position; no otherwise.



One-player algo deciding SOKOBAN in poly-space

```
one-player-algo sokoban(position)
    while position is not winning do
        choose position := one of the next possible positions
                                                from position
    win
```

Savitch's theorem

Theorem

$\mathrm{AP} = \textsc{Pspace} = \mathrm{NP}\textsc{space}.$

Savitch's theorem

Theorem

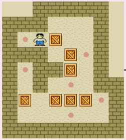
$\mathrm{AP} \underset{\subseteq}{=} \mathrm{P}\textsc{space} \underset{\subseteq}{=} \mathrm{NP}\textsc{space}.$

Proof of $\mathrm{NP}\textsc{space} \subseteq \mathrm{AP}$

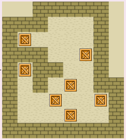
One player poly-space algo $\xrightarrow{\text{reformulation}}$ Two player poly-time algo

Two-player poly-time algo equivalent to Sokoban

- Player one chooses a mid-position of Sokoban
- Player two chooses which part to check



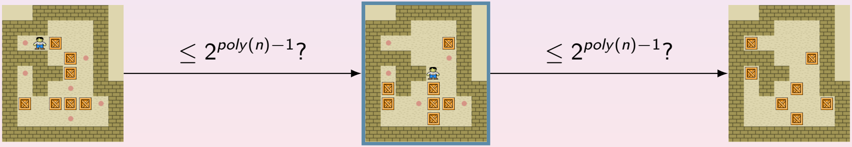
$\leq 2^{poly(n)}$ steps?



$n =$ size of the Sokoban position given in input

Two-player poly-time algo equivalent to Sokoban

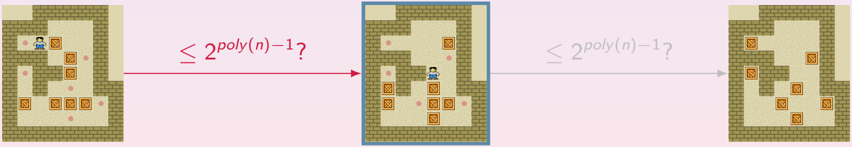
- Player one chooses a mid-position of Sokoban
- Player two chooses which part to check



$n =$ size of the Sokoban position given in input

Two-player poly-time algo equivalent to Sokoban

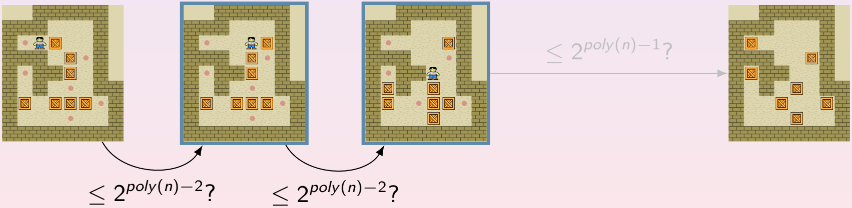
- Player one chooses a mid-position of Sokoban
- Player two chooses which part to check



$n =$ size of the Sokoban position given in input

Two-player poly-time algo equivalent to Sokoban

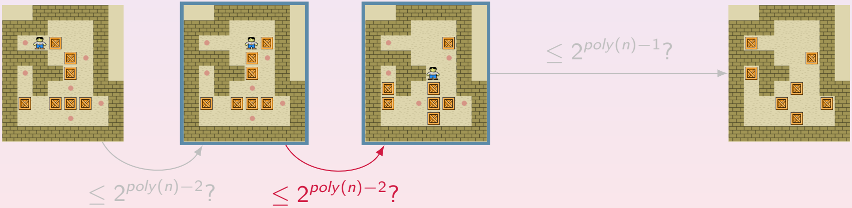
- Player one chooses a mid-position of Sokoban
- Player two chooses which part to check



$n =$ size of the Sokoban position given in input

Two-player poly-time algo equivalent to Sokoban

- Player one chooses a mid-position of Sokoban
- Player two chooses which part to check



$n =$ size of the Sokoban position given in input

Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
3. Proving hardness
4. PSPACE
   - Two-player poly-time games
   - One-player poly-space games
   - PSPACE-complete problems
5. Big theorems in computational complexity

