Artificial Intelligence

Planning 1: Planning Tasks

Jendrik Seipp

Linköping University

based on slides by Thomas Keller and Malte Helmert (University of Basel)

Questions?

post feedback and ask questions anonymously at

`https://padlet.com/jendrikseipp/tddc17`

Intended Learning Outcomes

- explain what “automated planning” is
- contrast the PDDL, STRIPS and SAS^+ planning formalisms
- model planning tasks in these formalisms
- explain what a heuristic is and how we can obtain them
- justify why the STRIPS heuristic is not very informative

Introduction

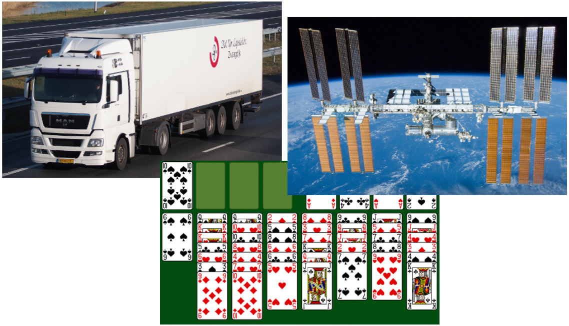
Automated Planning

> "Planning is the art and practice of thinking before acting."
>
> — P. Haslum

⇝ finding plans (sequences of actions)
that lead from an initial state to a goal state

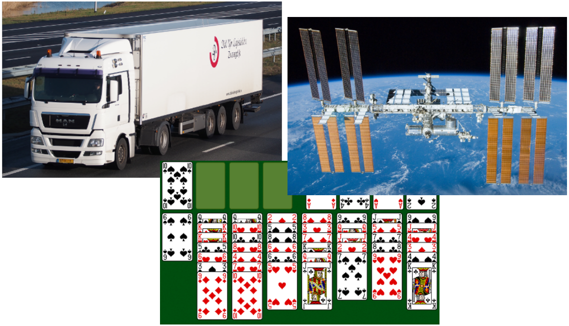
- general approach to solving state-space search problems
- classical planning: static, deterministic, fully observable
- variants (not considered here):
  - probabilistic planning
  - planning under partial observability
  - online planning
  - ...

Motivation



- general: domain-independent
- relevant: Ericsson, Saab, NASA
- declarative: “what?” instead of “how?”

Motivation



- general: domain-independent
- relevant: Ericsson, Saab, NASA
- declarative: “what?” instead of “how?”
- MSc and PhD theses on planning available

Planning: Informally

given:

- state space description in terms of suitable problem description language (planning formalism)

required:

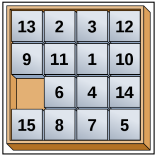
- a plan, i.e., a solution for the described state space (sequence of actions from initial state to goal)
- or a proof that no plan exists

distinguish between

- optimal planning: guarantee that returned plans are optimal, i.e., have minimal overall cost
- suboptimal planning (satisficing): suboptimal plans are allowed

What is New?

we have seen planning tasks before, e.g.:



- as before: we solve these tasks with informed search algorithms like A^* or greedy-best first search
- as before: search is guided by a heuristic
- new: we are now interested in general algorithms, i.e., the developer of the search algorithm does not know the tasks that the algorithm needs to solve

⇝ no problem-specific heuristics!

⇝ input language to model the planning task

Compact Descriptions

State Spaces with Declarative Representations

How do we represent state spaces in the computer?

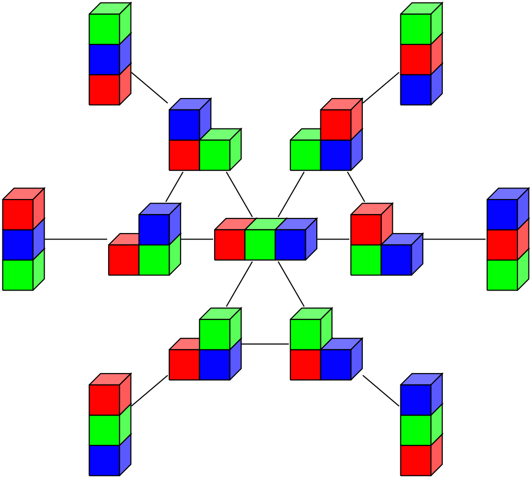
so far, states were black boxes

now, we represent state spaces declaratively:

- compact description of state space as input to algorithms
  $\rightsquigarrow$ state spaces exponentially larger than the input
- algorithms directly operate on compact description

$\rightsquigarrow$ allows automatic reasoning about problem:
reformulation, simplification, abstraction, etc.

