Planning (03)

Planning with state-space search

Alban Grastien
alban.grastien@rsise.anu.edu.au

Planning as a state-space search problem

Planning procedures are often state-space search problem, *i.e.* find a path on a state-space

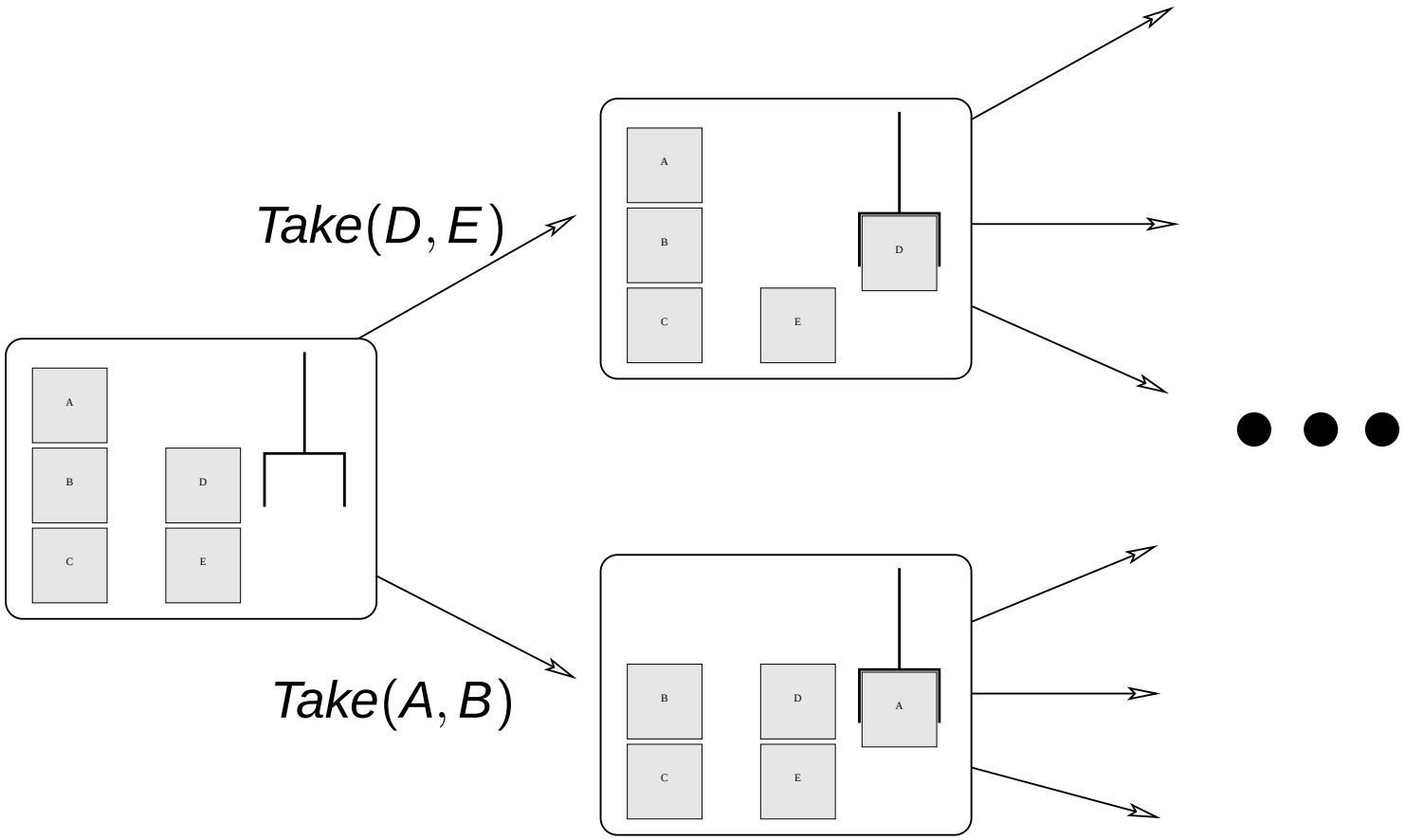
- Nodes = states of the world
- Transitions between nodes = actions
- Path on the state-space = plan

It is possible to explore the state-space in different ways

- forward, backward
- with different strategies (breath-first, depth-first, best-first, greedy, . . .)
- using heuristics

The STRIPS representation enables an *efficient* exploration

Forward search



Forward search: a general algorithm

General algorithm

param: s_0, O, g
$states = \{s_0\}$
$\pi(s_0) = \langle\rangle$
$E(s_0) = \{a \mid a$ is a ground instance of an operation $\in O$ such that PRECOND(a) is satisfied in s_0 $\}$
while true **do**
  if $states = \emptyset$ **then return** failure **end if**
  choose a state $s \in states$
  if $s \in g$ **then return** $\pi(s)$ **end if**
  if $E(s) = \emptyset$ **then**
    remove s from $states$
  else
    choose and **remove** an action $a \in E(s)$
    $s' \leftarrow \gamma(s, a)$
    if $s' \notin states$ **then**
      $\pi(s') = \pi(s).a$
      $E(s') = \{a \mid a$ is a ground instance of an operation $\in O$ such that PRECOND(a) is satisfied in s' $\}$
    end if
  end if
end while

