Artificial Intelligence

12. Planning, Part I: Framework

How to Describe Arbitrary Search Problems

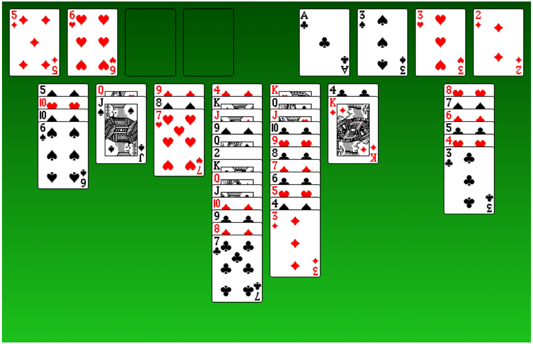
Jörg Hoffmann

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Online (Summer) Term 2020

Agenda

Reminder: Discrete Search Problems (Chapters 1 and 2)



- States: Card positions (*position_Jspades=Qhearts*).
- Actions: Card moves (*move_Jspades_Qhearts_freecell4*).
- Initial state: Start configuration.
- Goal states: All cards "home".
- Solution: Card moves solving this game.

Planning

Ambition:

Write one program that can solve all discrete search problems.

Problem Descriptions

- The **blackbox description** of a problem Π is an API (a programming interface) providing functionality allowing to construct the state space: InitialState(), GoalTest(s), . . .

  $\rightarrow$ "Specifying the problem" $=$ programming the API.

- The **declarative description** of Π comes in a problem description language. This allows to implement the API, and much more.

  $\rightarrow$ "Specifying the problem" $=$ writing a problem description.

$\rightarrow$ Here, "problem description language" $=$ planning language.

"Planning Language"?

How does a planning language describe a problem?

- A *logical description* of the possible states (vs. Blackbox: data structures). E.g.: predicate $Eq(.,.)$.
- A *logical description* of the initial state I (vs. data structures). E.g.: $Eq(x, 1)$.
- A *logical description* of the goal condition G (vs. a goal-test function). E.g.: $Eq(x, 2)$.
- A *logical description* of the set A of actions in terms of preconditions and effects (vs. functions returning applicable actions and successor states).
  E.g.: "increment x: pre $Eq(x, 1)$, eff $Eq(x, 2) \wedge \neg Eq(x, 1)$".

$\rightarrow$ Solution (plan) $=$ sequence of actions from A, transforming I into a state that satisfies G. E.g.: "increment x".

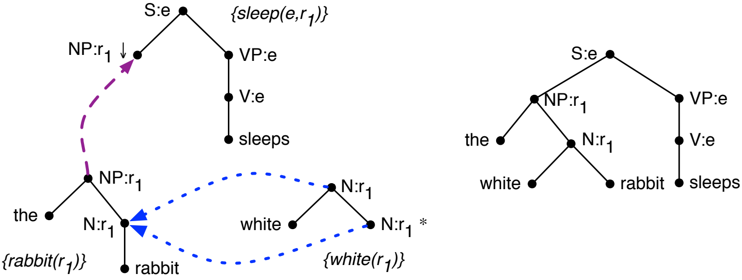
"Planning Language"?

Disclaimer:

→ Planning languages go way beyond discrete search problems. There are variants for partially observable, stochastic, dynamic, continuous, and multi-agent settings.

- We focus on discrete search problems for simplicity (combined with practical relevance).
- For a comprehensive overview, see [Ghallab *et al.* (2004)].

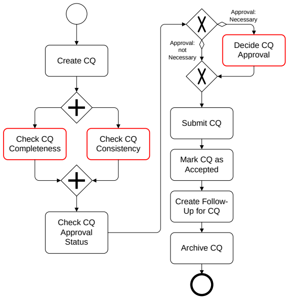
Planning: Language Generation (Project w/ CoLi Dept.)



- **Input:** Tree-adjoining grammar, intended meaning.
- **Output:** Sentence expressing that meaning.

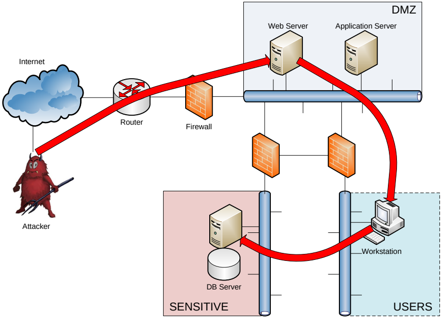
Planning: Business Process Templates at SAP

Action name	precondition	effect
Check CQ Completeness	CQ.archiving:notArchived	CQ.completeness:complete OR CQ.completeness:notComplete
Check CQ Consistency	CQ.archiving:notArchived	CQ.consistency:consistent OR CQ.consistency:notConsistent
Check CQ Approval Status	CQ.archiving:notArchived AND CQ.approval:notChecked AND CQ.completeness:complete AND CQ.consistency:consistent	CQ.approval:necessary OR CQ.approval:notNecessary
Decide CQ Approval	CQ.archiving:notArchived AND CQ.approval:necessary	CQ.approval:granted OR CQ.approval:notGranted
Submit CQ	CQ.archiving:notArchived AND (CQ.approval:notNecessary OR CQ.approval:granted)	CQ.submission:submitted
Mark CQ as Accepted	CQ.archiving:notArchived AND CQ.submission:submitted	CQ.acceptance:accepted
Create Follow-Up for CQ	CQ.archiving:notArchived AND CQ.acceptance:accepted	CQ.followUp:documentCreated
Archive CQ	CQ.archiving:notArchived	CQ.archiving:archived



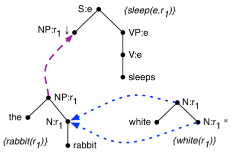
- **Input:** SAP-scale model of behavior of activities on Business Objects, process endpoint.
- **Output:** Process template leading to this point.

Planning: Security Testing (Project w/ CISPA)

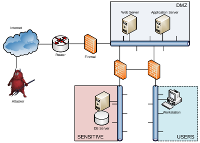


- **Input:** Network configuration, location of sensible data.
- **Output:** Sequence of exploits giving access to that data.

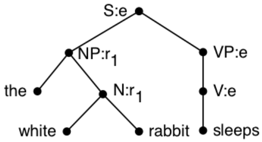
Planning!