Blocks World



Compact Description of State Spaces

How to describe state spaces compactly?

- introduce state variables
- states: assignments to state variables

⇝ e.g., n binary state variables can describe 2^n states

- transitions and goal are compactly described with a logic-based formalism

different variants: different planning formalisms

Three Planning Formalisms

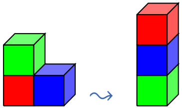
- a description language for planning tasks is called a planning formalism
- we introduce three planning formalisms:
  1. "AIMA-PDDL" (Planning Domain Definition Language as introduced in AIMA)
  2. STRIPS (Stanford Research Institute Problem Solver)
  3. SAS$^+$ (Simplified Action Structures)
- STRIPS and SAS$^+$ are simpler formalisms than PDDL

PDDL

Planning Domain Definition Language

- PDDL is the standard language used to describe planning tasks in practice
- descriptions in (restricted) predicate logic (even more compact than propositional logic)
- support for many "advanced" features like
  - numeric variables
  - temporal semantics
  - stochastic effects
  - ...

PDDL planning task



a first-order PDDL planning task is given by

- a set of predicates: on/2, ontable/1, clear/1
- a set of objects: R, B, G
- a set of action schemata (move, to-table, from-table) with
  - a schematic precondition
  - a schematic effect
  - a cost (optionally)
- an initial state:
  on(G, R) ∧ ontable(R) ∧ ontable(B) ∧ clear(G) ∧ clear(B)
- a goal description: on(R, B) ∧ on(B, G)

Example: Blocks World in PDDL

Emacs: blocksworld-domain.pddl

File Edit Options Buffers Tools Help

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; blocksworld
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain BLOCKS)
  (:requirements :strips)

  (:predicates
    (on ?x ?y)
    (ontable ?x)
    (clear ?x)
  )

  (:action move
    :parameters (?block ?from ?to)
    :precondition
      (and
        (on ?block ?from)
        (clear ?block)
        (clear ?to)
      )
    :effect
      (and
        (on ?block ?to)
        (clear ?from)
        (not (on ?block ?from))
        (not (clear ?to))
      )
  )

  (:action to-table
    :parameters (?block ?from)
```

Emacs: blocksworld-instance.pddl

File Edit Options Buffers Tools Help

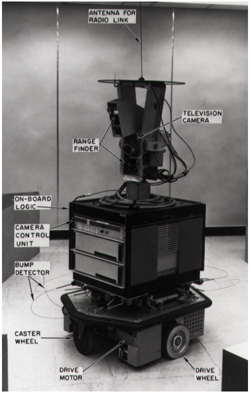
```
(define (problem BLOCKS-3-0)
  (:domain BLOCKS)

  (:objects R B G)

  (:init
    (on G R)
    (ontable R)
    (ontable B)
    (clear G)
    (clear B)
  )

