Computational Complexity in Automated Planning and Scheduling

Jussi Rintanen
Department of Computer Science
Aalto University
Helsinki, Finland

This Tutorial

- Why is Complexity (very) important in Planning?
- Brief overview of basic concepts
- NP vs. PSPACE
- Succinctness vs. Complexity
- Planning and Scheduling outside PSPACE
- types of search trees vs. plans
  - OR-trees for sequential plans
  - AND-OR-trees for branching plans
- Solvability vs. Unsolvability
  - Numeric state variables
  - Continuous change
  - Belief states and Partial Observability
What?

- How much resources (CPU time, memory) are needed?
- Most problems exponential. Question: How exponential?
- Connections between problems: (polynomial time) transformations
  → complexity classes
  → classification of problems by classes

Much of standard complexity theory [Pap94] relevant to planning
Why?

Complexity is one of the important properties of an algorithm.

- **Correctness**: Are the solutions correct?
- **Completeness**: Is a solution found whenever one exists?
- **Complexity**: Is resource use of the algorithm reasonable?

If complexity is unknown, it is difficult to do anything about it.
⇒ Analyze. Then look at ways attacking it.
What Is It Good For? (In Planning)

Research on Algorithms
Is an algorithm as good as it can be?
- Does it use more resources than it should? Why?

Research on Modeling Languages
What can be expressed in a modeling language?
- Comparisons between modeling languages
- Mappings between languages (time, size)

Research on Applications
How should an application problem be solved?
- Match or a mismatch with a modeling language?
- Match or a mismatch with an algorithm?
What Is It Good For? (In Planning)

Research on Algorithms
Is an algorithm as good as it can be?
- Does it use more resources than it should? Why?

Research on Modeling Languages
What can be expressed in a modeling language?
- Comparisons between modeling languages
- Mappings between languages (time, size)

Research on Applications
How should an application problem be solved?
- Match or a mismatch with a modeling language?
- Match or a mismatch with an algorithm?
# What Is It Good For? (In Planning)

## Research on Algorithms

Is an **algorithm** as good as it can be?
- Does it use more resources than it should? Why?

## Research on Modeling Languages

What can be **expressed** in a **modeling language**?
- Comparisons between modeling languages
- Mappings between languages (time, size)

## Research on Applications

How should **an application problem** be solved?
- Match or a mismatch with a modeling language?
- Match or a mismatch with an algorithm?
Big O in Analysis of Algorithms

Standard tool in analyzing algorithms is asymptotic resource consumption in the worst-case.

**Big O - Asymptotic growth rates**

function $f(n)$ is in $O(g(n))$ iff

$$f(n) \leq c \cdot g(n)$$

for all $n \geq 0$ and some $c$.

For input of size $n$:

- logarithmic resource consumption
  - $O(\log n)$
- polynomial resource consumption
  - $O(n^k)$
- exponential resource consumption
  - $O(2^n k)$
- doubly exponential resource consumption
  - $O(2^{2^n k})$

...
Big O in Analysis of Algorithms

Standard tool in analyzing algorithms is asymptotic resource consumption in the worst-case.

**Big O - Asymptotic growth rates**

Function $f(n)$ is in $O(g(n))$ iff

$$f(n) \leq c \cdot g(n)$$

for all $n \geq 0$ and some $c$.

For input of size $n$:

- logarithmic resource consumption $O(\log n)$
- polynomial resource consumption $O(n^k)$
- exponential resource consumption $O(2^n)$
- doubly exponential resource consumption $O(2^{2^n})$
## Coarseness of Big O vs. Complexity Classes

<table>
<thead>
<tr>
<th>complexity class</th>
<th>best algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>memory big O</td>
</tr>
<tr>
<td>co-NP</td>
<td>$\mathcal{O}(p(n))$</td>
</tr>
<tr>
<td>NP</td>
<td>$\mathcal{O}(p(n))$</td>
</tr>
<tr>
<td>PSPACE</td>
<td>$\mathcal{O}(p(n))$</td>
</tr>
</tbody>
</table>

- Big practical differences between (co-)NP and PSPACE!
- Big O only applies to *algorithms*, not directly to *problems*.

⇒ Structural Complexity Theory: Theory of Complexity Classes
Applicability to Reactive Control (Robotics)

- Literature mostly about **complete plans, covering all future situations**
- Selecting **only the next action** sometimes *believed* to reduce complexity (as a part of the sense-plan-act loop in closed-loop control)
- Most results in the literature apply to both
  - on-line planning (only first action chosen, repeatedly)
  - off-line planning (full plan constructed before execution)
- Existence of a complete plan (satisfying some criteria) equivalent to the possibility of selecting the first/next action (satisfying same criteria).
  ⇒ No complexity reduction by doing things on-line
Applicability to Reactive Control (Robotics)

- Literature mostly about complete plans, covering all future situations
- Selecting only the next action sometimes believed to reduce complexity (as a part of the sense-plan-act loop in closed-loop control)
- Most results in the literature apply to both
  - on-line planning (only first action chosen, repeatedly)
  - off-line planning (full plan constructed before execution)
- Existence of a complete plan (satisfying some criteria) equivalent to the possibility of selecting the first/next action (satisfying same criteria).
  ⇒ No complexity reduction by doing things on-line
Applicability to Reactive Control (Robotics)

- Literature mostly about **complete plans, covering all future situations**
- Selecting **only the next action** sometimes **believed** to reduce complexity (as a part of the sense-plan-act loop in closed-loop control)
- Most results in the literature apply to both
  - on-line planning (only first action chosen, repeatedly)
  - off-line planning (full plan constructed before execution)
- Existence of a complete plan (satisfying some criteria) **equivalent** to the possibility of selecting the first/next action (satisfying same criteria).
  ⇒ No complexity reduction by doing things on-line
Polynomial-Time Transformations (Karp reductions)

A decision problem $X$ is transformed in polynomial time to decision problem $Y$ (written $X \leq_p Y$) if and only if there is function $f$ such that

1. $f$ is computable in polynomial time, and
2. for all $s, s \in X$ if and only if $f(s) \in Y$.

