Computational Complexity of Automated Planning and Scheduling

Computational Complexity in Automated Planning and Scheduling

Jussi Rintanen
Department of Computer Science
Aalto University
Helsinki, Finland

ICAPS 2016, London, U.K.

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.
$\Rightarrow$ 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?

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 $\mathcal{O}(g(n))$ iff

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

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

For input of size n:

$\vdots$

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

$\vdots$

Coarseness of Big O vs. Complexity Classes

	best algorithms	
complexity class	memory big O	time big O
co-NP	$\mathcal{O}(p(n))$	$\mathcal{O}(2^n)$
NP	$\mathcal{O}(p(n))$	$\mathcal{O}(2^n)$
PSPACE	$\mathcal{O}(p(n))$	$\mathcal{O}(2^n)$

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

$\Longrightarrow$ 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

Polynomial-Time Transformations

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.

Polynomial-Time Transformations

Example

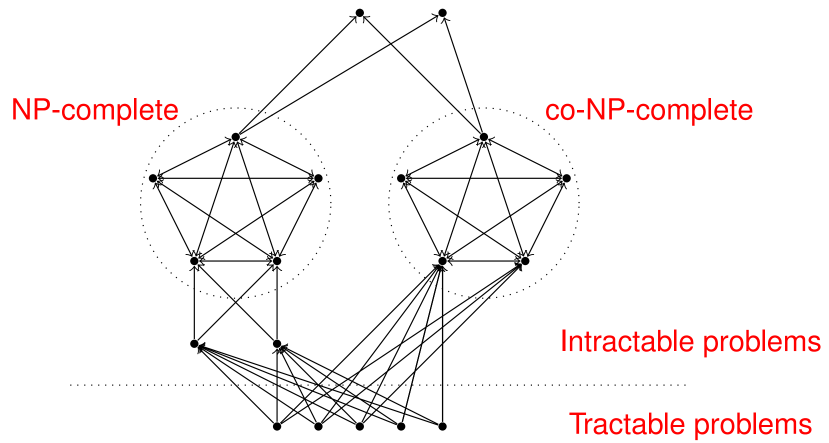
Let $G = \langle N, E \rangle$ be a graph. Then G is in 3-COLORABLE if and only if the conjunction of the following is in SAT.

$$(R_i \vee G_i \vee B_i) \text{ for all } i \in N \quad (1)$$
$$\neg(R_i \wedge R_j) \text{ for all } \{i,j\} \in E \quad (2)$$
$$\neg(G_i \wedge G_j) \text{ for all } \{i,j\} \in E \quad (3)$$
$$\neg(B_i \wedge B_j) \text{ for all } \{i,j\} \in E \quad (4)$$

Therefore 3-COLORABLE$\leq_p$SAT

Insights from PTIME Transformations (1970ies)

[Coo71, Kar72]



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

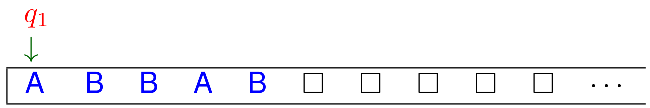
memory

```
01101101101100101011
10110110101101101100
01101101101100110011
01001010110110101101
00110011001010110110
10110110101100110011
01001010101101101100
00110011110110101101
00110011001010110110
01101100110011001010
01101101110011110011
01001010110110101101
00110011001010110110
01101100110011001010
10110110101100011010
00110011110110001100
01101100001110011010
```

time

Turing Machines

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



Transitions of the Turing machine:

old state	read	write	new state	move
q_1	A	A	q_3	L
q_1	B	A	q_1	N
q_1	□	A	q_1	N
q_1	$\vert$	$\vert$	q_1	R
q_2	A	B	q_2	R
q_2	B	A	q_2	R
q_2	□	B	q_1	N
q_2	$\vert$	$\vert$	q_1	R
q_3	A	B	q_1	L
q_3	B	B	q_3	R
q_3	□	B	q_1	N
q_3	$\vert$	$\vert$	q_1	R

Turing machines

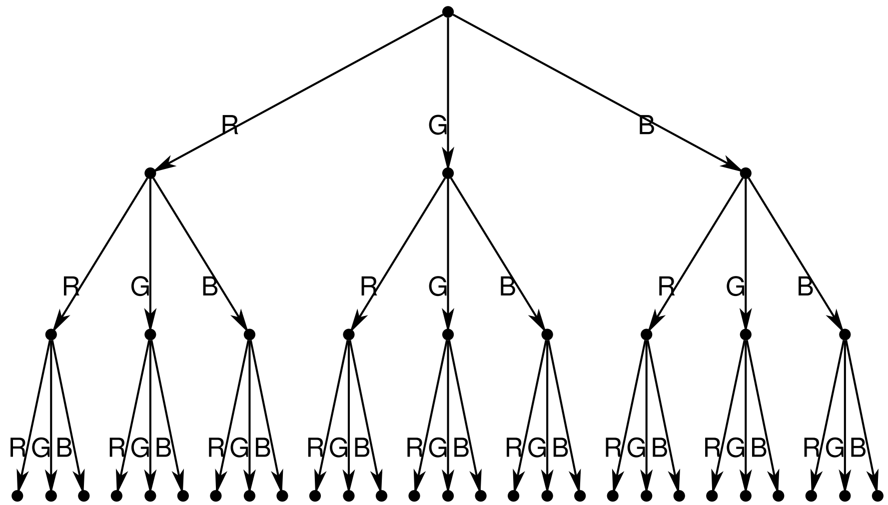
Definition

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

1. an alphabet Σ (a set of symbols),
2. a set Q of internal states,
3. a transition function δ 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 $g : Q \to \{\text{accept}, \text{reject}, \exists\}$ of states.