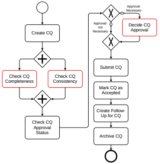


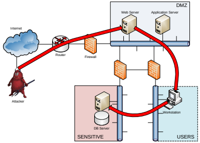
Action name	precondition	effect
Check CQ Completeness	CQ.archiving:notArchived	CQ.completeness:complete OR CQ.completeness:notComplete
Check CQ Consistency	CQ.archiving:notArchived	CQ.consistency:consistent OR CQ.consistency:notConsistent
Check CQ Approval Status	CQ.archiving:notArchived AND CQ.approval:notChecked AND CQ.completeness:complete AND CQ.consistency:consistent	CQ.approval:necessary OR CQ.approval:notNecessary
Decide CQ Approval	CQ.archiving:notArchived AND CQ.approval:necessary	CQ.approval:granted OR CQ.approval:notGranted
Submit CQ	CQ.archiving:notArchived AND (CQ.approval:notNecessary OR CQ.approval:granted)	CQ.submission:submitted
Mark CQ as Accepted	CQ.archiving:notArchived AND CQ.submission:submitted	CQ.acceptance:accepted
Create Follow-Up for CQ	CQ.archiving:notArchived AND CQ.acceptance:accepted	CQ.followUp:documentCreated
Archive CQ	CQ.archiving:notArchived	CQ.archiving:archived



Planning Domain Definition Language (PDDL) $\mapsto$ Planning System







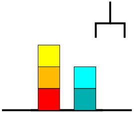
Planning: Pros and Cons

- **Powerful:** In some applications, generality is absolutely necessary. (E.g. SAP)
- **Quick:** Rapid prototyping: 10s lines of problem description vs. 1000s lines of C++ code. (E.g. language generation)
- **Flexible:** Adapt/maintain *the description*. (E.g. network security)
- **Intelligent:** Determines automatically how to solve a complex problem effectively! (The ultimate goal, no?!)

- **Efficiency loss:** Domain-specific knowledge/engineering usually is key to performance . . .

How to make fully automatic algorithms effective?

The Problem



- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

blocks	states	blocks	states
1	1	9	4596553
2	3	10	58941091
3	13	11	824073141
4	73	12	12470162233
5	501	13	202976401213
6	4051	14	3535017524403
7	37633	15	65573803186921
8	394353	16	1290434218669921

$\rightarrow$ State spaces typically are huge (not only for Go and Chess).

Algorithmic Problems in Planning

Satisficing Planning

Input: A planning task Π.
Output: A plan for Π, or "unsolvable" if no plan for Π exists.

Optimal Planning

Input: A planning task Π.
Output: An *optimal* plan for Π, or "unsolvable" if no plan for Π exists.

$\rightarrow$ The techniques successful for either one of these are almost disjoint. And satisficing planning is *much* more effective in practice.

$\rightarrow$ Programs solving these problems are called (optimal) planners, planning systems, or planning tools.

Our Agenda for This Topic

→ Our treatment of the topic "Planning" consists of Chapters 12 and 13.

- **This Chapter:** Background, planning languages, complexity.

  → Sets up the framework. Computational complexity is essential to distinguish different algorithmic problems, and for the design of heuristic functions (see next).

- **Chapter 13:** How to automatically generate a heuristic function, given planning language input?

  → Focussing on heuristic search as the solution method, this is the main question that needs to be answered.

→ We focus on model-based techniques. The use of neural networks is an active research topic (in my research group among others). It's difficult due to the extremely general nature of planning languages.

Our Agenda for This Chapter

- **The History of Planning:** How did this come about?

  → Gives you some background, and motivates heuristic search.

- **The STRIPS Planning Formalism:** Which concrete planning formalism will we be using?

  → Lays the framework we'll be looking at.

- **The PDDL Language:** What do the input files for off-the-shelf planning software look like?

  → So you can actually play around with such software. (Exercises!)

- **Why Complexity Analysis?** Why do we bother?

  → I'll try to convince you that this is USEFUL.

- **Planning Complexity:** How complex is planning?

  → The price of generality is complexity. Here's what that "price" is, exactly.

In the Beginning . . .

. . . Man invented Robots:

"Planning" as in "the making of plans by an autonomous robot".

In a little more detail:

- Newell and Simon (1963) introduced general problem solving.
- *. . . not much happened (well not much we still speak of today) . . .*
- Early 70s Stanford Research Institute developed a robot.
- They needed a "planning" component taking decisions.
- They took inspiration from general problem solving and theorem proving, and called their algorithm "STRIPS" (see slide 25).

And then:

History of Planning Algorithms

Compilation into Logics/Theorem Proving:

- **Popular when:** Stone Age – 1990.
- **Approach:** *From planning task description, generate PL1 formula φ that is satisfiable iff there exists a plan; use a theorem prover on φ.*
- **Keywords/cites:** Situation calculus, frame problem, ...

Partial-Order Planning:

- **Popular when:** 1990 – 1995.
- **Approach:** *Starting at goal, extend partially ordered set of actions by inserting achievers for open sub-goals, or by adding ordering constraints to avoid conflicts.*
- **Keywords/cites:** UCPOP [Penberthy and Weld (1992)], causal links, flaw-selection strategies, ...

History of Planning Algorithms, ctd.

GraphPlan:

- **Popular when:** 1995 – 2000.
- **Approach:** *In a forward phase, build a layered "planning graph" whose "time steps" capture which pairs of actions can achieve which pairs of facts; in a backward phase, search this graph starting at goals and excluding options proved to not be feasible.*
- **Keywords/cites:** [Blum and Furst (1995, 1997); Koehler *et al.* (1997)], action/fact mutexes, step-optimal plans, ...

Planning as SAT:

- **Popular when:** 1996 – today.
- **Approach:** *From planning task description, generate propositional CNF formula* φ_k *that is satisfiable iff there exists a plan with* k *steps; use a SAT solver on* φ_k*, for different values of* k*.*
- **Keywords/cites:** [Kautz and Selman (1992, 1996); Rintanen *et al.* (2006); Rintanen (2010)], SAT encoding schemes, BlackBox, ...

History of Planning Algorithms, ctd.

Planning as Heuristic Search:

- **Popular when:** 1999 – today.
- **Approach:** Devise a method $\mathcal{R}$ to simplify ("relax") any planning task Π; given Π, solve $\mathcal{R}(\Pi)$ to generate a heuristic function h for informed search.
- **Keywords/cites:** [Bonet and Geffner (1999); Haslum and Geffner (2000); Bonet and Geffner (2001); Hoffmann and Nebel (2001); Edelkamp (2001); Gerevini *et al.* (2003); Helmert (2006); Helmert *et al.* (2007); Helmert and Geffner (2008); Karpas and Domshlak (2009); Helmert and Domshlak (2009); Richter and Westphal (2010); Nissim *et al.* (2011); Katz *et al.* (2012); Keyder *et al.* (2012); Katz *et al.* (2013); Domshlak *et al.* (2015)], critical path heuristics, ignoring delete lists, relaxed plans, landmark heuristics, abstractions, partial delete relaxation, LP heuristics, . . .

The International Planning Competition (IPC)

Competition?

"Run competing planners on a set of benchmarks devised by the IPC organizers. Give awards to the most effective planners."