Significance:

1. If $X \leq_p Y$ and $Y$ has an algorithm, then so has $X$.
2. If $X \leq_p Y$ and $Y$ is easy to solve (tractable), then so is $X$.
3. If $X \leq_p Y$ and $X$ is difficult to solve (intractable), then so is $Y$. 
Example

Let $G = \langle N, E \rangle$ be a graph. Then $G$ is in 3-COLORABLE if and only if the conjunction of the following is in SAT.

\begin{align*}
(R_i \lor G_i \lor B_i) & \text{ for all } i \in N \quad (1) \\
\neg (R_i \land R_j) & \text{ for all } \{i, j\} \in E \\
\neg (G_i \land G_j) & \text{ for all } \{i, j\} \in E \\
\neg (B_i \land B_j) & \text{ for all } \{i, j\} \in E \quad (4)
\end{align*}

Therefore $3\text{-COLORABLE} \leq_p \text{SAT}$

Polynomial-Time Transformations

Complexity

Introduction

Robotics

Basics

Transformations

Turing Machines

NP

More Classes

Classical Planning

Succinctness

Outside PSPACE

Temporal Planning

Branching Plans

Unsolvability

Conclusion

References
Insights from PTIME Transformations (1970ies)
[Coo71, Kar72]

No efficient algorithms?

Efficient (poly-time) algorithms
Insights from PTIME Transformations (1970ies) [Coo71, Kar72]

Intractable problems

Tractable problems

NP-complete

co-NP-complete
Insights from PTIME Transformations (1970ies)
[Coo71, Kar72]

NP-complete → co-NP-complete

Intractable problems

Tractable problems
Resource Requirements of Computation

Computation:
- sequence of states of the computation device, indicating the contents of its memory/registers/...
- changes from state to state follow the "program" of the device

```
memory
01101101101100101011
10110110101101101100
01101101101100110011
0100101101101011101
00110011001010110110
10110110101100110011
01001011011011001011
00110011100110110010
01101100110011001010
01101101110011110011
01001011110011011010
00110011110110001100
01101100001110011010
10110110101100011010
00110011001010110110
01101100110011001010
10110110101100011010
00110011001010110110
01101100110011001010
10110110101100011010
00110011110110001100
01101100001110011010
01101100110011001010
10110110101100011010
00110011110110001100
01101100001110011010
01101100110011001010
```

```
time
```
Turing Machines

Turing machine configuration (state, R/W head, tape contents):

\[ \begin{array}{cccccc}
q_1 \\
\downarrow \\
A & B & B & A & B & [ ] [ ] [ ] [ ] [ ] [ ] \ldots
\end{array} \]

Transitions of the Turing machine:

<table>
<thead>
<tr>
<th>old state</th>
<th>read</th>
<th>write</th>
<th>new state</th>
<th>move</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_1)</td>
<td>A</td>
<td>A</td>
<td>(q_3)</td>
<td>L</td>
</tr>
<tr>
<td>(q_1)</td>
<td>B</td>
<td>A</td>
<td>(q_1)</td>
<td>N</td>
</tr>
<tr>
<td>(q_1)</td>
<td>[ ]</td>
<td>A</td>
<td>(q_1)</td>
<td>N</td>
</tr>
<tr>
<td>(q_1)</td>
<td>[ ]</td>
<td>[ ]</td>
<td>(q_1)</td>
<td>R</td>
</tr>
<tr>
<td>(q_2)</td>
<td>A</td>
<td>B</td>
<td>(q_2)</td>
<td>R</td>
</tr>
<tr>
<td>(q_2)</td>
<td>B</td>
<td>A</td>
<td>(q_2)</td>
<td>R</td>
</tr>
<tr>
<td>(q_2)</td>
<td>[ ]</td>
<td>B</td>
<td>(q_1)</td>
<td>N</td>
</tr>
<tr>
<td>(q_2)</td>
<td>[ ]</td>
<td>[ ]</td>
<td>(q_1)</td>
<td>R</td>
</tr>
<tr>
<td>(q_3)</td>
<td>A</td>
<td>B</td>
<td>(q_1)</td>
<td>L</td>
</tr>
<tr>
<td>(q_3)</td>
<td>B</td>
<td>B</td>
<td>(q_3)</td>
<td>R</td>
</tr>
<tr>
<td>(q_3)</td>
<td>[ ]</td>
<td>B</td>
<td>(q_1)</td>
<td>N</td>
</tr>
<tr>
<td>(q_3)</td>
<td>[ ]</td>
<td>[ ]</td>
<td>(q_1)</td>
<td>R</td>
</tr>
</tbody>
</table>
Turing machines

Definition

A Turing machine $\langle \Sigma, Q, \delta, q_0, g \rangle$ consists of

1. an alphabet $\Sigma$ (a set of symbols),
2. a set $Q$ of internal states,
3. a transition function $\delta$ that maps $\langle q, s \rangle$ to a tuple $\langle s', q', m \rangle$ where $q, q' \in Q$, $s \in \Sigma \cup \{|, \square\}$, $s' \in \Sigma \cup \{|\}$ and $m \in \{L, N, R\}$.
4. an initial state $q_0 \in Q$, and
5. a labeling $g : Q \rightarrow \{\text{accept, reject, } \exists\}$ of states.
Nondeterministic Computation: Graph Coloring

Nodes 1, 2 and 3 are made Red, Green or Blue.
Resource-limited nondeterministic Turing machines (NDTM) represent search with bounds on memory use and size of search tree.