Nondeterministic Computation: Graph Coloring

Nodes 1, 2 and 3 are made Red, Green or Blue.



Nondeterministic Computation

- 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

The Complexity Class NP: Motivation

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.

NP-Completeness of SAT

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

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

Definition

NTIME(f) is defined similarly for nondeterministic Turing machines.

Definition

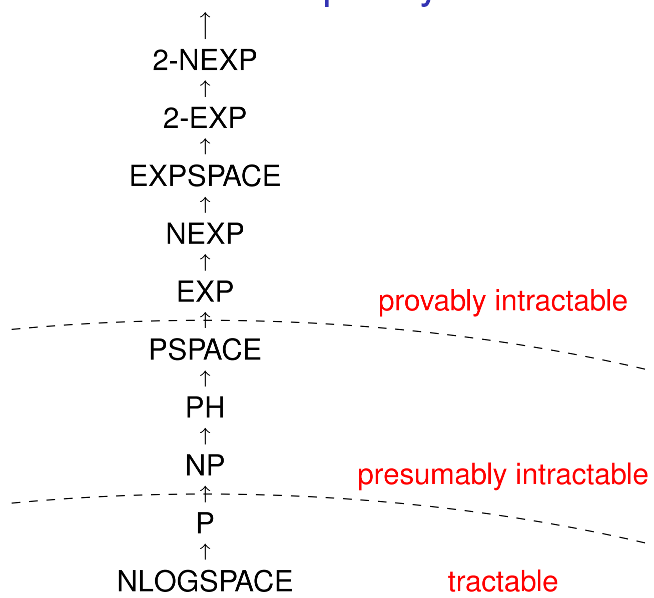
DSPACE(f) is the class of decision problems solved by a deterministic Turing machine in $\mathcal{O}(f(n))$ space when n is the input string length.

Definitions of Complexity Classes

Complexity classes express worst-case time and memory requirements.

$$\begin{aligned}
\text{P} &= \textstyle\bigcup_{k\geq 0} \text{DTIME}(n^k) \\
\text{EXP} &= \textstyle\bigcup_{k\geq 0} \text{DTIME}(2^{n^k}) \\
\text{2-EXP} &= \textstyle\bigcup_{k\geq 0} \text{DTIME}(2^{2^{n^k}}) \\
\\
\text{NP} &= \textstyle\bigcup_{k\geq 0} \text{NTIME}(n^k) \\
\text{NEXP} &= \textstyle\bigcup_{k\geq 0} \text{NTIME}(2^{n^k}) \\
\text{2-NEXP} &= \textstyle\bigcup_{k\geq 0} \text{NTIME}(2^{2^{n^k}}) \\
\\
\text{PSPACE} &= \textstyle\bigcup_{k\geq 0} \text{DSPACE}(n^k) \\
\text{EXPSPACE} &= \textstyle\bigcup_{k\geq 0} \text{DSPACE}(2^{n^k}) \\
\\
\text{NLOGSPACE} &= \text{NSPACE}(\log n) \\
\text{NPSPACE} &= \textstyle\bigcup_{k\geq 0} \text{NSPACE}(n^k) \\
\text{NEXPSPACE} &= \textstyle\bigcup_{k\geq 0} \text{NSPACE}(2^{n^k})
\end{aligned}$$

Overview of Complexity Classes



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 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:

state q_1:	q_1
state q_2:	q_2
state q_3:	q_3

tape cell:	0	1	2	3	$\cdots$	15	16
R/W head:	h_0	h_1	h_2	h_3	$\cdots$	h_{15}	h_{16}
symbol u:		u_1	u_2	u_3	$\cdots$	u_{15}	u_{16}
symbol v:		v_1	v_2	v_3	$\cdots$	v_{15}	v_{16}
symbol w:		w_1	w_2	w_3	$\cdots$	w_{15}	w_{16}
symbol $\square$:		$\square_1$	$\square_2$	$\square_3$	$\cdots$	$\square_{15}$	$\square_{16}$

Simulation of PSPACE Turing machines

Example

True state variables marked with color:

TM config.	state variable values: work tape	R/W head	state	plan
$q_1 \mid \hat{u}vuvv\square$	$\widehat{uvw}uvwuvwuvwuvwuvw$	$h_0h_1h_2h_3h_4h_5h_6$	$q_1q_2q_3$	$a_{u,q_1,1}$
$q_2 \mid v\hat{v}uvv\square$	$uvw\widehat{uvw}uvwuvwuvwuvw$	$h_0h_1h_2h_3h_4h_5h_6$	$q_1q_2q_3$	$a_{v,q_2,2}$
$q_3 \mid vw\hat{u}vv\square$	$uvwuvw\widehat{uvw}uvwuvwuvw$	$h_0h_1h_2h_3h_4h_5h_6$	$q_1q_2q_3$	$a_{u,q_3,3}$
$q_3 \mid vwv\hat{v}v\square$	$uvwuvwuvw\widehat{uvw}uvwuvw$	$h_0h_1h_2h_3h_4h_5h_6$	$q_1q_2q_3$	$a_{v,q_3,4}$
$q_1 \mid vw\hat{v}wv\square$	$uvwuvw\widehat{uvw}uvwuvwuvw$	$h_0h_1h_2h_3h_4h_5h_6$	$q_1q_2q_3$	$a_{v,q_1,3}$
$q_3 \mid v\hat{w}uwv\square$	$uvw\widehat{uvw}uvwuvwuvwuvw$	$h_0h_1h_2h_3h_4h_5h_6$	$q_1q_2q_3$	$a_{w,q_3,2}$
$q_3 \mid \hat{v}uuwv\square$	$\widehat{uvw}uvwuvwuvwuvwuvw$	$h_0h_1h_2h_3h_4h_5h_6$	$q_1q_2q_3$	$a_{v,q_3,1}$

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.

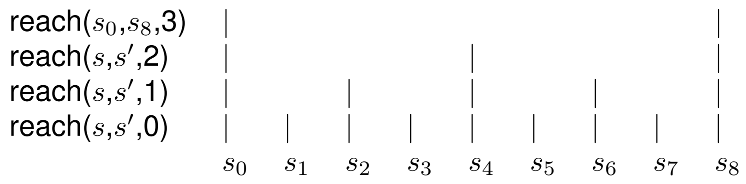
Classical Planning is in PSPACE

- 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$\neq$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.

Classical Planning is in PSPACE

Proof idea

Recursive algorithm for testing m-step reachability between two states with $\log m$ memory consumption.



Classical planning is in PSPACE

Algorithm

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

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

This algorithm does not store the plan anywhere (would violate the space bound!) but could be modified to output it.

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 $\Rightarrow$ 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)

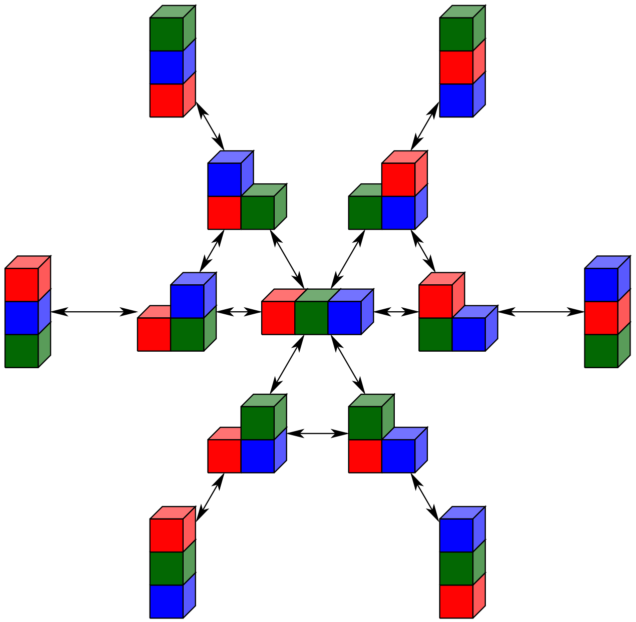
Succinctness

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



Planning Problems as Sets of (Ground) Actions

state variables: RonG, RonB, GonR, GonB, BonR, BonG, Rontable, Gontable, Bontable, Rclr, Gclr, Bclr

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

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

actions:
move(b,s,t) = ({ t≠b≠s, on(b,s), clr(b), clr(t)}, {¬on(b,s), on(b,t), clr(s), ¬clr(t)})
movefromtable(b,t) = ({ b≠t, ontable(b), clr(b), clr(t)}, {¬ontable(b), on(b,t)})
movetotable(b,s) = ({ b≠s, on(b,s), clr(b)}, {¬on(b,s), ontable(b)})
where $\{b, s, t\}$ ⊆BLOCKS

This representation has size $\mathcal{O}(n)$ for n blocks.

(Ground actions exponential in size of schematic actions only when arity of predicates grows.)

Succinctness

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

representation	complexity
graph (nodes, arcs)	NLOGSPACE-complete
ground actions	PSPACE-complete [GW83, Loz88, LB90, Byl94]
schematic actions	EXPSPACE-complete, undecidable [ENS91]

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 $\wedge$, $\vee$, $\neg$ 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].