- 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018
- PDDL [McDermott *et al.* (1998); Fox and Long (2003); Hoffmann and Edelkamp (2005); Gerevini *et al.* (2009)]
- ≈ 70 domains, > 1500 instances, 74 planning systems in 2011
- Optimal track vs. satisficing track
- Various others: uncertainty, learning, . . .

`http://ipc.icaps-conference.org/`

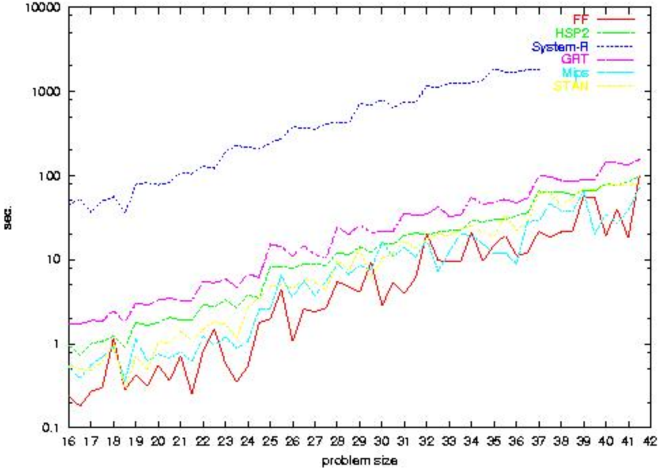
IPC 2000: Competitors

- BlackBox: Compilation to SAT [Kautz and Selman (1999)].
- HSP: Heuristic search [Bonet and Geffner (2001)].
- IPP: GraphPlan variant [Koehler *et al.* (1997)].
- STAN: Heuristic search.
- GRT: Heuristic search.
- Mips: Heuristic search.
- FF: Heuristic search [Hoffmann and Nebel (2001)].
- ... (13 altogether)

IPC 2000: Benchmark Domains

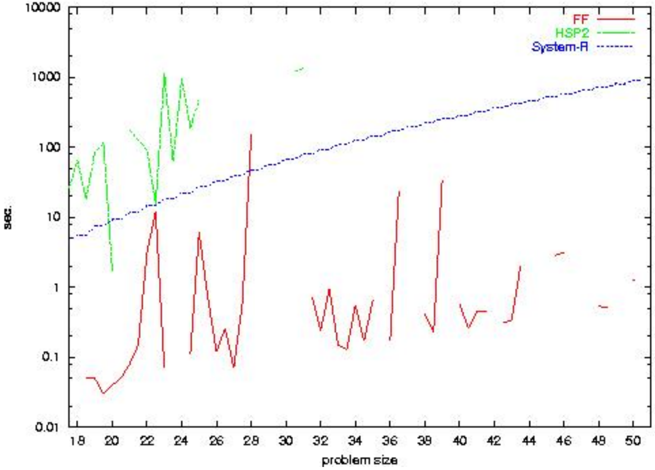
- Blocksworld: Move around blocks on a table (yeah, I know).
- Freecell: The card game.
- Logistics: Transport packages using trucks and airplanes.
- Miconic-ADL: A complex elevator-control problem (see slide 61).
- Schedule: A simple scheduling problem where objects must be processed with various machines.

IPC'00 Results, Fully Automatic Track



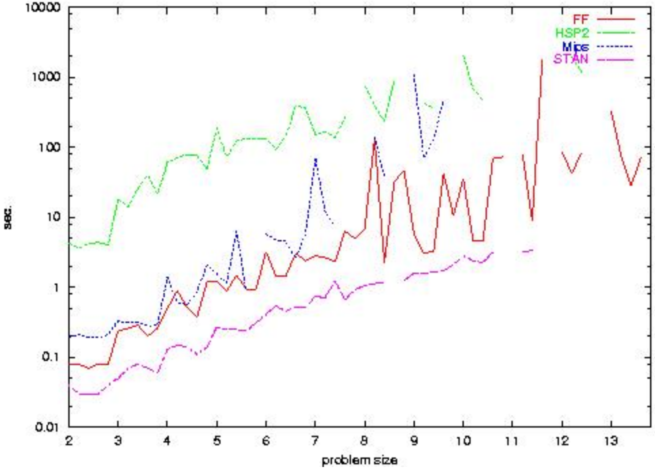
Logistics

IPC'00 Results, Fully Automatic Track



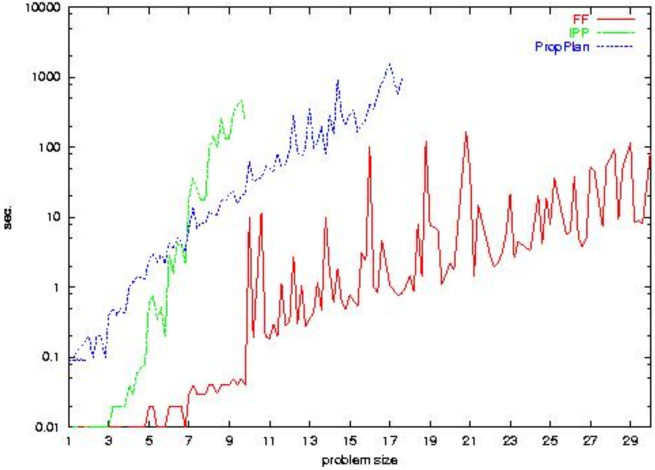
Blocksworld

IPC'00 Results, Fully Automatic Track



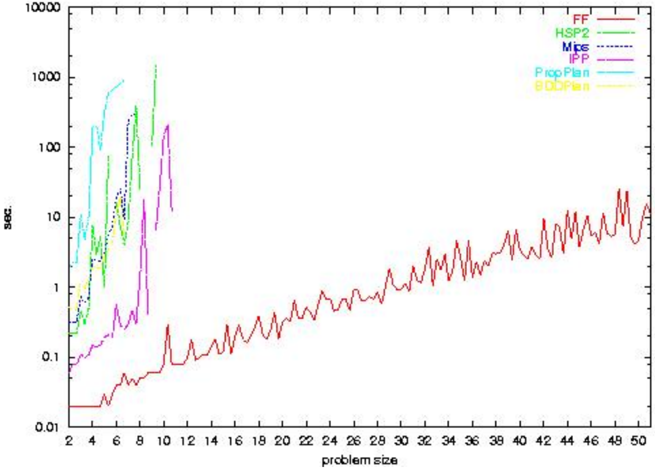
Freecell

IPC'00 Results, Fully Automatic Track



Miconic

IPC'00 Results, Fully Automatic Track



Schedule

And Since Then?