Non-determinism = choice of branch of a computation/search tree

Memory consumption = max. used tape in any configuration

Time consumption = max. path length in the tree
It was observed in early 1970ies [Coo71] that there are many important problems that

- do not seem to have polynomial-time algorithms,
- can be easily solved with non-deterministic TMs, and
- can be transformed to each other in poly-time.
The Complexity Class NP

Definition

A decision problem $X$ gives a yes or no answer for a given input $x$, often written as a set membership question $x \in X$?

Definition

The complexity class NP consists of decision problems that are solvable by a non-deterministic Turing machine in a polynomial number of steps.
NP-Hardness and NP-Completeness

**Definition (NP-hardness)**

A decision problem $Y$ is **NP-hard** iff $X \leq_p Y$ for every $X$ in NP.

**Definition (NP-completeness)**

A decision problem $Y$ is **NP-complete** iff $Y$ is NP-hard and $Y$ is in NP.
Theorem

SAT (the satisfiability problem of the propositional logic) is NP-complete.

Proof.

Membership in NP: guess a satisfying assignment.

NP-hardness: Proof similar to Planning as SAT [KS92]. Express non-deterministic TM executions of given length: change between two consecutive configurations easily expressible as a Boolean formula.
More Complexity Classes

Definition

\text{DTIME}(f) \text{ is the class of decision problems solved by a deterministic Turing machine in } \mathcal{O}(f(n)) \text{ time when } n \text{ is the input string length.}

Definition

\text{NTIME}(f) \text{ is defined similarly for nondeterministic Turing machines.}

Definition

\text{DSPACE}(f) \text{ is the class of decision problems solved by a deterministic Turing machine in } \mathcal{O}(f(n)) \text{ space when } n \text{ is the input string length.}
Definitions of Complexity Classes

Complexity classes express worst-case time and memory requirements.

\[
\begin{align*}
P &= \bigcup_{k \geq 0} \text{DTIME}(n^k) \\
\text{EXP} &= \bigcup_{k \geq 0} \text{DTIME}(2^n^k) \\
2\text{-EXP} &= \bigcup_{k \geq 0} \text{DTIME}(2^{2^n^k}) \\
\text{NP} &= \bigcup_{k \geq 0} \text{NTIME}(n^k) \\
\text{NEXP} &= \bigcup_{k \geq 0} \text{NTIME}(2^n^k) \\
2\text{-NEXP} &= \bigcup_{k \geq 0} \text{NTIME}(2^{2^n^k}) \\
\text{PSPACE} &= \bigcup_{k \geq 0} \text{DSPACE}(n^k) \\
\text{EXPSPACE} &= \bigcup_{k \geq 0} \text{DSPACE}(2^n^k) \\
\text{NLOGSPACE} &= \text{NSPACE}(\log n) \\
\text{NPSPACE} &= \bigcup_{k \geq 0} \text{NSPACE}(n^k) \\
\text{NEXPSPACE} &= \bigcup_{k \geq 0} \text{NSPACE}(2^n^k)
\end{align*}
\]
Overview of Complexity Classes

- **2-NEXP**  
- **2-EXP**  
- **EXPSPACE**  
- **NEXP**  
- **EXP**  
- **PSPACE**  
- **PH**  
- **NP**  
- **P**  
- **NLOGSPACE**  

- **provably intractable**
- **presumably intractable**
- **tractable**
Classical Planning

Properties:
- untimed (asynchronous): one action a time, change instantaneous
- one known initial state
- actions are deterministic, environment otherwise static
- objective is to reach a goal state (finite executions)

This is the state space search problem also in
- problem-solving (search) in AI
- reachability analysis in Computer-Aided Verification
- model-checking (non-modal safety properties) in Computer-Aided Verification
- other areas
Simulation of PSPACE Turing machines

Match polynomially space-bounded Turing machines $\sim$ classical planning:

1. Turing machine configurations $\sim$ states
2. Turing machine transitions $\sim$ actions
3. initial configuration $\sim$ initial state
4. accepting configurations $\sim$ goal states

For simulation of PSPACE TMs a number of state variables that is \textit{polynomial} in input string length suffices.
Simulation of PSPACE Turing machines

Turing machine with $\Sigma = \{u, v, w\}$, input string of length $n = 4$, space bound $p(n) = n^2 = 16$, internal states $Q = \{q_1, q_2, q_3\}$.