Extensions to Classical Planning in PSPACE

- 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 (e1 ; e2) [Rin08]
- Practical works often unnecessarily limit to STRIPS, even when more general language straigthforward 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:

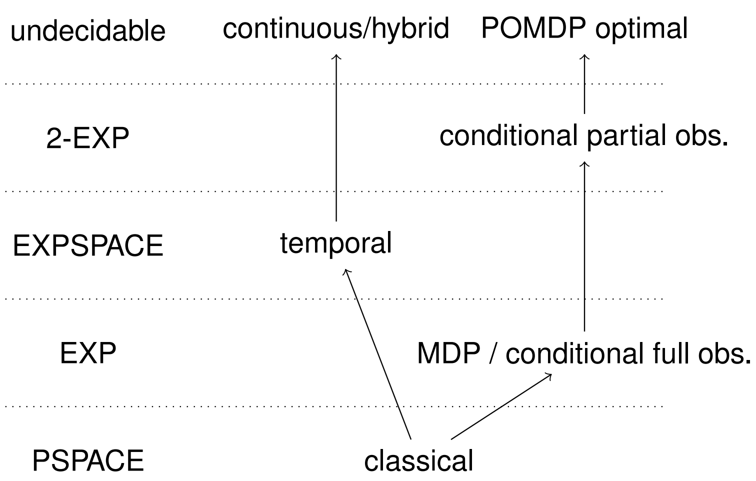
	memory consumption	
algorithm	poly-long plans	exp-long plans
A*, greedy best-first	exp	exp
IDA*	poly	exp
BDDs [CBM90, BCM+92]	exp	exp
SAT with DPLL [KS92]	poly	exp
SAT with CDCL	exp [2]	exp
QBF with QBF-DPLL [Rin01]	poly	poly [3]

Best practical algorithms exceed theoretical requirements. Why?

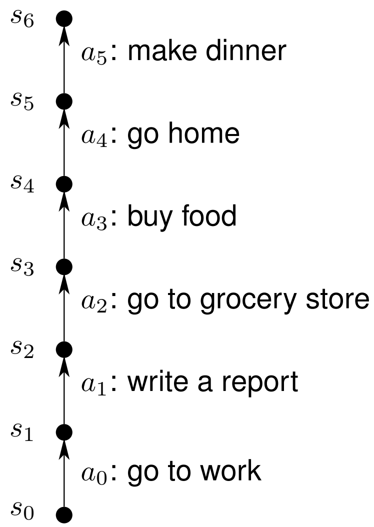
[2] Conflict-Driven Clause Learning algorithm [MSS99, MMZ+01] has no inherent exponential memory requirement, but also no clear polynomial bounds.

[3] Test if a plan exists. Output plan one action at a time.

Outside NP and PSPACE

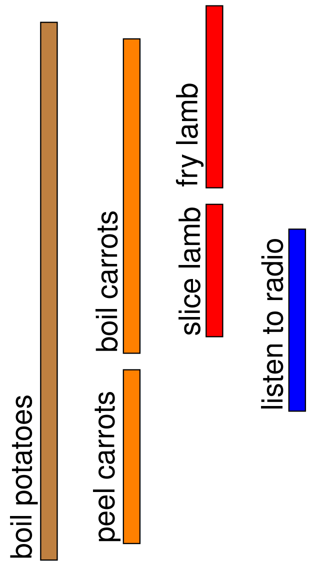


Classical Planning



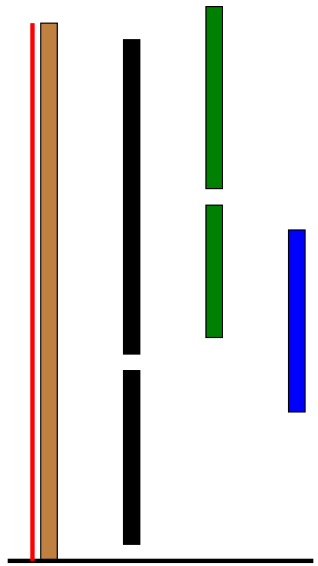
- time not explicit
- an action $\sim$ change between two consecutive states
- only one action at a time

Temporal Planning



- 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



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]

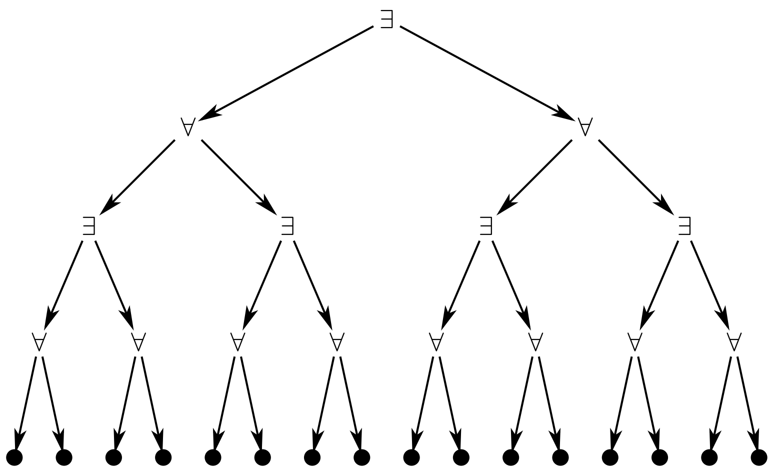
	1st configuration					2nd configuration					
time	0	1	2	3	4	5	6	7	8	9	$\ldots$
cell	0	1	2	3	4	0	1	2	3	4	$\ldots$
R/W	0	1	0	0	0	0	1	0	0	0	
A	0	1	0	0	0	0	0	0	0	0	
B	0	0	1	0	0	0	1	1	0	0	
$\Box$	0	0	0	1	1	0	0	0	1	1	
$\vert$	1	0	0	0	0	1	0	0	0	0	
q_0	1	1	1	1	1	1	0	0	0	0	
q_1	0	0	0	0	0	0	1	1	1	1	