- **IPC 2000:** Winner heuristic search.
- **IPC 2002:** Winner heuristic search.
- **IPC 2004:** Winner satisficing heuristic search; optimal compilation to SAT.
- **IPC 2006:** Winner satisficing heuristic search; optimal compilation to SAT.
- **IPC 2008:** Winner satisficing heuristic search; optimal symbolic search.
- **IPC 2011:** Winner satisficing heuristic search; optimal heuristic search.
- **IPC 2014:** Winner satisficing heuristic search; optimal symbolic search.
- **IPC 2018:** Winner satisficing heuristic search; optimal portfolio/symbolic search/heuristic search.

→ For the rest of this topic, we focus on planning as heuristic search.

→ This is a VERY short summary of the history of the IPC! There are many different categories, and many different awards.

Questionnaire

Question!

If planners x, y both compete in IPC'YY, and x wins, is x "better than" y?

(A): Yes. (B): No.

→ Yes, but only on the IPC'YY benchmarks, and only according to the criteria used for determining a "winner"! On other domains and/or according to other criteria, you may well be better off with the "loser".

→ It's complicated. Over-simplification is dangerous. (But, of course, nevertheless is being done all the time).

"STRIPS" Planning

- STRIPS = Stanford Research Institute Problem Solver.

STRIPS is the simplest possible (reasonably expressive) logics-based planning language.

- STRIPS has only Boolean variables: propositional logic atoms.
- Its preconditions/effects/goals are as canonical as imaginable:
  - Preconditions, goals: conjunctions of positive atoms.
  - Effects: conjunctions of literals (positive or negated atoms).
- We use the common set-based notation for this simple formalism.
- I'll outline some extensions beyond STRIPS later on, when we discuss PDDL.

→ Historical note: STRIPS [Fikes and Nilsson (1971)] was originally a planner (cf. slide 14), whose language actually wasn't quite that simple.

STRIPS Planning: Syntax

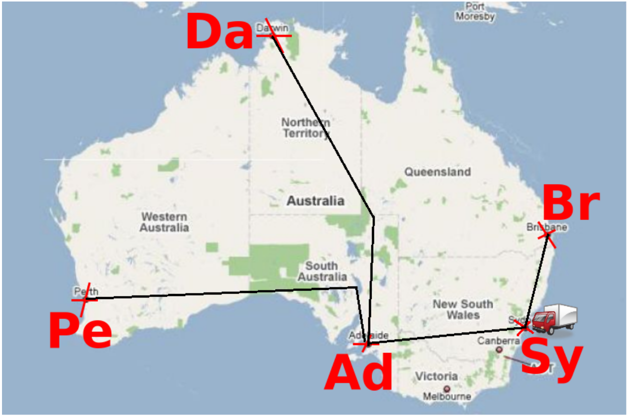
Definition (STRIPS Planning Task). *A STRIPS planning task, short planning task, is a 4-tuple $\Pi = (P, A, I, G)$ where:*

- *P is a finite set of facts (aka propositions).*
- *A is a finite set of actions; each $a \in A$ is a triple $a = (pre_a, add_a, del_a)$ of subsets of P referred to as the action's precondition, add list, and delete list respectively; we require that $add_a \cap del_a = \emptyset$.*
- *$I \subseteq P$ is the initial state.*
- *$G \subseteq P$ is the goal.*

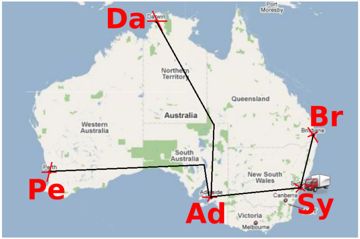
We will often give each action $a \in A$ a name (a string), and identify a with that name.

Note: We assume unit costs for simplicity: every action has cost 1.

"TSP" in Australia



STRIPS Encoding of "TSP"



- Facts P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Initial state I: $\{at(Sydney), visited(Sydney)\}$.
- Goal G:
  $\{at(Sydney)\} \cup \{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  Precondition pre_a: $\{at(x)\}$.
  Add list add_a: $\{at(y), visited(y)\}$.
  Delete list del_a: $\{at(x)\}$.
- Plan: $\langle drive(Sydney, Brisbane), drive(Brisbane, Sydney), drive(Sydney, Adelaide), drive(Adelaide, Perth), drive(Perth, Adelaide), drive(Adelaide, Darwin), drive(Darwin, Adelaide), drive(Adelaide, Sydney)\rangle$.

STRIPS Planning: Semantics

Definition (STRIPS State Space). *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. The state space of Π is $\Theta_\Pi = (S, A, T, I, S^G)$ where:*

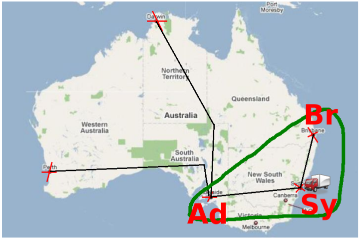
- *The states (also world states) $S = 2^P$ are the subsets of P.*
- *A is Π's action set.*
- *The transitions are $T = \{s \xrightarrow{a} s' \mid pre_a \subseteq s, s' = appl(s, a)\}$.*
  If $pre_a \subseteq s$, then a is applicable in s and $appl(s, a) := (s \cup add_a) \setminus del_a$. If $pre_a \not\subseteq s$, then $appl(s, a)$ is undefined.
- *I is Π's initial state.*
- *The goal states $S^G = \{s \in S \mid G \subseteq s\}$ are those that satisfy Π's goal.*

An (optimal) plan for $s \in S$ is an (optimal) solution for s in Θ_Π, i.e., a path from s to some $s' \in S^G$. A solution for I is called a plan for Π. Π is solvable if a plan for Π exists.

For $\vec{a} = \langle a_1, \dots, a_n \rangle$, $appl(s, \vec{a}) := appl(\dots appl(appl(s, a_1), a_2) \dots, a_n)$ if each a_i is applicable in the respective state; else, $appl(s, \vec{a})$ is undefined.

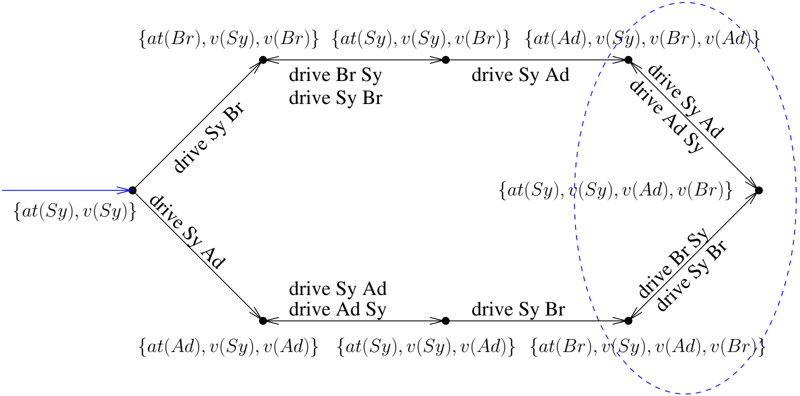
Note: This is exactly like the state spaces of **Chapter 1**, without a cost function. Solutions are defined as before (paths from I to a state in S^G).