State variables in the corresponding planning problem:

<table>
<thead>
<tr>
<th>state $q_1$:</th>
<th>q_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>state $q_2$:</td>
<td>q_2</td>
</tr>
<tr>
<td>state $q_3$:</td>
<td>q_3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tape cell:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>\cdots</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W head:</td>
<td>$h_0$</td>
<td>$h_1$</td>
<td>$h_2$</td>
<td>$h_3$</td>
<td>\cdots</td>
<td>$h_{15}$</td>
<td>$h_{16}$</td>
</tr>
<tr>
<td>symbol $u$:</td>
<td>$u_1$</td>
<td>$u_2$</td>
<td>$u_3$</td>
<td>\cdots</td>
<td>$u_{15}$</td>
<td>$u_{16}$</td>
<td></td>
</tr>
<tr>
<td>symbol $v$:</td>
<td>$v_1$</td>
<td>$v_2$</td>
<td>$v_3$</td>
<td>\cdots</td>
<td>$v_{15}$</td>
<td>$v_{16}$</td>
<td></td>
</tr>
<tr>
<td>symbol $w$:</td>
<td>$w_1$</td>
<td>$w_2$</td>
<td>$w_3$</td>
<td>\cdots</td>
<td>$w_{15}$</td>
<td>$w_{16}$</td>
<td></td>
</tr>
<tr>
<td>symbol $\Box$:</td>
<td>$\Box_1$</td>
<td>$\Box_2$</td>
<td>$\Box_3$</td>
<td>\cdots</td>
<td>$\Box_{15}$</td>
<td>$\Box_{16}$</td>
<td></td>
</tr>
</tbody>
</table>
Simulation of PSPACE Turing machines

Example

True state variables marked with color:

<table>
<thead>
<tr>
<th>TM config.</th>
<th>work tape</th>
<th>R/W head</th>
<th>state</th>
<th>plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$\hat{u}uvv$</td>
<td>$uuvuwvwuwuwuwuw$</td>
<td>$h_0h_1h_2h_3h_4h_5h_6$</td>
<td>$q_1q_2q_3$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$v\hat{u}uvv$</td>
<td>$uuvuwvwuwuwuwuw$</td>
<td>$h_0h_1h_2h_3h_4h_5h_6$</td>
<td>$q_1q_2q_3$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$vuw\hat{u}vv$</td>
<td>$uuvuwvwuwuwuwuw$</td>
<td>$h_0h_1h_2h_3h_4h_5h_6$</td>
<td>$q_1q_2q_3$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$vuw\hat{u}vv$</td>
<td>$uuvuwvwuwuwuwuw$</td>
<td>$h_0h_1h_2h_3h_4h_5h_6$</td>
<td>$q_1q_2q_3$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$vw\hat{u}wv$</td>
<td>$uuvuwvwuwuwuwuw$</td>
<td>$h_0h_1h_2h_3h_4h_5h_6$</td>
<td>$q_1q_2q_3$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$vw\hat{u}wv$</td>
<td>$uuvuwvwuwuwuwuw$</td>
<td>$h_0h_1h_2h_3h_4h_5h_6$</td>
<td>$q_1q_2q_3$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$\hat{v}uwv\hat{v}$</td>
<td>$uuvuwvwuwuwuwuw$</td>
<td>$h_0h_1h_2h_3h_4h_5h_6$</td>
<td>$q_1q_2q_3$</td>
</tr>
</tbody>
</table>

Preconditions of $a_{u,q_1,1}$ are $u_1$, $q_1$, $h_1$.

Effects of $a_{u,q_1,1}$ are

- $\neg q_1$, $q_2$ (state changes from $q_1$ to $q_2$)
- $\neg h_1$, $h_2$ (head location changes from 1 to 2)
- $\neg u_1$, $v_1$ (symbol $u$ replaced by $v$ at location 1)

obtained directly from the TMs transition function.
Classical Planning is in PSPACE

- The PSPACE-hardness result provides a lower bound on the complexity of deterministic planning.
- We next give an upper bound on the complexity by showing that the problem belongs to PSPACE.
- Hence the problem is PSPACE-complete, determining complexity exactly.
- It is not known whether NP ≠ PSPACE or even P ≠ PSPACE, but the result is still useful because for all practical purposes we can assume that NP ≠ PSPACE.
- For example, we may conclude that there is, most likely, no polynomial-time transformation from planning to SAT.
Classical Planning is in PSPACE

- The PSPACE-hardness result provides a lower bound on the complexity of deterministic planning.
- We next give an upper bound on the complexity by showing that the problem belongs to PSPACE.
- Hence the problem is PSPACE-complete, determining complexity exactly.
- It is not known whether NP ≠ PSPACE or even P ≠ PSPACE, but the result is still useful because for all practical purposes we can assume that NP ≠ PSPACE.
- For example, we may conclude that there is, most likely, no polynomial-time transformation from planning to SAT.
The PSPACE-hardness result provides a lower bound on the complexity of deterministic planning.

We next give an upper bound on the complexity by showing that the problem belongs to PSPACE.

Hence the problem is PSPACE-complete, determining complexity exactly.

It is not known whether NP \( \neq \) PSPACE or even P \( \neq \) PSPACE, but the result is still useful because for all practical purposes we can assume that NP \( \neq \) PSPACE.

For example, we may conclude that there is, most likely, no polynomial-time transformation from planning to SAT.
The PSPACE-hardness result provides a lower bound on the complexity of deterministic planning.

We next give an upper bound on the complexity by showing that the problem belongs to PSPACE.

Hence the problem is PSPACE-complete, determining complexity exactly.

It is not known whether NP ≠ PSPACE or even P ≠ PSPACE, but the result is still useful because for all practical purposes we can assume that NP ≠ PSPACE.