  (:goal
    (and
      (on R B)
      (on B G)
    )
  )
)
```

PDDL Fragments

- even without “advanced” features, PDDL is
  - very expressive
  - but non-trivial to formalize
- there are predefined PDDL fragments
- PDDL as presented in AIMA is also a PDDL fragment

STRIPS

STRIPS



- was developed as input language for Shakey the robot (1971)
- is the simplest commonly used planning formalism
- is a special case of ground AIMA-PDDL where
  - preconditions are restricted to conjunctions over positive literals
  - goals are restricted to conjunctions over positive literals

STRIPS: Basic Concepts

- all state variables in V are binary (true or false)

STRIPS: Basic Concepts

- all state variables in V are binary (true or false)
- states s can be represented in three equivalent ways:
  - as assignments $s : V \longrightarrow \{\mathbf{F}, \mathbf{T}\}$
  - as a conjunction over V (closed world assumption)
  - as sets $s \subseteq V$,
    where s encodes the set of state variables that are true in s

we use the set representation

STRIPS: Basic Concepts

- all state variables in V are binary (true or false)
- states s can be represented in three equivalent ways:
  - as assignments $s : V \longrightarrow \{\mathbf{F}, \mathbf{T}\}$
  - as a conjunction over V (closed world assumption)
  - as sets $s \subseteq V$,
    where s encodes the set of state variables that are true in s

  we use the set representation
- goals and preconditions of actions
  are given as sets of variables that must be true
  (values of other variables do not matter)
- effects of actions are given as sets of variables
  that are set to true and set to false, respectively

STRIPS Planning Task

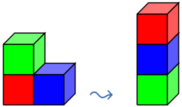
Definition (STRIPS Planning Task)

A STRIPS planning task is a 4 tuple $\Pi = \langle V, I, G, A \rangle$, where

- V is a finite set of binary state variables
- $I \subseteq V$ is the initial state
- $G \subseteq V$ is the set of goals
- A is a finite set of actions $a = \langle pre, add, del, cost \rangle$ with
  - preconditions $pre(a) \subseteq V$
  - add effects (or add list) $add(a) \subseteq V$
  - delete effects (or delete list) $del(a) \subseteq V$
  - costs $cost(a) \in \mathbb{N}_0$ ($cost(a) = 1$ if not specified explicitly)

remark: action costs are an extension of “traditional” STRIPS

Example: Blocks World in STRIPS



$\Pi = \langle V, I, G, A \rangle$ with:

- $V = \{on(R,B), on(R,G), on(B,R), on(B,G), on(G,R), on(G,B),$
  $on\text{-}table(R), on\text{-}table(B), on\text{-}table(G),$
  $clear(R), clear(B), clear(G)\}$
- $I = \{on(G,R), on\text{-}table(R), on\text{-}table(B), clear(G), clear(B)\}$
- $G = \{on(R,B), on(B,G)\}$
- $A = \{move(R,B,G), move(R,G,B), move(B,R,G),$
  $move(B,G,R), move(G,R,B), move(G,B,R),$
  $to\text{-}table(R,B), to\text{-}table(R,G), to\text{-}table(B,R),$
  $to\text{-}table(B,G), to\text{-}table(G,R), to\text{-}table(G,B),$
  $from\text{-}table(R,B), from\text{-}table(R,G), from\text{-}table(B,R),$
  $from\text{-}table(B,G), from\text{-}table(G,R), from\text{-}table(G,B)\}$

Example: Blocks World in STRIPS

action *move(R,B,G)*:

- $pre(move(R,B,G)) = \{on(R,B), clear(R), clear(G)\}$
- $add(move(R,B,G)) = \{on(R,G), clear(B)\}$
- $del(move(R,B,G)) = \{on(R,B), clear(G)\}$
- $cost(move(R,B,G)) = 1$

Example: Blocks World in STRIPS

action *move(R,B,G)*:

- $pre(move(R,B,G)) = \{on(R,B), clear(R), clear(G)\}$
- $add(move(R,B,G)) = \{on(R,G), clear(B)\}$
- $del(move(R,B,G)) = \{on(R,B), clear(G)\}$
- $cost(move(R,B,G)) = 1$

action *to-table(R, B)*:

- $pre(to\text{-}table(R, B)) =$
- $add(to\text{-}table(R, B)) =$
- $del(to\text{-}table(R, B)) =$
- $cost(to\text{-}table(R, B)) = 1$

Example: Blocks World in STRIPS

action *move(R,B,G)*:

- $pre(move(R,B,G)) = \{on(R,B), clear(R), clear(G)\}$