STRIPS Encoding of Simplified "TSP"



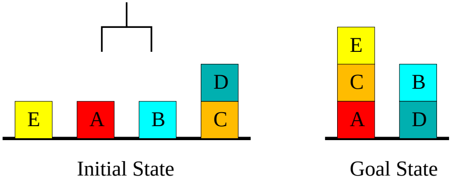
- Facts P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$.
- Initial state I: $\{at(Sydney), visited(Sydney)\}$.
- Goal G: $\{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$. (Note: no "$at(Sydney)$".)
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  Precondition pre_a: $\{at(x)\}$.
  Add list add_a: $\{at(y), visited(y)\}$.
  Delete list del_a: $\{at(x)\}$.

STRIPS Encoding of Simplified "TSP": State Space



→ Is this actually the state space? No, only the reachable part. E.g., Θ_Π also includes the states $\{v(Sy)\}$ and $\{at(Sy), at(Br)\}$.

(Oh no it's) The Blocksworld



- Facts: $on(x, y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- Initial state: $\{onTable(E), clear(E), \dots, onTable(C), on(D, C), clear(D), armEmpty()\}$.
- Goal: $\{on(E, C), on(C, A), on(B, D)\}$.
- Actions: $stack(x, y)$, $unstack(x, y)$, $putdown(x)$, $pickup(x)$.
- $stack(x, y)$? $pre: \{holding(x), clear(y)\}$
  $add: \{on(x, y), armEmpty()\}$
  $del: \{holding(x), clear(y)\}$.

Questionnaire

Question!

Which are correct encodings (part of <u>some</u> correct overall encoding) of the STRIPS Blocksworld $pickup(x)$ action schema?

(A): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x)\})$.

(B): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{armEmpty()\})$.

(C): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty(), clear(x)\})$.

(D): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty()\})$.

→ (A): No, must delete $armEmpty()$. (B): No, must delete $onTable(x)$. (C), (D): Both yes: We can, but don't have to, encode the *single-arm* Blocksworld so that the block currently in the hand is not clear. (For (C), $stack(x, y)$ and $putdown(x)$ need to add $clear(x)$, so the encoding on the previous slide does not work.)

PDDL History

Planning Domain Description Language:

- A description language for planning in the STRIPS formalism and various extensions.
- Used in the International Planning Competition (IPC).

- 1998: PDDL [McDermott *et al.* (1998)].
- 2000: "PDDL subset for the 2000 competition" [Bacchus (2000)].
- 2002: PDDL2.1, Levels 1-3 [Fox and Long (2003)].
- 2004: PDDL2.2 [Hoffmann and Edelkamp (2005)].
- 2006: PDDL3 [Gerevini *et al.* (2009)].

PDDL Quick Facts

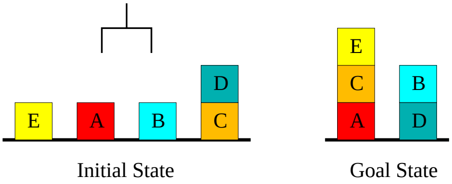
PDDL is not a propositional language:

- Representation is "lifted", using variables ranging over a universe of objects like in predicate logic. The universe is finite however.
- Predicates as in predicate logic.
- Action schemas parameterized by objects.

A PDDL planning task comes in two pieces:

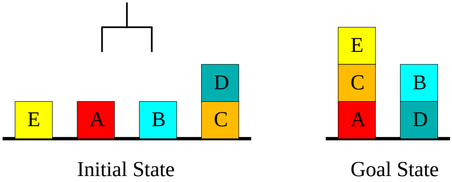
- The domain file and the problem file.
- The problem file gives the objects, the initial state, and the goal state.
- The domain file gives the predicates and the operators; each benchmark domain has *one* domain file.

The Blocksworld in PDDL: Domain File




```
(define (domain blocksworld)
(:predicates (clear ?x) (holding ?x) (on ?x ?y)
             (on-table ?x) (arm-empty))
(:action stack
 :parameters (?x ?y)
 :precondition (and (clear ?y) (holding ?x))
 :effect (and (arm-empty) (on ?x ?y)
              (not (clear ?y)) (not (holding ?x)))
)
. . .
```

The Blocksworld in PDDL: Problem File




```
(define (problem bw-abcde)
(:domain blocksworld)
(:objects a b c d e)
(:init (on-table a) (clear a)
       (on-table b) (clear b)
       (on-table e) (clear e)
       (on-table c) (on d c) (clear d)
       (arm-empty))
(:goal (and (on e c) (on c a) (on b d))))
```

PDDL in 1998

STRIPS + ADL (Action Description Language):

- Arbitrary first-order logic formulas in action preconditions and the goal.
- Conditional effects, i.e., effects that occur only if their separate effect condition holds.

ADL is a real headache to implement:

- The systems that do handle ADL *compile* it down to simpler formats [Gazen and Knoblock (1997)]. (Typically, STRIPS with conditional effects.)
- Example FF: 7000 C lines for compilation, 2000 lines core planner.

Miconic-ADL “Stop” Action Schema in PDDL

```
(:action stop
:parameters (?f - floor)
:precondition (and (lift-at ?f)
                   (imply
                   (exists
                   (?p - conflict-A)
                   (or (and (not (served ?p))
                            (origin ?p ?f))
                   (and (boarded ?p)
                            (not (destin ?p ?f)))))
                   (forall
                   (?q - conflict-B)
                   (and (or (destin ?q ?f)
                            (not (boarded ?q)))
                   (or (served ?q)
                            (not (origin ?q ?f))))))
                   (imply (exists
                   (?p - conflict-B)
                   (or (and (not (served ?p))
                            (origin ?p ?f))
                   (and (boarded ?p)
                            (not (destin ?p ?f)))))
                   (forall
                   (?q - conflict-A)
                   (and (or (destin ?q ?f)
                            (not (boarded ?q)))
                   (or (served ?q)
                            (not (origin ?q ?f))))))
                   (imply
                   (exists
                   (?p - never-alone)
                   (or (and (origin ?p ?f)
```

PDDL in 2000

Fahiem Bacchus selected a subset of the ADL subset of McDermott's PDDL for the 2000 competition.

(Actually, he first designed a whole new language all of his own, but the IPC'00 organizing committee didn't like it.)