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

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 Σ (a set of symbols),
2. a set Q of internal states,
3. a transition function δ 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 $g : Q \rightarrow \{\text{accept}, \text{reject}, \exists, \forall\}$ of states.

Complexity Classes Defined with Alternation

Complexity Classes

Define complexity classes

$$\begin{aligned} \text{APTIME} &= \textstyle\bigcup_{k \geq 0} \text{ATIME}(n^k) \\ \text{APSPACE} &= \textstyle\bigcup_{k \geq 0} \text{ASPACE}(n^k) \\ \text{AEXP} &= \textstyle\bigcup_{k \geq 0} \text{ATIME}(2^{n^k}) \\ \text{AEXPSPACE} &= \textstyle\bigcup_{k \geq 0} \text{ASPACE}(2^{n^k}) \end{aligned}$$

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

$$\begin{aligned} \text{PSPACE} &= \text{APTIME} \\ \text{EXPSPACE} &= \text{AEXP} \\ \text{EXP} &= \text{APSPACE} \\ \text{2-EXP} &= \text{AEXPSPACE} \end{aligned}$$

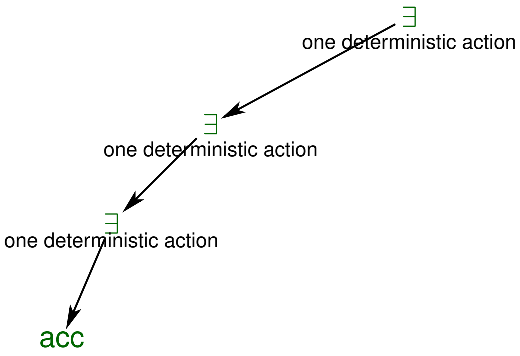
EXP-hardness of Conditional Planning

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

- $\exists$ states: *one deterministic action* is chosen to the plan, *from several possible ones.*
- $\forall$ states: *one nondeterministic action simulates all possible transitions.*
- In branching plans, actions for $\forall$ states are followed by observing the new configuration and continuing the simulation accordingly.

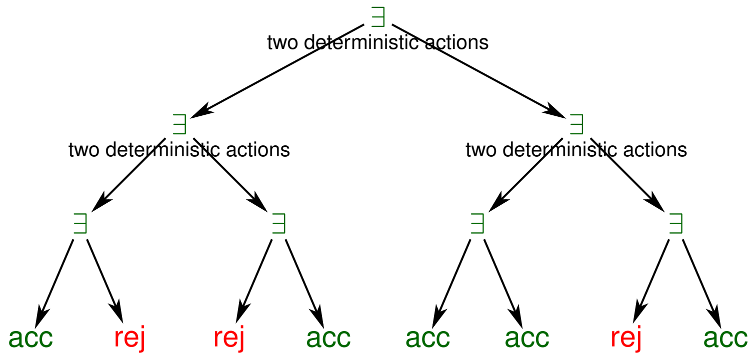
Simulation of Deterministic Turing Machines

PSPACE-hardness proof of classical planning



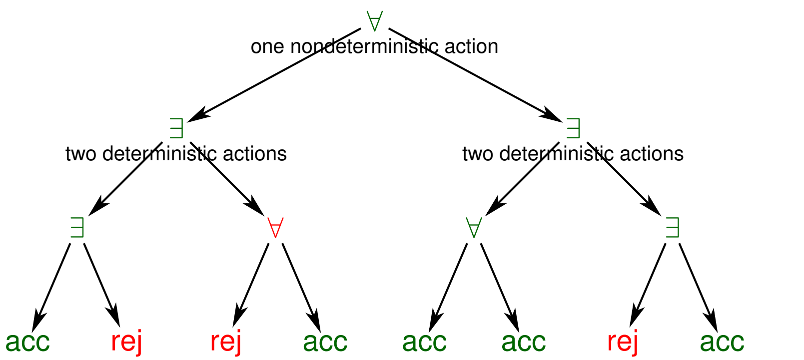
Simulation of Nondeterministic Turing Machines