PSPACE-complete problems: some two-player poly-time games

QBF-SAT

- Input: a closed quantified binary formula φ;
- Output: yes if φ is true; no otherwise.

$$\exists p, \forall q, \forall r, \exists s (p \rightarrow (q \wedge r \rightarrow s))$$

First-order query on a finite model

- input: a finite model $\mathcal{M}$, a first-order formula φ;
- output: yes if $\mathcal{M}$ satisfies φ, no otherwise.

$$\exists x, \forall y, (R(x, y) \rightarrow \exists z, p(f(x, y, z))$$

also HEX!

PSPACE-complete problems: some one-player poly-space games

Universality of a regular expression

- input: a regular expression e;
- output: yes if the language denoted by e is Σ^*, no otherwise.

Classical planning

- input: an initial state ι, a final state γ, description of actions;
- output: yes if γ is reachable from ι by executing some actions, no otherwise.

Also Sokoban!

Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
3. Proving hardness
4. PSPACE
5. Big theorems in computational complexity
   - About games
   - About LOGSPACE and NLOGSPACE
   - About non-uniform poly-time algorithm

Outline

Correspondence between alternating and usual classes

[Chandra, Stockmeyer, 1980, Alternations]

Theorem

$$\begin{array}{ccccccc} \mathrm{AP} & = & \textsc{Pspace} & \quad & \mathrm{A}\textsc{Logspace} & = & \mathrm{P} \\ \textsc{Aexptime} & = & \textsc{Expspace} & & \mathrm{AP}\textsc{space} & = & \textsc{Exptime} \\ \vdots & & \vdots & & \vdots & & \vdots \end{array}$$

Outline

1. Complexity classes defined with deterministic algorithms
2. Abstracting the combinatorics
3. Proving hardness
4. PSPACE
5. Big theorems in computational complexity
   - About games
   - About LOGSPACE and NLOGSPACE
   - About non-uniform poly-time algorithm

Definitions

Definition

LOGSPACE is the class of decision problems decided by an algorithm in logarithmic space.

- The input is read-only
- $\sim$ a constant number of pointers

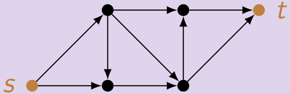
Definition

NLOGSPACE is the class of decision problems decided by a one-player algo in logarithmic space.

Important results

Theorem

Reachability in a directed graph is NLOGSPACE*-complete.*

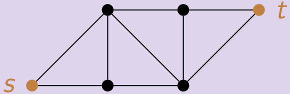


Theorem (Immerman-Szelepcsényi, 1988)

NLOGSPACE = *co*NLOGSPACE

Theorem (Reingold, 2005)

Reachability in an undirected graph is in LOGSPACE.



Outline

We believe $\mathrm{P} \neq \mathrm{NP}$, even we believe $\mathrm{P}_{/poly} \neq \mathrm{NP}$

Definition

$\mathrm{P}_{/poly}$ = class of decision problems s.th. there are deterministic algorithms $A_1, A_2, \ldots$ such that

$$A_n \text{ decides inputs of size } n \text{ in } poly(n).$$

Theorem (Karp and Lipton, 1980)

If $\mathrm{NP} \subseteq \mathrm{P}_{/poly}$, the polynomial hierarchy collapses at level 2:

$$\boxed{\begin{array}{c}\textit{Two-player}\\ \textit{poly-time algo}\\ \textit{with a fixed number of alternations}\end{array}} \xrightarrow{\textit{transformation}} \boxed{\begin{array}{c}\textit{Two-player}\\ \textit{poly-time algo}\\ \textit{with 2 alternations}\end{array}}$$

This tutorial did not present...

- Formal definitions of complexity classes with Turing machines
- The obscure terminology (non-determinism, alternation, certificates, reduction, etc.)
- Other types of reduction: log-space, FO, etc.
- Probabilistic complexity classes
- Quantum complexity classes
- Counting and Toda's theorem
- Descriptive complexity
- Circuit complexity
- Other computation models: RAM, etc.
- Interactive proofs
- Parametrized complexity
- ...

Thank you for your attention!