- $add(move(R,B,G)) = \{on(R,G), clear(B)\}$
- $del(move(R,B,G)) = \{on(R,B), clear(G)\}$
- $cost(move(R,B,G)) = 1$

action *to-table(R, B)*:

- $pre(to\text{-}table(R, B)) = \{clear(R), on(R, B)\}$
- $add(to\text{-}table(R, B)) = \{on\text{-}table(R), clear(B)\}$
- $del(to\text{-}table(R, B)) = \{on(R, B)\}$
- $cost(to\text{-}table(R, B)) = 1$

State Space for STRIPS Planning Task

Definition (state space induced by STRIPS planning task)

Let $\Pi = \langle V, I, G, A \rangle$ be a STRIPS planning task.

Then Π induces the state space $\mathcal{S}(\Pi) = \langle S, A, cost, T, s_0, S_\star \rangle$:

- set of states: $S = 2^V$ (= power set of V)
- actions: actions A as defined in Π
- action costs: $cost$ as defined in Π
- transitions: $s \xrightarrow{a} s'$ for states s, s' and action a iff
  - $pre(a) \subseteq s$ (preconditions satisfied)
  - $s' = (s \setminus del(a)) \cup add(a)$ (effects are applied)
- initial state: $s_0 = I$
- goal states: $s \in S_\star$ for state s iff $G \subseteq s$ (goals reached)

Why STRIPS?

- STRIPS is particularly simple

$\rightsquigarrow$ simplifies the design and implementation of planning algorithms and heuristics

- restriction to positive preconditions and goals makes it cumbersome for the “user” to model tasks directly in STRIPS
- but: STRIPS is equally “powerful” to much more complex planning formalisms

$\rightsquigarrow$ automatic “compilers” exist that translate more complex formalisms (like AIMA-PDDL and SAS$^+$) to STRIPS

SAS$^+$

Basic Concepts of SAS$^+$

basic concepts of SAS$^+$:

- very similar to STRIPS: state variables not necessarily binary, but with given finite domain (cf. CSPs)
- states are assignments to these variables (cf. CSPs)
- preconditions and goals given as partial assignments
- effects are assignments to subset of variables

SAS$^+$ Planning Task

Definition (SAS$^+$ planning task)

A SAS$^+$ planning task is a 5-tuple $\Pi = \langle V, dom, I, G, A \rangle$, where

- V is a finite set of state variables
- $dom(v)$ is a finite and non-empty domain for all $v \in V$
- I is a total assignment of V to dom, the initial state
- G is a partial assignment of V to dom, the goals
- A is a finite set of actions $a = \langle pre, eff, cost \rangle$ with
  - preconditions $pre(a)$, a partial assignment of V to dom
  - effects $eff(a)$, a partial assignment of V
  - cost $cost(a) \in \mathbb{N}_0$

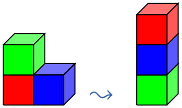
State Space of SAS$^+$ Planning Task

Definition (state space induced by SAS$^+$ planning task)

Let $\Pi = \langle V, dom, I, G, A\rangle$ be a SAS$^+$ planning task.
Then Π induces the state space $\mathcal{S}(\Pi) = \langle S, A, cost, T, s_0, S_\star\rangle$:

- set of states: total assignments of V according to dom
- actions: actions A as defined in Π
- action costs: $cost$ as defined in Π
- transitions: $s \xrightarrow{a} s'$ for states s, s' and action a iff
  - $pre(a)$ complies with s (precondition satisfied)
  - s' complies with $eff(a)$ for all variables mentioned in eff; complies with s for all other variables (effects are applied)
- initial state: $s_0 = I$
- goal states: $s \in S_\star$ for state s iff G complies with s

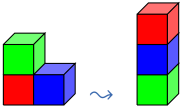
Example: Blocks World in SAS$^+$



$\Pi = \langle V, dom, I, G, A \rangle$ with:

- $V = \{pos(R), pos(B), pos(G),$
  $clear(R), clear(B), clear(G)\}$
- $dom(pos(R)) = \{B, G, T\}$
  $dom(pos(B)) = \{R, G, T\}$
  $dom(pos(G)) = \{R, B, T\}$
  $dom(clear(R)) = \{\mathbf{F}, \mathbf{T}\}$
  $dom(clear(B)) = \{\mathbf{F}, \mathbf{T}\}$
  $dom(clear(G)) = \{\mathbf{F}, \mathbf{T}\}$

Example: Blocks World in SAS$^+$



- $I = \{pos(R) \mapsto T, pos(B) \mapsto T, pos(G) \mapsto R,$
  $clear(R) \mapsto \mathbf{F}, clear(B) \mapsto \mathbf{T}, clear(G) \mapsto \mathbf{T}\}$