PDDL in 2002

Maria Fox and Derek Long promoted **numeric and temporal planning:**

- **PDDL 2.1 level 1**: Bacchus's PDDL.
- **PDDL 2.1 level 2**: Level 1 extended with numeric state variables. Comparisons between numeric expressions are allowed as logical atoms ("$fuel(v) \geq dist(x, y) * consumption(v)$"). Effects can assign the value of an expression to a numeric variable ("$fuel(v) := fuel(v) - dist(x, y) * consumption(v)$").
- **PDDL 2.1 level 3**: Level 2 extended with action durations. Actions take an amount of time given by the value of a numeric expression ("$dist(x, y)/speed(v)$"). Conditions and effects are evaluated at either the start or the end of the action, and several actions can be executed in parallel.

PDDL After 2002

For IPC'04, Stefan Edelkamp and I deemed PDDL2.1 to be challenging enough, so made only two small language extensions for **PDDL 2.2**: Derived Predicates (e.g., flow of current in an electricity network) and Timed Initial Literals (e.g., sunrise and sunset, shop closing times).

Gerevini & Long thought that PDDL2.2 is still not enough, and extended it with various complex notions of soft goals and preferences to obtain **PDDL 3**.

→ The good news (from my perspective): Since 2008, PDDL has remained largely stable.

→ Having said that: There's variants for partial observability, stochastic effects, uncertain initial states, multi-agents, ...

Questionnaire

Question!

What is PDDL good for?

(A): Nothing.
(B): Free beer.
(C): Those AI planning guys.
(D): Being lazy at work.

→ (A): Nah, it's definitely good for *something* (see remaining answers).

→ (B): Generally, no. Sometimes, yes: PDDL is needed for the IPC, and if you win the IPC you get price money (= free beer).

→ (C): Yep. (When I started in this area, every system had its own language, so running experiments felt a lot like "Lost in Translation".)

→ (D): Yep. You can be a busy bee, programming a solver yourself. Or you can be lazy and just write the PDDL. (I think I said that before . . .)

Why Complexity Analysis?

Why? Why?

Two very good reasons:

1. It saves you from spending lots of time trying to invent algorithms that do not exist.
2. Killer app in planning: tractable fragments for heuristic functions.

   → Identify special cases that can be solved in polynomial time.

   → Relax the input into the special case to obtain a heuristic function! (→ **Chapter 13**)

→ I'll next briefly remind you of the basic concepts in complexity theory, then I'll illustrate both 1 and 2 with an example. Afterwards we'll have a look at the complexity of the main decision problems in STRIPS planning.

Reminder (?): **NP** and **PSPACE**

Def Turing machine: Works on a tape consisting of tape cells, across which its R/W head moves. The machine has internal states. There are transition rules specifying, given the current cell content and internal state, what the subsequent internal state will be, how what the R/W head does (write a symbol and/or move). Some internal states are accepting.

Def NP: Decision problems for which there exists a *non-deterministic* Turing machine that runs in *time* polynomial in the size of its input. Accepts if *at least one* of the possible runs accepts.

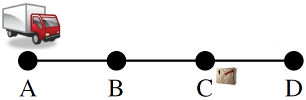
Def PSPACE: Decision problems for which there exists a *deterministic* Turing machine that runs in *space* polynomial in the size of its input.

Relation: Non-deterministic polynomial space can be simulated in deterministic polynomial space. Thus **PSPACE** = **NPSPACE**, and hence (trivially) **NP** $\subseteq$ **PSPACE**. It is commonly believed that **NP** $\not\supseteq$ **PSPACE** (similar to **P** $\subseteq$ **NP**).

$\rightarrow$ For comprehensive details, please see a text book. My personal favorite is [Garey and Johnson (1979)]. (On the first 3 pages, they explain why knowing about **NP**-hardness will help you talk to your future boss.)

The "Only-Adds" Relaxation

Example: "Logistics"



- Facts P: $\{truck(x) \mid x \in \{A, B, C, D\}\} \cup$ $\{pack(x) \mid x \in \{A, B, C, D, T\}\}$.
- Initial state I: $\{truck(A), pack(C)\}$.
- Goal G: $\{truck(A), pack(D)\}$.
- Actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
  - $drive(x, y)$, where x, y have a road: "$truck(x) \Rightarrow truck(y), \neg truck(x)$".
  - $load(x)$: "$truck(x), pack(x) \Rightarrow pack(T), \neg pack(x)$".
  - $unload(x)$: "$truck(x), pack(T) \Rightarrow pack(x), \neg pack(T)$".

Only-Adds Relaxation: Drop the preconditions and deletes.

"$drive(x, y)$: $\Rightarrow truck(y)$"; "$load(x)$: $\Rightarrow pack(T)$"; "$unload(x)$: $\Rightarrow pack(x)$".

$\rightarrow$ Say we want to use this for generating a heuristic function: We solve the relaxed problem on state s to obtain $h(s)$ (details see next chapter).

Solving Only-Adds STRIPS Tasks

Our problem:

- Given STRIPS task $\Pi = (P, A, I, G)$.
- Find action sequence $\vec{a}$ leading from I to a state that contains G, when pretending that preconditions and deletes are empty.

Solution 1: (simplest possible approach)

$\vec{a} := \langle\rangle$
while $G \neq \emptyset$ **do**
  select $a \in A$
  $G := G \setminus add_a$
  $\vec{a} := \vec{a} \circ \langle a \rangle$; $A := A \setminus \{a\}$
endwhile
return $h := |\vec{a}|$

$\rightarrow$ Is this h admissible? No. Admissibility is only guaranteed if we find a *shortest possible* $\vec{a}$; else, $\vec{a}$ might be longer than a plan for Π itself. Selecting an arbitrary action each time, $\vec{a}$ may be longer than needed.

Solving Only-Adds STRIPS Tasks, ctd.

So, what about this?

$\vec{a} := \langle\rangle$
while $G \neq \emptyset$ **do**
select $a \in A$ s.t. $|add_a|$ is maximal
$G := G \setminus add_a$
$\vec{a} := \vec{a} \circ \langle a \rangle$; $A := A \setminus \{a\}$
endwhile
return $h := |\vec{a}|$

→ h admissible? No, large add_a doesn't help if the intersection with G is small.

And this?

$\vec{a} := \langle\rangle$
while $G \neq \emptyset$ **do**
select $a \in A$ s.t. $|add_a \cap G|$ is maximal
$G := G \setminus add_a$
$\vec{a} := \vec{a} \circ \langle a \rangle$; $A := A \setminus \{a\}$
endwhile
return $h := |\vec{a}|$

→ h admissible? Still no. Example: $G = \{A, B, C, D, E, F\}$;
$add_{a_1} = \{A, C, E\}$; $add_{a_2} = \{A, B\}$; $add_{a_3} = \{C, D\}$; $add_{a_4} = \{E, F\}$.

Solving Only-Adds STRIPS Tasks, ctd.