For example, we may conclude that there is, most likely, no polynomial-time transformation from planning to SAT.
Recursive algorithm for testing \( m \)-step reachability between two states with \( \log m \) memory consumption.

\[
\text{reach}(s_0, s_8, 3) \quad | \quad \text{reach}(s, s', 2) \quad | \quad \text{reach}(s, s', 1) \quad | \quad \text{reach}(s, s', 0)
\]

\[
s_0 \quad s_1 \quad s_2 \quad s_3 \quad s_4 \quad s_5 \quad s_6 \quad s_7 \quad s_8
\]
Classical planning is in PSPACE

Algorithm

Testing whether a plan of length \( \leq 2^n \) exists:

\[
\text{PROCEDURE } \text{reach}(s,s',n) \\
\text{IF } n = 0 \text{ THEN} \\
\text{IF } s = s' \text{ OR } s' = \text{exec}_a(s) \text{ for some action } a \\
\text{THEN RETURN true} \\
\text{ELSE RETURN false;} \\
\text{ELSE} \\
\text{FOR all states } s'' \text{ DO} \\
\text{IF } \text{reach}(s,s'',n - 1) \text{ AND } \text{reach}(s'',s',n - 1) \\
\text{THEN RETURN true} \\
\text{END} \\
\text{RETURN false;}
\]

This algorithm does not store the plan anywhere (would violate the space bound!) but could be modified to output it.
Many types of NP-complete problems solved effectively: guess a solution (with good heuristics!)

Same far harder with PSPACE-problems:
- polynomial number of guesses not enough
- either exponential number of guesses, or
- search tree is an AND-OR tree.

Why real-world planning and scheduling often feasible?

- Schedules always and sequential plans often polynomial size
  \[ \Rightarrow \text{problems are in NP!} \]
- effective heuristics available
  - real-world P&S
    - some plan/schedule (with unlimited resources) trivial to find
    - solvable with scalable constraint-based methods (MILP, CP, ...)
    - good schedules can be found for very large problem instances
  - IPC benchmark sets (classical/temporal planning without optimization)
NP vs. PSPACE for Planning and Scheduling

- Many types of NP-complete problems solved effectively: guess a solution (with good heuristics!)
- Same far harder with PSPACE-problems:
  - polynomial number of guesses not enough
  - either exponential number of guesses, or
  - search tree is an AND-OR tree.

Why real-world planning and scheduling often feasible?

- Schedules always and sequential plans often polynomial size
  ⇒ problems are in NP!
- effective heuristics available
  - real-world P&S
    - some plan/schedule (with unlimited resources) trivial to find
    - solvable with scalable constraint-based methods (MILP, CP, ...)
    - good schedules can be found for very large problem instances
  - IPC benchmark sets (classical/temporal planning without optimization)
Succinctness

There is no one unique classical planning problem. Differences: succinctness/compactness of input to the planning algorithm.

1. flat/enumerative representation (as a graph: nodes, arcs)
2. ground actions (can represent an exponential size graph)
3. schematic actions (can represent a doubly exponential size graph)
Planning Problems given as a Graph
Blocks world with three blocks
state variables: RonG, RonB, GonR, GonB, BonR, BonG, Rontable, Gontable, Bontable, Rclr, Gclr, Bclr

actions:
moveRfromGtoB = (\{RonG, Rclr, Bclr\}, \{\neg RonG, RonB, Gclr, \neg Bclr\})
moveRfromBtoG = (\{RonB, Rclr, Gclr\}, \{\neg RonB, RonG, Bclr, \neg Gclr\})
moveGfromRtoB = (\{GonR, Gclr, Bclr\}, \{\neg GonR, GonB, Rclr, \neg Bclr\})
moveGfromBtoR = (\{GonB, Gclr, Rclr\}, \{\neg GonB, GonR, Bclr, \neg Rclr\})

This representation has size $\mathcal{O}(n^3)$ for $n$ of blocks, representing 1, 3, 13, 73, 501, 4051, 37633, 394353, 4596553, ... states for 1, 2, 4, 5, ... blocks, respectively.
Planning Problems as Sets of Schematic Actions

variable domains: BLOCKS = \{ A, B, C, \ldots \}

state variables: on(x,y), ontable(x), clr(x) for all x, y \in BLOCKS

actions:
move(b, s, t) = (\{ t \neq b \neq s, on(b, s), clr(b), clr(t) \}, \{ \neg on(b, s), on(b, t), clr(s), \neg clr(t) \})
movefromtable(b, t) = (\{ b \neq t, ontable(b), clr(b), clr(t) \}, \{ \neg ontable(b), on(b, t) \})
movetotable(b, s) = (\{ b \neq s, on(b, s), clr(b) \}, \{ \neg on(b, s), ontable(b) \})

where \{ b, s, t \} \subseteq BLOCKS

This representation has size \( O(n) \) for \( n \) blocks.

(Ground actions exponential in size of schematic actions only when \textbf{arity of predicates} grows.)
Question: Succinctness Reduces Complexity?

Some problems are hard to solve, due to their large size. If problem instance can be represented succinctly (compact, factored representation), will it have regularities that allow solving it more efficiently?