Forward search: a general algorithm

STRIPS algorithm

param: s_0, O, g
$states = \{s_0\}$
$\pi(s_0) = \langle\rangle$
$E(s_0) = \{a \mid a$ is a ground instance of an operation $\in O$ such that $\textsc{Precond}(a) \subseteq s_0\ \}$
while true **do**
  if $states = \emptyset$ **then return** failure **end if**
  choose a state $s \in states$
  if $g \subseteq s$ **then return** $\pi(s)$ **end if**
  if $E(s) = \emptyset$ **then**
    remove s from $states$
  else
    choose and **remove** an action $a \in E(s)$
    $s' \leftarrow s \setminus \textsc{Eff}^-(a) \cup \textsc{Eff}^+(a)$
    if $s' \notin states$ **then**
      $\pi(s') = \pi(s).a$
      $E(s') = \{a \mid a$ is a ground instance of an operation $\in O$ such that $\textsc{Precond}(a) \subseteq s'\ \}$
    end if
  end if
end while

Properties of forward search

- It can be used in conjunction with any search strategy (*i.e* implementation of **choose**): breath-first, depth-first, iterative-deepening, greedy search, A*[1], IDA*, ...
- It is sound (any solution found is a good solution)
- It is complete (it returns a solution if there is one), for instance:
  - breath-first is complete if the number of actions is finite
  - depth-first is complete if the state space is finite[2]

[1] small modifications should be made for this strategy

[2] because the algorithm detects loops

Branching factor in forward search

Example:

- $1,000$ planes and 100 airport
- actions: one plane go from an airport to another airport
- goal: plane P_{153} in Rennes and plane P_{542} in Le Mans

$\Rightarrow$ $100,000$ possible actions from each states

$\Rightarrow$ $100,000$ different possible states after the first action

$\Rightarrow$ $\simeq 10^{10}$ different possible states after the second action

How to cope with this?

- domain-specific: search control rules, heuristics
- domain-independant: heuristics automatically generated from the STRIPS problem description

Search control knowledge

Knowledge on what constitutes a promising plan in the domain

Constraints on the plan

Temporal logic with First Order Logic

- Example: every stop of an elevator is made on a floor where a passenger is waiting or that is the destination of a passenger

$$\Box(\forall f_2 : \bigcirc atFloor(Elevator, f_2) \Rightarrow ($$
$$(\exists f_1 : \ f_1 \neq f_2 \wedge atFloor(Elevator, f_1)) \Rightarrow$$
$$((\exists p : waiting(p) \wedge atFloor(p, f_2)) \vee$$
$$(in(p, elevator) \wedge destination(p, f_2))))$$

Automatically prune the plan prefixes violating the control knowledge

cf. TLPlan [Bacchus & Kabanza, AIJ 2000]

Automatically heuristics in STRIPS

The heuristic $\Delta(s, g)$ estimates the minimum cost (number of actions) needed from s to g

The estimation is made by considering only $\textsc{Eff}^+$

- $\Delta(s, \{p\}) = 0$ if $p \in s$
- $\Delta(s, \{p\}) = \infty$ if $\forall a \in A : \ p \notin \textsc{Eff}^+(a)$
- $\Delta(s, \{p\}) = 1 + \min_{a \in A}\{\Delta(s, \textsc{Precond}(a) \mid p \in \textsc{Eff}^+(a))\}$
- $\Delta(s, g) = \max_{p \in g}\{Delta(s, \{p\})\}$

This heuristic is admissible

Regression planning (backward search)

Forward search

- starts from the initial state
- transition from s leads to the new state $\gamma(s, a)$

Backward search

- starts from the goal (set S of states $\neq$ state)
- transition from S leads to a new set of states $\gamma^{-1}(S, a)$

Two problems for the backward search:

- Which actions a to consider (relevance)?
- What is the definition of $\gamma^{-1}(S, a)$?