From [Garey and Johnson (1979)]:

222 NP-COMPLETE PROBLEMS

[SP5] **MINIMUM COVER**

INSTANCE: Collection C of subsets of a finite set S, positive integer $K \leqslant |C|$.
QUESTION: Does C contain a cover for S of size K or less, i.e., a subset $C' \subseteq C$ with $|C'| \leqslant K$ such that every element of S belongs to at least one member of C'?

Reference: [Karp, 1972]. Transformation from X3C.
Comment: Remains NP-complete even if all $c \in C$ have $|c| \leqslant 3$. Solvable in polynomial time by matching techniques if all $c \in C$ have $|c| \leqslant 2$.

So what?

- Given STRIPS task $\Pi = (P, A, I, G)$.
- Find $\vec{a}$ of length $\leq K$ leading from I to a state that contains G, when pretending that preconditions and deletes are empty.

$\rightarrow$ $\vec{a}$ leads to G $\Leftrightarrow$ $\bigcup_{a \in \vec{a}} add_a \supseteq G$ $\Leftrightarrow$ the add lists in $\vec{a}$ cover G. QED.

Questionnaire

Assume: In 2 years from now, you have finished your studies and are working in an industry job. Your boss Mr. X gives you a problem and says "Solve It!". By which he means, "write a program that solves it efficiently".

Question!

Could knowing about NP-hardness help?

(A): Yes. (B): No.

→ Yes! Assume further that, after trying in vain for 4 weeks, you got the next meeting with Mr. X. Do you want to say "Um, sorry, but I couldn't find an efficient solution, please don't fire me"?

Or would you rather say "Look, I didn't find an efficient solution. But neither could all the Turing-award winners out there put together, because the problem is **NP**-hard"? (Copyright [Garey and Johnson (1979)])

Reminder: Algorithmic Problems in Planning

Satisficing Planning

Input: A planning task Π.
Output: A plan for Π, or "unsolvable" if no plan for Π exists.

Optimal Planning

Input: A planning task Π.
Output: An *optimal* plan for Π, or "unsolvable" if no plan for Π exists.

Decision Problems in (STRIPS) Planning

Definition (PlanEx). *By PlanEx, we denote the problem of deciding, given a STRIPS planning task Π, whether or not there exists a plan for Π.*

$\rightarrow$ Corresponds to satisficing planning.

Definition (PlanLen). *By PlanLen, we denote the problem of deciding, given a STRIPS planning task Π and an integer K, whether or not there exists a plan for Π of length at most K.*

$\rightarrow$ Corresponds to optimal planning.

Definition (PolyPlanLen). *By PolyPlanLen, we denote the problem of deciding, given a STRIPS planning task Π and an integer K bounded by a polynomial in the size of Π, whether or not there exists a plan for Π of length at most K.*

$\rightarrow$ Corresponds to optimal planning with "small" plans. Example of a planning domain with exponentially long plans? Towers of Hanoi.

Complexity of PlanEx [Bylander (1994)]

Lemma. PlanEx is **PSPACE**-hard.

$\rightarrow$ "*At least* as hard as any other problem contained in **PSPACE**."

Proof Sketch. Given a Turing machine with space bounded by polynomial $p(|w|)$, we can in polynomial time (in the size of the machine) generate an equivalent STRIPS planning task. Say the possible symbols in tape cells are $x_1, \dots, x_m$ and the internal states are $s_1, \dots, s_n$, accepting state s_{acc}.

- The contents of the tape cells: $in(1, x_1), \dots, in(p(|w|), x_1), \dots, in(1, x_m), \dots, in(p(|w|), x_m)$.
- The position of the R/W head: $at(1), \dots, at(p(|w|))$.
- The internal state of the machine: $state(s_1), \dots, state(s_n)$.
- Transitions rules $\mapsto$ STRIPS actions; accepting state $\mapsto$ STRIPS goal $\{state(s_{acc})\}$; initial state obvious.
- This reduction to STRIPS runs in polynomial time because we need only polynomially many facts.

Complexity of PlanEx, ctd. [Bylander (1994)]

Lemma. *PlanEx is a member of* **PSPACE***.*

$\rightarrow$ "*At most* as hard as any other problem contained in **PSPACE**."

Proof. Because **PSPACE** $=$ **NPSPACE**, it suffices to show that PlanEx is a member of **NPSPACE**:

1. $s := I$; $l := 0$;
2. Guess an applicable action a, compute the outcome state s', set $l := l + 1$;
3. If s' contains the goal then succeed;
4. If $l \geq 2^{|P|}$ then fail else goto 2;

$\rightarrow$ Remembering the actual action *sequence* would take exponential space in case of exponentially long plans (cf. slide 55). But, to decide PlanEx, we only need to remember its length.

Theorem (Complexity of PlanEx). *PlanEx is* **PSPACE***-complete.* (Immediate from previous two lemmas)

Complexity of PlanLen [Bylander (1994)]

PlanLen isn't any easier than PlanEx:

Corollary. *PlanLen is* **PSPACE***-complete.*

Proof. Membership: Same as before but failing at $l \geq K$. Hardness? Setting $K := 2^{|P|}$, PlanLen answers PlanEx.

PolyPlanLen is easier than PlanEx:

Theorem. *PolyPlanLen is* **NP***-complete.*

Proof. Membership? Guess K actions and check whether they form a plan. This runs in polynomial time because K is polynomially bounded. Hardness: E.g., by reduction from SAT. (Exercises, perhaps)

$\rightarrow$ Bounding plan length does not help in the general case as we can set the bound to a trivial (exponential) upper bound on plan length. If we restrict plan length to be "short" (polynomial), planning becomes easier.

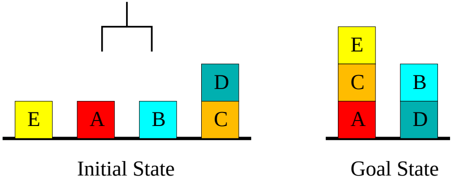
Domain-Specific PlanEx vs. PlanLen . . .

. . . is more interesting than the general case.

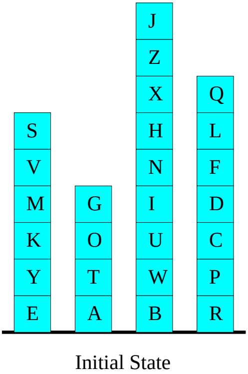
- *In general*, both have the same complexity.
- *Within particular applications*, bounded length plan existence (optimal planning) is often harder than plan existence (satisficing planning).
- This happens in many planning competition benchmark domains: PlanLen is **NP**-complete while PlanEx is in **P**.
- For example: Blocksworld and Logistics.

$\rightarrow$ In practice, optimal planning is (almost) never easy.

The Blocksworld is Hard?