- $G = \{pos(R) \mapsto B, pos(B) \mapsto G\}$
- $A = \{move(R,B,G), move(R,G,B), move(B,R,G),$
  $move(B,G,R), move(G,R,B), move(G,B,R),$
  $move(R, B, T), move(R, G, T), move(B, R, T),$
  $move(B, G, T), move(G, R, T), move(G, B, T),$
  $move(R, T, B), move(R, T, G), move(B, T, R),$
  $move(B, T, G), move(G, T, R), move(G, T, B)\}$

Example: Blocks World in SAS$^+$

action *move(R,B,G)*:

- $pre(move(R,B,G)) = \{pos(R) \mapsto B, clear(R) \mapsto \mathbf{T}, clear(G) \mapsto \mathbf{T}\}$
- $eff(move(R,B,G)) =$
- $cost(move(R,B,G)) = 1$

Example: Blocks World in SAS$^+$

action *move(R,B,G)*:

- $pre(move(R,B,G)) = \{pos(R) \mapsto B, clear(R) \mapsto \mathbf{T}, clear(G) \mapsto \mathbf{T}\}$
- $eff(move(R,B,G)) = \{pos(R) \mapsto G, clear(B) \mapsto \mathbf{T}, clear(G) \mapsto \mathbf{F}\}$
- $cost(move(R,B,G)) = 1$

Why SAS$^+$

- modeling with finite-domain variables is often more user friendly than modeling with binary variables
- some techniques benefit from STRIPS, some from SAS$^+$
- automatic “compilers” exist that translate simpler formalisms (like AIMA-PDDL and STRIPS) to SAS$^+$

Why SAS+

- modeling with finite-domain variables is often more user friendly than modeling with binary variables
- some techniques benefit from STRIPS, some from SAS+
- automatic "compilers" exist that translate simpler formalisms (like AIMA-PDDL and STRIPS) to SAS+

$\rightsquigarrow$ in practice, planning systems convert automatically to the "best-fitting" planning formalism

Heuristics

Reminder: Heuristics

Definition (heuristic)

Let $\mathcal{S}$ be a state space with states S.

A heuristic function or heuristic for $\mathcal{S}$ is a function

$$h : S \longrightarrow \mathbb{R}_0^+ \cup \{\infty\},$$

mapping each state to a non-negative number (or ∞).

Reminder: Perfect Heuristic

Definition (perfect heuristic)

Let $\mathcal{S}$ be a state space with states S.

The perfect heuristic for $\mathcal{S}$, written h^*, maps each state $s \in S$ to the cost of an optimal solution for s.

remark: $h^*(s) = \infty$ if no solution for s exists

Reminder: Properties of Heuristics

Definition (safe, goal-aware, admissible, consistent)

Let $\mathcal{S}$ be a state space with states S.

A heuristic h for $\mathcal{S}$ is called

- safe if $h^*(s) = \infty$ for all $s \in S$ with $h(s) = \infty$
- goal-aware if $h(s) = 0$ for all goal states s
- admissible if $h(s) \leq h^*(s)$ for all states $s \in S$
- consistent if $h(s) \leq cost(a) + h(s')$ for all transitions $s \xrightarrow{a} s'$

A Simple Planning Heuristic

The STRIPS planner (Fikes & Nilsson, 1971) uses the number of goals not yet satisfied in a STRIPS planning task as heuristic:

$$h(s) := |G \setminus s|.$$

intuition: fewer unsatisfied goals $\rightsquigarrow$ closer to goal state

$\rightsquigarrow$ STRIPS heuristic

Problems of STRIPS Heuristic

drawback of STRIPS heuristic?

- rather uninformed:
  for state s, if there is no applicable action a in s such that applying a in s satisfies strictly more (or fewer) goals,
  then all successor states have the same heuristic value as s
- very sensitive to reformulation:
  can easily transform any planning task into an equivalent one where $h(s) = 1$ for all non-goal states
- ignores almost the whole task structure:
  the heuristic values do not depend on the actions

$\leadsto$ we need better methods to design heuristics

Planning Heuristics

General Procedure for Obtaining a Heuristic

Solve a simplified version of the problem.

Planning Heuristics

General Procedure for Obtaining a Heuristic

Solve a simplified version of the problem.

there are many ideas for domain-independent planning heuristics:

- abstraction ⇝ this course
- delete relaxation ⇝ this course
- landmarks
- critical paths
- network flows
- potential heuristics