Artificial Intelligence

Planning: Planning Tasks

Jendrik Seipp

Linköping University

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

Intended Learning Outcomes

- explain what "AI planning" is
- contrast the 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

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

Classification of Classical Planning

"Planning is the art and practice of thinking before acting."

— P. Haslum

environment:

- fully vs. partially vs. not observable
- single-agent vs. multi-agent (competitive and/or cooperative)
- deterministic vs. non-deterministic vs. stochastic
- episodic vs. sequential
- static vs. dynamic
- discrete vs. continuous

problem solving method:

- problem-specific vs. general vs. learning

Informal Description

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

objective of the agent:

- find a plan (a sequence of actions)
- that reaches a goal state
- from an initial state

performance measure:

- optimal planning: guarantee that returned plans are optimal, i.e., have minimal cost or a proof that no plan exists
- suboptimal planning (satisficing): minimize plan cost (given available resources)

Classical Planning

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

Looks familiar?

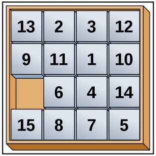
- description and quote also fit (state space) search
- environment as in constraint satisfaction problems

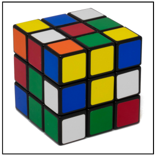
Classical Planning

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

Looks familiar?

- description and quote also fit (state space) search
- environment as in constraint satisfaction problems
- many previously encountered problems are indeed planning tasks or can be modelled as one, e.g.:



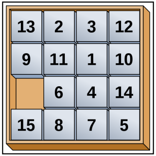


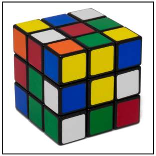
Classical Planning

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

Looks familiar?

- description and quote also fit (state space) search
- environment as in constraint satisfaction problems
- many previously encountered problems are indeed planning tasks or can be modelled as one, e.g.:





So what is old and what is new?

What is Old?

Reminder: State Spaces

To cleanly study search problems we need a formal model.

Definition (state space)

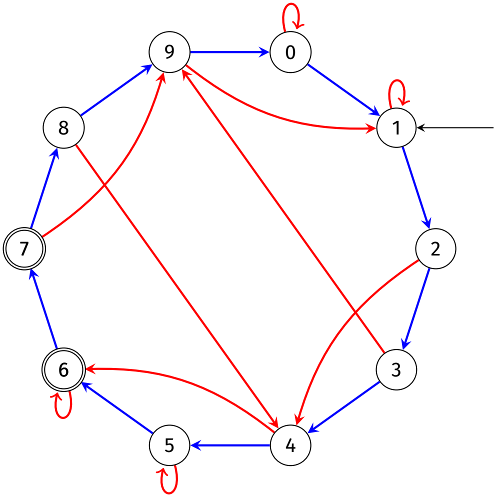
A state space or transition system is a 6-tuple $\mathcal{S} = \langle S, A, cost, T, s_I, S_\star \rangle$ with

- finite set of states S
- finite set of actions A
- action costs $cost : A \to \mathbb{R}_0^+$
- transition relation $T \subseteq S \times A \times S$ that is deterministic in $\langle s, a \rangle$
- initial state $s_I \in S$
- set of goal states $S_\star \subseteq S$

Reminder: Graph Interpretation

state spaces are often depicted as directed, labeled graphs

- states: graph vertices
- transitions: labeled arcs (here: colors instead of labels)
- initial state: incoming arrow
- goal states: double circles
- actions: the arc labels
- action costs: described separately (or implicitly = 1)



Reminder: Heuristic Search Algorithms

we still use heuristic search algorithms like A* or GBFS

⇝ search is guided by a heuristic

```
def best-first-search(⟨S, A, cost, T, s_I, S_⋆⟩):
    open := new MinHeap ordered by f
    open.insert(make_root_node(s_I))
    distances := new HashMap
    while not open.is_empty()
        n := open.pop_min()
        if n.state ∉ distances or g(n) < distances[n.state]:
            distances[n.state] := g(n)
            if n.state ∈ S_⋆: return extract_path(n)
            for each ⟨a, s′⟩ s.t. ⟨n.state,a,s'⟩ ∈ T:
                open.insert(make_node(n, a, s′))
    return unsolvable
```

What is New?