Answer to a high number of graph problems is negative [GW83, Loz88, LB90]: cost of computation in real-world terms is not reduced (in worst case)
Levels of Succinctness for Classical Planning

<table>
<thead>
<tr>
<th>representation</th>
<th>complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph (nodes, arcs)</td>
<td>NLOGSPACE-complete</td>
</tr>
<tr>
<td>ground actions</td>
<td>PSPACE-complete [GW83, Loz88, LB90, Byl94]</td>
</tr>
<tr>
<td>schematic actions</td>
<td>EXPSPACE-complete, undecidable [ENS91]</td>
</tr>
</tbody>
</table>

In the worst case, for graphs of size $2^{2^n}$ these respectively correspond to

1. $\mathcal{O}(n)$ time in size $\mathcal{O}(2^{2^n})$ of a graph
2. $\mathcal{O}(2^n)$ time in size $\mathcal{O}(2^n)$ ground action set
3. $\mathcal{O}(2^{2^n})$ time in size $\mathcal{O}(n)$ schematic action set

This is same $\mathcal{O}(2^{2^n})$ in the size of the graph, in all three cases!!
Complexity vs. Expressivity

Classical planning can be expressed in terms of

- STRIPS
  - preconditions: conjunctions of $x = 0$, $x = 1$
  - effects: assignments $x := 0$, $x := 1$

- PDDL/ADL: STRIPS + Boolean connectives $\land$, $\lor$, $\neg$ and IF-THEN

- arbitrary propositional formulas (cf. BDD-based model-checking [BCL+94], Planning as SAT [KS92, Rin09])

Can the same planning problems be expressed in all formalisms?
Complexity vs. Expressivity

Different answers, depending what is meant:

1. In all cases, planning is PSPACE-complete, so decision problems “is there a plan” intertranslatable.¹

2. Translations so that the transition graph remains the same:
   - Translating PDDL/ADL into STRIPS exponential size/time.
   - Translating Boolean formulas into PDDL exponential size/time.

Lessons:

- Even if complexity is same, a modeling language can be exponentially more compact.
- Simpler languages do not (necessarily) offer performance benefits, and may make compact modeling impossible.

¹ Under partial observability, features of actions has stronger impact [Rin04].
Many extensions within PSPACE possible:
- bounded integers, bounded rationals, floats, enums
- any other bounded-size data
- more complex effects
  - assignments $a[x] := b[y]$ [Gef00]
  - sequential composition $(e_1 ; e_2)$ [Rin08]

Practical works often unnecessarily limit to STRIPS, even when more general language straightforward to handle [Rin06, Rin08]

Extensions that make classical planning unsolvable discussed later...
Classical Planning: Theory vs. Practice

How do actual algorithms perform w.r.t. theoretical requirements?

All algorithms use exponential time. Memory consumption differs:

<table>
<thead>
<tr>
<th>algorithm</th>
<th>memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*, greedy best-first</td>
<td>exp</td>
</tr>
<tr>
<td>IDA*</td>
<td>poly</td>
</tr>
<tr>
<td>BDDs [CBM90, BCM+92]</td>
<td>exp</td>
</tr>
<tr>
<td>SAT with DPLL [KS92]</td>
<td>poly</td>
</tr>
<tr>
<td>SAT with CDCL</td>
<td>exp (^2)</td>
</tr>
<tr>
<td>QBF with QBF-DPLL [Rin01]</td>
<td>poly</td>
</tr>
</tbody>
</table>

Best practical algorithms exceed theoretical requirements. Why?

\(^2\)Conflict-Driven Clause Learning algorithm [MSS99, MMZ\(^+\)01] has no inherent exponential memory requirement, but also no clear polynomial bounds.

\(^3\)Test if a plan exists. Output plan one action at a time.
Outside NP and PSPACE

- Undecidable
- Continuous/hybrid
- POMDP optimal
- 2-EXP
- Configurations
- EXPSPACE
- EXP
- EXPSPACE
- PSPACE
- Classical
- MDP / conditional full obs.
- Conditional partial obs.
- Temporal

Complexity
Introduction
Robotics
Basics
Classical Planning
Succinctness
Outside PSPACE
Temporal Planning
Branching Plans
Unsolvability
Conclusion
References
Classical Planning

- $a_0$: go to work
- $a_1$: write a report
- $a_2$: go to grocery store
- $a_3$: buy food
- $a_4$: go home
- $a_5$: make dinner

- time not explicit
- an action $\sim$ change between two consecutive states
- only one action at a time
More realistic model for many applications

Several actions simultaneously active

An action can change the state at several time points

Possibility of taking an action depends on other current and earlier actions

Effects of an action might depend on whether other actions are taken simultaneously
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
Temporal State = Static State + Event Agenda
EXPSPACE: Exponentially Long Tapes

- If (static) state is poly-size, where to encode an exponentially long tape?
- Dynamic state (= future events) can be exponential
- Proof idea: spread the TM working tape over timeline
  [Rin07b]

<table>
<thead>
<tr>
<th>time</th>
<th>1st configuration</th>
<th>2nd configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>cell</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>R/W</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>□</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>q0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>q1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Branching Plans

Sequential plans (= classical planning) sufficient when
- there is unique (known) initial state,
- all actions are deterministic

When actions or the environment non-deterministic, action choice depends on the past (observations)
- More complex forms of plans required:
  - mapping from states to actions (full observability)
  - mapping from belief states to actions (partial observability)
  - programs/controllers that output actions (partial observability)
- Complexity far higher, from EXP to 2-EXP to unsolvable [Lit97, Rin04, MHC03].
- Analyzed with alternating Turing machines (ATM).
Computation with Alternation (AND-OR Trees)
Alternating Turing Machines

Nondeterministic Turing machines = search trees with OR nodes
Alternating Turing machines = search trees with both AND and OR nodes

Originally defined to model games and game trees [CKS81].