PSPACE=NPSPACE-hardness proof of classical planning



Simulation of Alternating Turing Machines

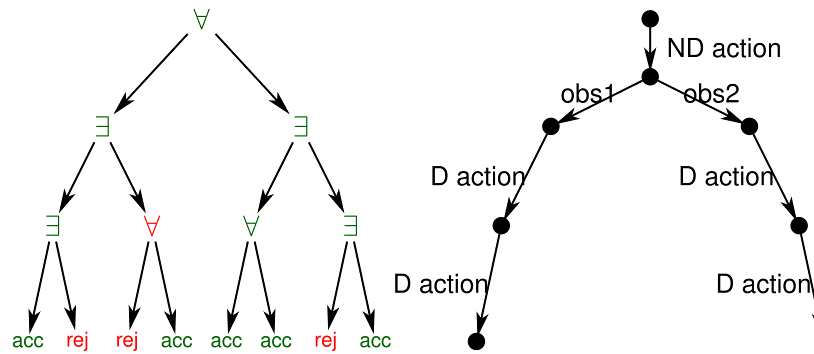
EXP=APSPACE-hardness proof with full observability



Correspondence of ATM executions and plans

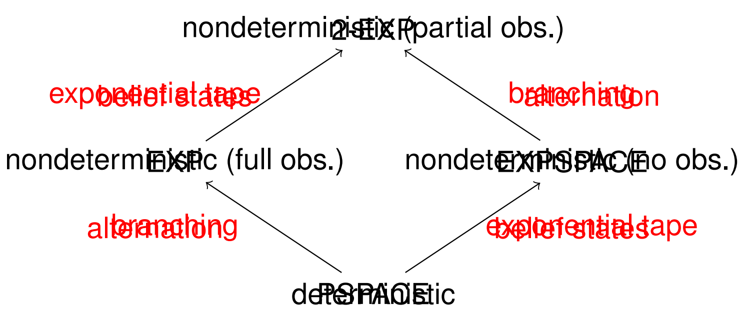
An accepting computation tree is mapped to a plan:

1. ∃-configuration to action
2. ∀-configuration to observation + action



Partial Observability vs. Branching

Extending Classical Planning with Branching and Observability Limitations [Rin04]



Alternation $\sim$ Branching plans

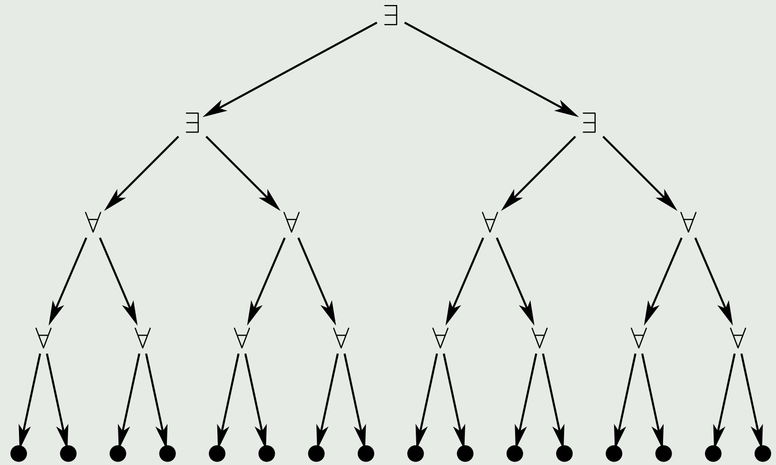
Exponential tape $\sim$ Belief states

Polynomial Hierarchy

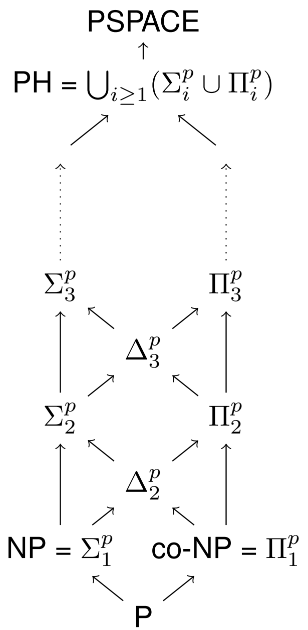
Polynomial Hierarchy = PSPACE problems with limited alternation

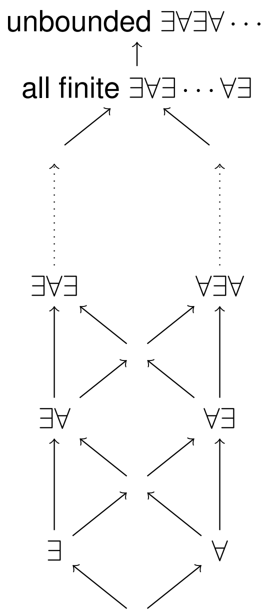
Example

Σ_2^p = trees with polynomial depth and $\exists$ nodes followed by $\forall$ nodes



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 Σ_2^p, 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 QBF prefix $\overbrace{\exists\forall\exists\cdots\exists}^{n \text{ alternations}}$ [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
  - Π_2^p-complete when all uncertainty resolved in the beginning
  - Σ_2^p-complete when contingent schedules are poly-size

Limits of Planning: Unsolvability

- 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

Unsolvability from (Unbounded) Numbers

Integer problems are unsolvable:

- Halting problem of general Turing machines encodable in classical planning + integers
- unbounded working tape ($\sim$ 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 ($\Rightarrow$ incompleteness)

Probabilistic Plans and Partial Observability

- 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 ($\Rightarrow$ incompleteness) [MKKC99, LLS$^+$99, CCD16]
  - practical POMDP algorithms don't prove optimality

Hybrid Systems: Solvability vs. Unsolvability

- 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.