What is New?

```
def best-first-search(⟨S, A, cost, T, s_I, S_⋆⟩):
    open := new MinHeap ordered by f
    open.insert(make_root_node(s_I))
    distances := new HashMap
    while not open.is_empty()
        n := open.pop_min()
        if n.state ∉ distances or g(n) <distances[n.state]:
            distances[n.state] := g(n)
            if n.state ∈ S_⋆: return extract_path(n)
            for each ⟨a, s′⟩ s.t. ⟨n.state,a,s′⟩ ∈ T:
                open.insert(make_node(n, a, s′))
    return unsolvable
```

so far, we didn't care where these came from

What is New?

```
def best-first-search(⟨S, A, cost, T, s_I, S_⋆⟩):
    open := new MinHeap ordered by f
    open.insert(make_root_node(s_I))
    distances := new HashMap
    while not open.is_empty()
        n := open.pop_min()
        if n.state ∉ distances or g(n) <distances[n.state]:
            distances[n.state] := g(n)
            if n.state ∈ S_⋆: return extract_path(n)
            for each ⟨a, s′⟩ s.t. ⟨n.state,a,s'⟩ ∈ T:
                open.insert(make_node(n, a, s′))
    return unsolvable
```

so far, we didn't care where these came from

and the developer needed to know the problem to design a heuristic

State Spaces with Declarative Representations

- now we are interested in general algorithms, i.e., the developer of the solver does not know the tasks that the algorithm needs to solve

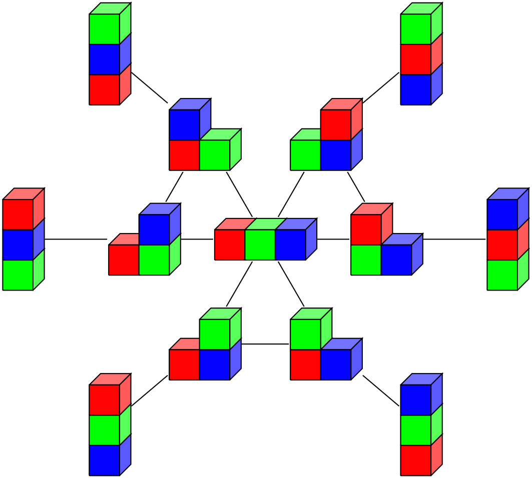
$\rightsquigarrow$ input is a state space description given in terms of suitable problem description language (planning formalism)

$\rightsquigarrow$ problem-independent heuristics!

- 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

$\rightsquigarrow$ e.g., n binary state variables can describe 2^n states

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

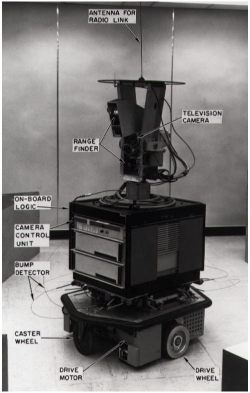
different variants: different planning formalisms

Three Planning Formalisms

- a description language for planning tasks
  is called a planning formalism
- we consider two planning formalisms:
  1. STRIPS (Stanford Research Institute Problem Solver)
  2. SAS^+ (Simplified Action Structures)
- more expressive formalisms exist,
  - e.g., PDDL (Planning Domain Definition Language)

STRIPS

STRIPS



- was developed as input language for Shakey the robot (1971)
- is the simplest commonly used planning formalism

STRIPS: Basic Concepts

- state variables V describe properties that can be true or false
- states are sets $s \subseteq V$ representing which properties are true
- goals are given as sets of properties that must be true (values of other variables do not matter)
- actions describe which transitions are allowed
  - preconditions denote properties required to apply the action
  - effects specify which properties the action changes

STRIPS Planning Task

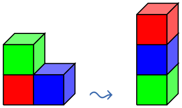
Definition (STRIPS Planning Task)

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)*:

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

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 $\mathcal{S}(\Pi) = \langle S, A, cost, T, s_I, S_\star \rangle$:

- 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: $s_I = I$
- goal states: $s \in S_\star$ for state s iff $G \subseteq s$ (goals reached)

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 $cost(a) \in \mathbb{N}_0$

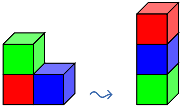
State Space of SAS$^+$ Planning Task

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

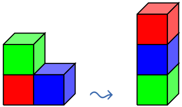
Example: Blocks World in SAS$^+$



$\Pi = \langle V, dom, I, G, A \rangle$ 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 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 STRIPS) to SAS$^+$

$\leadsto$ in practice, planning systems convert automatically to the “best-fitting” planning formalism