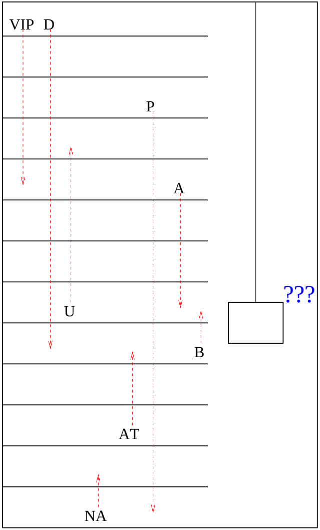


The Blocksworld is Hard!



Miconic-ADL: PlanEx is Hard

- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone; AT: Attendant.
- A, B: Never together in the same elevator (!)
- P: Normal passenger :-)



Summary

- General problem solving attempts to develop solvers that perform well across a large class of problems.
- Planning, as considered here, is a form of general problem solving dedicated to the class of classical search problems. (Actually, we also address inaccessible, stochastic, dynamic, continuous, and multi-agent settings.)
- Heuristic search planning has dominated the International Planning Competition (IPC). We focus on it here.
- STRIPS is the simplest possible, while reasonably expressive, language for our purposes. It uses Boolean variables (facts), and defines actions in terms of precondition, add list, and delete list.
- PDDL is the de-facto standard language for describing planning problems.
- Plan existence (bounded or not) is **PSPACE**-complete to decide for STRIPS. If we bound plans polynomially, we get down to **NP**-completeness.

Reading

- *Chapters 10: Classical Planning and 11: Planning and Acting in the Real World* [Russell and Norvig (2010)].

  Content: Ok as a background read, but not a good introduction to modern planning techniques.

  Chapter 10 gives some background. Some issues are, imho, misrepresented, and it's far from being an up-to-date account. But it's Ok to get some additional intuitions in words different from my own.

  Chapter 11 is useful in our context here because I don't cover any of it. If you're interested in extended/alternative planning paradigms, do read it.

Reading, ctd.

- *Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)* [Hoffmann (2011)].

  Available at:

  `http://fai.cs.uni-saarland.de/hoffmann/papers/ki11.pdf`

  Content: My personal perspective on planning. Excerpt from the abstract:

  The area has long had an affinity towards playful illustrative examples, imprinting it on the mind of many a student as an area concerned with the rearrangement of blocks, and with the order in which to put on socks and shoes (not to mention the disposal of bombs in toilets). Working on the assumption that this "student" is you – the readers in earlier stages of their careers – I herein aim to answer three questions that you surely desired to ask back then already:

  What is it good for? Does it work? Is it interesting to do research in?

References I

Fahiem Bacchus. *Subset of PDDL for the AIPS2000 Planning Competition*. The AIPS-00 Planning Competition Comitee, 2000.

Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. In S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1636–1642, Montreal, Canada, August 1995. Morgan Kaufmann.

Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):279–298, 1997.

Blai Bonet and Héctor Geffner. Planning as heuristic search: New results. In S. Biundo and M. Fox, editors, *Proceedings of the 5th European Conference on Planning (ECP'99)*, pages 60–72. Springer-Verlag, 1999.

Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

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

References II

Carmel Domshlak, Jörg Hoffmann, and Michael Katz. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence*, 221:73–114, 2015.

Stefan Edelkamp. Planning with pattern databases. In A. Cesta and D. Borrajo, editors, *Proceedings of the 6th European Conference on Planning (ECP'01)*, pages 13–24. Springer-Verlag, 2001.

Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

Michael R. Garey and David S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

B. Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In S. Steel and R. Alami, editors, *Proceedings of the 4th European Conference on Planning (ECP'97)*, pages 221–233. Springer-Verlag, 1997.

References III

Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research*, 20:239–290, 2003.

Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

Patrik Haslum and Hector Geffner. Admissible heuristics for optimal planning. In S. Chien, R. Kambhampati, and C. Knoblock, editors, *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 140–149, Breckenridge, CO, 2000. AAAI Press, Menlo Park.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 162–169. AAAI Press, 2009.

References IV

Malte Helmert and Hector Geffner. Unifying the causal graph and additive heuristics. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, editors, *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 140–147. AAAI Press, 2008.

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Mark Boddy, Maria Fox, and Sylvie Thiebaux, editors, *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 176–183, Providence, Rhode Island, USA, 2007. Morgan Kaufmann.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

Jörg Hoffmann and Stefan Edelkamp. The deterministic part of ipc-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

References V

Jörg Hoffmann. Everything you always wanted to know about planning (but were afraid to ask). In Joscha Bach and Stefan Edelkamp, editors, *Proceedings of the 34th Annual German Conference on Artificial Intelligence (KI'11)*, volume 7006 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2011.

Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In Craig Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1728–1733, Pasadena, California, USA, July 2009. Morgan Kaufmann.

Michael Katz, Jörg Hoffmann, and Malte Helmert. How to relax a bisimulation? In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pages 101–109. AAAI Press, 2012.

Michael Katz, Jörg Hoffmann, and Carmel Domshlak. Who said we need to relax *all* variables? In Daniel Borrajo, Simone Fratini, Subbarao Kambhampati, and Angelo Oddi, editors, *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, pages 126–134, Rome, Italy, 2013. AAAI Press.

References VI

Henry A. Kautz and Bart Selman. Planning as satisfiability. In B. Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, Vienna, Austria, August 1992. Wiley.

Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In William J. Clancey and Daniel Weld, editors, *Proceedings of the 13th National Conference of the American Association for Artificial Intelligence (AAAI'96)*, pages 1194–1201, Portland, OR, July 1996. MIT Press.

Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In M. Pollack, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 318–325, Stockholm, Sweden, August 1999. Morgan Kaufmann.

Emil Keyder, Jörg Hoffmann, and Patrik Haslum. Semi-relaxed plan heuristics. In Blai Bonet, Lee McCluskey, José Reinaldo Silva, and Brian Williams, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pages 128–136. AAAI Press, 2012.

References VII

Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. In S. Steel and R. Alami, editors, *Proceedings of the 4th European Conference on Planning (ECP'97)*, pages 273–285. Springer-Verlag, 1997.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee, 1998.

Allen Newell and Herbert Simon. GPS, a program that simulates human thought. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw-Hill, 1963.

Raz Nissim, Jörg Hoffmann, and Malte Helmert. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 1983–1990. AAAI Press/IJCAI, 2011.

References VIII

J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In B. Nebel, W. Swartout, and C. Rich, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*, pages 103–114, Cambridge, MA, October 1992. Morgan Kaufmann.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

Jussi Rintanen. Heuristics for planning with SAT. In *Proceeedings of the 16th International Conference on Principles and Practice of Constraint Programming*, pages 414–428, 2010.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Third Edition)*. Prentice-Hall, Englewood Cliffs, NJ, 2010.