Hybrid Systems: Solvability vs. Unsolvability

Approaches to Tackle the Unsolvability

- Limit to short plans (⇒ incompleteness)
  - non-linear polynomials highly complex [BD07], with functions like *sine* unsolvable
  - some solvers give approximation guarantees [GKC13]
  - approximation problematic due to lack of 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	optimal POMDPs [MHC03]
↑	
2-EXP	non-deterministic partially observable [Rin04]
↑	
EXPSPACE	unobservable ("conformant") [HJ00, Rin04]
↑	
NEXP	
↑	
EXP	probabilistic [Lit97];succinct MDPs [MGLA00]
↑	
PSPACE	classical [Byl94]
↑	
PH	branching plans with short executions [Tur02]
↑	
NP	poly-length classical
↑	
P	flat MDPs [PT87]
↑	
NLOGSPACE	s-t reachability

References I

[AMP95] Eugene Asarin, Oded Maler, and Amir Pnueli.
Reachability analysis of dynamical systems having piecewise-constant derivatives.
Theoretical Computer Science, 138(1):35–65, 1995.

[BCL+94] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. MacMillan, and David L. Dill.
Symbolic model checking for sequential circuit verification.
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 13(4):401–424, 1994.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang.
Symbolic model checking: 10^{20} states and beyond.
Information and Computation, 98(2):142–170, 1992.

[BD07] Christopher W. Brown, , and James H. Davenport.
The complexity of quantifier elimination and cylindrical algebraic decomposition.
In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, pages 54–60, 2007.

[BM99] Alberto Bemporad and Manfred Morari.
Control of systems integrating logic, dynamics, and constraints.
Automatica, 35(3):407–427, 1999.

[Byl94] Tom Bylander.
The computational complexity of propositional STRIPS planning.
Artificial Intelligence, 69(1-2):165–204, 1994.

[CBM90] Olivier Coudert, Christian Berthet, and Jean Christophe Madre.
Verification of synchronous sequential machines based on symbolic execution.
In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings*, volume 407 of *Lecture Notes in Computer Science*, pages 365–373. Springer-Verlag, 1990.

[CCD16] Krishnendu Chatterjee, Martin Chmelik, and Jessica Davies.
A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs.
In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016.

[CKS81] A. Chandra, D. Kozen, and L. Stockmeyer.
Alternation.
Journal of the ACM, 28(1):114–133, 1981.

References II

[CL00] Franck Cassez and Kim Larsen.
The impressive power of stopwatches.
In Catuscia Palamidessi, editor, *CONCUR 2000 - Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer-Verlag, 2000.

[Coo71] Stephen A. Cook.
The complexity of theorem-proving procedures.
In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[ČV96] Kārlis Čerāns and Juris Vīksna.
Deciding reachability for planar multi-polynomial systems.
In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 389–400. Springer-Verlag, 1996.

[ENS91] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian.
Complexity, decidability and undecidability results for domain-independent planning: A detailed analysis.
Technical Report CS-TR-2797, University of Maryland, Computer Science Department, 1991.

[Gef00] Héctor Geffner.
Functional STRIPS: a more flexible language for planning and problem solving.
In *Logic-based Artificial Intelligence*, pages 187–209. Springer-Verlag, 2000.

[GKC13] Sicun Gao, Soonho Kong, and Edmund M. Clarke.
dReal: An SMT solver for nonlinear theories over the reals.
In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24*, volume 7898 of *Lecture Notes in Computer Science*, pages 208–214. Springer-Verlag, 2013.

[GPM89] Carlos E. García, David M. Prett, and Manfred Morari.
Model predictive control: Theory and practice – a survey.
Automatica, 25(3):335–348, 1989.

[GW83] Hana Galperin and Avi Wigderson.
Succinct representations of graphs.
Information and Control, 56:183–198, 1983.
(See [Loz88] for a correction.).

[Hag88] J.N. Hagstrom.
Computational complexity of PERT problems.
Networks, 18(2):139–147, 1988.

References III

[HJ00] Patrik Haslum and Peter Jonsson.
Some results on the complexity of planning with incomplete information.
In *Recent Advances in AI Planning. 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999. Proceedings*, number 1809 in Lecture Notes in Artificial Intelligence, pages 308–318. Springer-Verlag, 2000.

[HKPV95] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya.
What's decidable about hybrid automata?
In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 373–382, 1995.

[Kar72] R.M. Karp.
Reducibility among combinatorial problems.
In R.E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KS92] Henry Kautz and Bart Selman.
Planning as satisfiability.
In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363. John Wiley & Sons, 1992.