Definition

A Turing machine \( \langle \Sigma, Q, \delta, q_0, g \rangle \) consists of

1. an alphabet \( \Sigma \) (a set of symbols),
2. a set \( Q \) of internal states,
3. a transition function \( \delta \) that maps \( \langle q, s \rangle \) to a set of tuples \( \langle s', q', m \rangle \) where \( q, q' \in Q, s \in \Sigma \cup \{ |, \Box \}, s' \in \Sigma \) and \( m \in \{ L, N, R \} \).
4. an initial state \( q_0 \in Q \), and
5. a labeling \( g : Q \rightarrow \{ \text{accept, reject, } \exists, \forall \} \) of states.
Complexity Classes Defined with Alternation

Define complexity classes

\[
\begin{align*}
\text{APTIME} &= \bigcup_{k \geq 0} \text{ATIME}(n^k) \\
\text{APSPACE} &= \bigcup_{k \geq 0} \text{ASPACE}(n^k) \\
\text{AEXP} &= \bigcup_{k \geq 0} \text{ATIME}(2^{n^k}) \\
\text{AEXPSPACE} &= \bigcup_{k \geq 0} \text{ASPACE}(2^{n^k})
\end{align*}
\]

Interestingly, poly-space = alternating poly-time, and exponential time = alternating poly-space [CKS81]:

\[
\begin{align*}
\text{PSPACE} &= \text{APTIME} \\
\text{EXPSPACE} &= \text{AEXP} \\
\text{EXP} &= \text{APSPACE} \\
\text{2-EXP} &= \text{AEXPSPACE}
\end{align*}
\]
EXP-hardness of Conditional Planning

**Proof idea:** Extend the PSPACE-hardness proof for classical planning with *alternation* (computation of an ATM is an AND/OR tree.)

- **∃ states:** *one deterministic action* is chosen to the plan, *from several possible ones.*
- **∀ states:** *one nondeterministic action simulates all possible transitions.*
- In branching plans, actions for ∀ states are followed by observing the new configuration and continuing the simulation accordingly.
PSPACE-hardness proof of classical planning

∃

one deterministic action

∃

one deterministic action

∃

one deterministic action

acc
PSPACE=NPSPACE-hardness proof of classical planning

Simulation of Nondeterministic Turing Machines

∃

∃

∃

∃

acc

rej

rej

acc

acc

acc

acc

acc

rej

acc
Simulation of Alternating Turing Machines

EXP=APSPACE-hardness proof with full observability

one nondeterministic action

two deterministic actions

acc rej rej acc acc acc rej acc
Correspondence of ATM executions and plans

An accepting computation tree is mapped to a plan:
1. $\exists$-configuration to action
2. $\forall$-configuration to observation + action
Partial Observability vs. Branching
Extending Classical Planning with Branching and Observability Limitations [Rin04]

Alternation $\sim$ Branching plans
Exponential tape $\sim$ Belief states
Partial Observability vs. Branching
Extending Classical Planning with Branching and Observability Limitations [Rin04]

2-EXP

exp\text{ponential tape}

\text{alternation}

PSPACE

\text{alternation}

EXP

EXPSPACE

exp\text{ponential tape}

Alternation \sim \text{Branching plans}

Exponential tape \sim \text{Belief states}
Polynomial Hierarchy

Polynomial Hierarchy = PSPACE problems with limited alternation

Example

$\Sigma_2^p = \text{trees with polynomial depth and } \exists \text{ nodes followed by } \forall \text{ nodes}$
Polynomial Hierarchy = PSPACE problems with limited alternation

Example

$$\Pi^P_2 = \text{trees with polynomial depth and } \forall \text{ nodes followed by } \exists \text{ nodes}$$
The Polynomial Hierarchy

\[ \text{PH} = \bigcup_{i \geq 1} (\Sigma^p_i \cup \Pi^p_i) \]

\[ \text{PSPACE} \]

\[ \text{NP} = \Sigma^p_1 \quad \text{co-NP} = \Pi^p_1 \]

\[ \exists \longleftarrow \forall \]

\[ \exists \forall \exists \longleftarrow \forall \exists \exists \]

\[ \exists \forall \exists \cdots \exists \forall \exists \]

unbounded \[ \exists \exists \exists \cdots \exists \exists \]

all finite \[ \exists \exists \exists \cdots \exists \exists \]

Complexity

Introduction

Robotics

Basics

Classical Planning

Succinctness

Outside PSPACE

Temporal Planning

Branching Plans

Alternation

Polynomial Hierarchy

Unsolvability

Conclusion

References
Planning Problems in the Polynomial Hierarchy

Conditional Planning with Poly-Size Plans

There is ($\exists$) a poly-size plan such that for all contingencies ($\forall$) there is an execution leading to goals.

Most naturally expressed as a quantified Boolean formula [Sto76] with prefix $\exists \forall \exists$ [Rin99], but as the problem is in $\Sigma^p_2$, it is possible to express it as a QBF with prefix $\exists \forall$ [Rin07a].

Conditional Planning with Short Executions

There is ($\exists$) an action such that for all ($\forall$) contingencies there is ($\exists$) an action such that for all ($\forall$) contingencies ••• a goal state is reached.

Conditional planning with $n$ consecutive actions expressible as a $n$ alternations QBF prefix $\exists \forall \exists \cdots \exists$ [Tur02]. This covers all of the Polynomial Hierarchy.
Planning Problems in the Polynomial Hierarchy

Conditional Planning with Poly-Size Plans

There is ($\exists$) a poly-size plan such that for all contingencies ($\forall$) there is an execution leading to goals.

Most naturally expressed as a quantified Boolean formula [Sto76] with prefix $\exists \forall \exists$ [Rin99], but as the problem is in $\Sigma^p_2$, it is possible to express it as a QBF with prefix $\exists \forall$ [Rin07a].