[LB90] Antonio Lozano and José L. Balcázar.
The complexity of graph problems for succinctly represented graphs.
In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG'89*, number 411 in Lecture Notes in Computer Science, pages 277–286. Springer-Verlag, 1990.

[Lit97] Michael L. Littman.
Probabilistic propositional planning: Representations and complexity.
In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pages 748–754. AAAI Press, 1997.

[LLS+99] Christopher Lusena, Tong Li, Shelia Sittinger, Chris Wells, and Judy Goldsmith.
My brain is full: When more memory helps.
In *Uncertainty in Artificial Intelligence, Proceedings of the Fifteenth Conference (UAI-99)*, pages 374–381. Morgan Kaufmann Publishers, 1999.

[Loz88] Antonio Lozano.
NP-hardness of succinct representations of graphs.
Bulletin of the European Association for Theoretical Computer Science, 35:158–163, June 1988.

[MGLA00] Martin Mundhenk, Judy Goldsmith, Christopher Lusena, and Eric Allender.
Complexity of finite-horizon Markov decision process problems.
Journal of the ACM, 47(4):681–720, 2000.

References IV

[MHC03] Omid Madani, Steve Hanks, and Anne Condon.
On the undecidability of probabilistic planning and related stochastic optimization problems.
Artificial Intelligence, 147(1–2):5–34, 2003.

[MKKC99] Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra.
Solving POMDPs by searching the space of finite policies.
In *Uncertainty in Artificial Intelligence, Proceedings of the Fifteenth Conference (UAI-99)*, pages 417–426. Morgan Kaufmann Publishers, 1999.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
Chaff: engineering an efficient SAT solver.
In *Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC'01)*, pages 530–535. ACM Press, 2001.

[MSS99] João P. Marques-Silva and Karem A. Sakallah.
GRASP: a search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.

[Pap94] Christos H. Papadimitriou.
Computational Complexity.
Addison-Wesley Publishing Company, 1994.

[Paz71] Azaria Paz.
Introduction to Probabilistic Automata.
Academic Press, 1971.

[PC07] André Platzer and Edmund M. Clarke.
The image computation problem in hybrid systems model checking.
In Alberto Bemporad, Antonio Bicchi, and Giorgio Buttazzo, editors, *Hybrid Systems: Computation and Control*, volume 4416 of *Lecture Notes in Computer Science*, pages 473–486. Springer-Verlag, 2007.

[PT87] Christos H. Papadimitriou and John N. Tsitsiklis.
The complexity of Markov decision processes.
Mathematics of Operations Research, 12(3), 1987.

[Rin99] Jussi Rintanen.
Constructing conditional plans by a theorem-prover.
Journal of Artificial Intelligence Research, 10:323–352, 1999.

References V

[Rin01] Jussi Rintanen.
Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formulae.
In R. Nieuwenhuis and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence and Reasoning. 8th International Conference, LPAR 2001, Havana, Cuba, December 3–7, 2001. Proceedings*, number 2250 in Lecture Notes in Artificial Intelligence, pages 362–376. Springer-Verlag, 2001.

[Rin04] Jussi Rintanen.
Complexity of planning with partial observability.
In *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 345–354. AAAI Press, 2004.

[Rin06] Jussi Rintanen.
Unified definition of heuristics for classical planning.
In *ECAI 2006. Proceedings of the 17th European Conference on Artificial Intelligence*, pages 600–604. IOS Press, 2006.

[Rin07a] Jussi Rintanen.
Asymptotically optimal encodings of conformant planning in QBF.
In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1045–1050. AAAI Press, 2007.

[Rin07b] Jussi Rintanen.
Complexity of concurrent temporal planning.
In *ICAPS 2007. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 280–287. AAAI Press, 2007.

[Rin08] Jussi Rintanen.
Regression for classical and nondeterministic planning.
In *ECAI 2008. Proceedings of the 18th European Conference on Artificial Intelligence*, pages 568–571. IOS Press, 2008.

[Rin09] Jussi Rintanen.
Planning and SAT.
In *Handbook of Satisfiability*, number 185 in Frontiers in Artificial Intelligence and Applications, pages 483–504. IOS Press, 2009.

[Rin13] Jussi Rintanen.
Scheduling with contingent resources and tasks.
In *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS 2013*, pages 189–196. AAAI Press, 2013.

[Sto76] L. J. Stockmeyer.
The polynomial-time hierarchy.
Theoretical Computer Science, 3(1):1–22, 1976.

References VI

[Tur02] Hudson Turner.
Polynomial-length planning spans the polynomial hierarchy.
In *Logics in Artificial Intelligence, European Conference, JELIA 2002*, number 2424 in Lecture Notes in Computer Science, pages 111–124. Springer-Verlag, 2002.

[YMH98] Hui Ye, Anthony N. Michel, and Ling Hou.
Stability theory for hybrid dynamical systems.
IEEE Transactions on Automatic Control, 43(4):461–474, 1998.