Conditional Planning with Short Executions

There is ($\exists$) an action such that for all ($\forall$) contingencies there is ($\exists$) an action such that for all ($\forall$) contingencies $\cdots$ a goal state is reached.

Conditional planning with $n$ consecutive actions expressible as a

\[ n \text{ alternations} \]

QBF prefix $\exists \forall \exists \cdots \exists$ [Tur02]. This covers all of the Polynomial Hierarchy.
Uncertainty in Scheduling

Most of the scheduling problems encountered in practice are NP-complete.

Harder scheduling problems typically involve uncertainty:

- expected makespan for stochastic task durations #P-hard [Hag88]
- scheduling with uncertain resource availability [Rin13]
  - general case PSPACE-complete
  - \( \Pi^p_2 \)-complete when all uncertainty resolved in the beginning
  - \( \Sigma^p_2 \)-complete when contingent schedules are poly-size
Planning is not only hard, but sometimes impossible.

Main forms of unsolvable planning problems:
- unbounded numeric state variables (extension of classical planning)
- continuous change (planning with hybrid systems)
- optimal probabilistic planning with partial observability (optimal POMDPs)

Impossibility associated with infinite state spaces and states of unbounded size
Integer problems are unsolvable:

- **Halting problem** of general Turing machines encodable in classical planning + integers
- unbounded working tape (∼ two stacks of a pushdown automaton) encodable with:
  - two integer variables, +1, test-even, multiply-by-2, divide-by-2
  - two integer variables, +1, test-even, shift-left, shift-right
  - other possibilities
- **Practical ways out:**
  - use bounded integers only (finite-state systems)
  - consider bounded length plans only (⇒ incompleteness)
Need to remember unbounded past history
Finding optimal POMDP policies unsolvable [MHC03]
Proof by reduction from probabilistic automata [Paz71]
Practical ways out:
  - finite-memory policies (⇒ incompleteness) [MKKC99, LLS+99, CCD16]
  - practical POMDP algorithms don’t prove optimality
reachability (planning) for hybrid systems undecidable [HKPV95, CL00, PC07]
  many problems with only 2 continuous variables undecidable!!

decidable cases for reachability: rectangular automata [HKPV95], 2-d PCD [AMP95], planar multi-polynomial systems [ČV96]

semi-decision procedures: no termination when plans don’t exist.
Limit to short plans (⇒ incompleteness)
- non-linear polynomials highly complex [BD07], with functions like \( \text{sine} \) unsolvable
- some solvers give approximation guarantees [GKC13]
- approximation problematic due to lack of \textit{stability}: small errors accumulate and cause plans to fail

A main challenge is the development of more useful solvers
- General-purpose methods in general do not work well
Complexity Classes vs. Types of Planning

undecidable \uparrow \uparrow 2\text{-EXP} \uparrow \uparrow \text{EXPSPACE} \uparrow \uparrow \text{NEXP} \uparrow \uparrow \text{EXP} \uparrow \uparrow \text{PSPACE} \uparrow \uparrow \text{PH} \uparrow \uparrow \text{NP} \uparrow \uparrow \text{P} \uparrow \uparrow \text{NLOGSPACE}

- optimal POMDPs [MHC03]
- non-deterministic partially observable [Rin04]
- unobservable ("conformant") [HJ00, Rin04]
- probabilistic [Lit97]; succinct MDPs [MGLA00]
- classical [Byl94]
- branching plans with short executions [Tür02]
- poly-length classical
- flat MDPs [PT87]
- s-t reachability
References

[AMP95] Eugene Asarin, Oded Maler, and Amir Pnueli.
Reachability analysis of dynamical systems having piecewise-constant derivatives.

Symbolic model checking for sequential circuit verification.

Symbolic model checking: $10^{20}$ states and beyond.

[BD07] Christopher W. Brown, and James H. Davenport.
The complexity of quantifier elimination and cylindrical algebraic decomposition.

[Byl94] Tom Bylander.
The computational complexity of propositional STRIPS planning.

[CBM90] Olivier Coudert, Christian Berthet, and Jean Christophe Madre.
Verification of synchronous sequential machines based on symbolic execution.

[CCD16] Krishnendu Chatterjee, Martin Chmelik, and Jessica Davies.
A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs.

Alternation.
[CL00] Franck Cassez and Kim Larsen.  
The impressive power of stopwatches.  

[Coo71] Stephen A. Cook.  
The complexity of theorem-proving procedures.  

Deciding reachability for planar multi-polynomial systems.  

Complexity, decidability and undecidability results for domain-independent planning: A detailed analysis.  

[Gef00] Héctor Geffner.  
Functional STRIPS: a more flexible language for planning and problem solving.  

dReal: An SMT solver for nonlinear theories over the reals.  

Succinct representations of graphs.  
(See [Loz88] for a correction.).

Computational complexity of PERT problems.  
References III

[HJ00] Patrik Haslum and Peter Jonsson.
Some results on the complexity of planning with incomplete information.

What’s decidable about hybrid automata?

Reducibility among combinatorial problems.

Planning as satisfiability.

[LB90] Antonio Lozano and José L. Balcázar.
The complexity of graph problems for succinctly represented graphs.

Probabilistic propositional planning: Representations and complexity.

[LLS+99] Christopher Lusena, Tong Li, Shelia Sittinger, Chris Wells, and Judy Goldsmith.
My brain is full: When more memory helps.

NP-hardness of succinct representations of graphs.
Complexity

Introduction

Robotics

Basics

Classical Planning

Succinctness

Outside PSPACE

Temporal Planning

Branching Plans

Unsolvability

Conclusion

References

References IV