Inverse state transitions

If we reach an state s such that a is applicable in s and $g \subseteq \gamma(s, a)$, then we found a plan

- if $g \cap Eff^{-}(a) \neq \emptyset$, then $g \nsubseteq \gamma(s, a)$ (do not lead to a goal state)
- if $g \cap Eff^{+}(a) = \emptyset$ and $\gamma(s, a) \subseteq g$, then $s \subseteq g$ (we have already found a plan)

- An action a is relevant to a goal g if:
  - it makes at least one of g's propositions true: $g \cap Eff^{+}(a) \neq \emptyset$
  - it does not make any of g's proposition false: $g \cap Eff^{-}(a) = \emptyset$

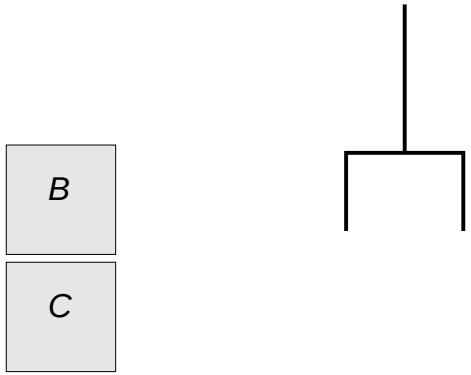
What does the state s must be such that $g \subseteq \gamma(s, a)$

- a must be applicable: $\textsc{Precond}(a) \subseteq s$
- the goal literals in g not given by a must be in a: $g - \textsc{Eff}^{+}(a) \subseteq s$

- $s = g \setminus \textsc{Eff}^{+}(a) \cup \textsc{Precond}(a)$

Backward search – example

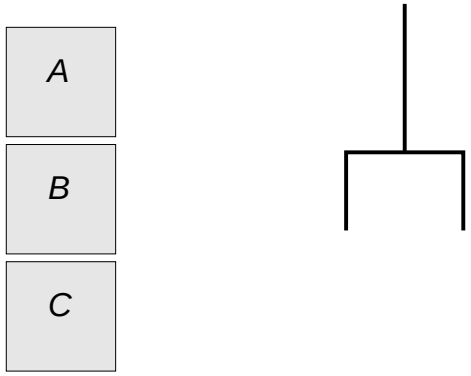
Goal state g: $OnTable(C) \wedge On(B, C) \wedge OnTop(B)$



Relevant actions:

- $PutOn(B, C)$
- $Take(A, B), \ldots, Take(E, B)$
- $\cancel{Take(A, D)}, \ldots$

New goal state $\gamma^{-1}(g, a)$: $OnTable(C) \wedge On(B, C) \wedge On(A, B) \wedge On(A)$



Backward search: a general algorithm

General algorithm

param: s_0, O, g
statesets $= \{g\}$
$\pi(g) = \langle\rangle$
$E(g) = \{a \mid a$ is a ground instance of an operation $\in O$ such that $\gamma(a,g)^{-1} \neq \emptyset$ and $\gamma(a,g)^{-1} \not\subseteq g\}$
while true **do**
  if *statesets* $= \emptyset$ **then return** failure **end if**
  choose a state set $S \in$ *statesets*
  if $s_0 \in S$ **then return** $\pi(S)$ **end if**
  if $E(S) = \emptyset$ **then**
    remove S from *statesets*
  else
    choose and **remove** an action $a \in E(S)$
    $S' \leftarrow \gamma^{-1}(S,a)$
    if $S' \notin$ *statesets* **then**
      $\pi(S') = a.\pi(S)$
      $E(S') = \{a \mid a$ is a ground instance of an operation $\in O$ such that $\gamma(a,S')^{-1} \neq \emptyset$ and $\gamma(a,S')^{-1} \not\subseteq S'\}$
    end if
  end if
end while

Backward search: a general algorithm

STRIPS algorithm

param: s_0, O, g
$statesets = \{g\}$
$\pi(g) = \langle\rangle$
$E(g) = \{a \mid a$ is a ground instance of an operation $\in O$ such that $g \cap Eff^{-}(a) = \emptyset$
and $g \cap Eff^{+}(a) \neq \emptyset$
while true **do**
  if $statesets = \emptyset$ **then return** failure **end if**
  choose a state set $S \in statesets$
  if $S \subseteq s_0$ **then return** $\pi(S)$ **end if**
  if $E(S) = \emptyset$ **then**
    remove S from $statesets$
  else
    choose and **remove** an action $a \in E(S)$
    $S' \leftarrow S \setminus \textsc{Eff}^{+}(a) \cup \textsc{Precond}(a)$
    if $S' \notin statesets$ **then**
      $\pi(S') = a.\pi(S)$
      $E(S') = \{a \mid a$ is a ground instance of an operation $\in O$ such that
      $S' \cap Eff^{-} = \emptyset$ and $S' \cap Eff^{+}(a) \neq \emptyset$
    end if
  end if
end while

Properties of backward search

- It can be used in conjunction with any search strategy (*i.e* implementation of **choose**): breath-first, depth-first, iterative-deepening, greedy search, A*[3], IDA*, ...
- It is sound (any solution found is a good solution)
- It is complete (it returns a solution if there is one), for instance:
  - breath-first is complete if the number of actions is finite
  - depth-first is complete if the state space is finite[4]

- Improvement: it is possible not to add a subgoal S if it is a subset of $S' \in statesets$

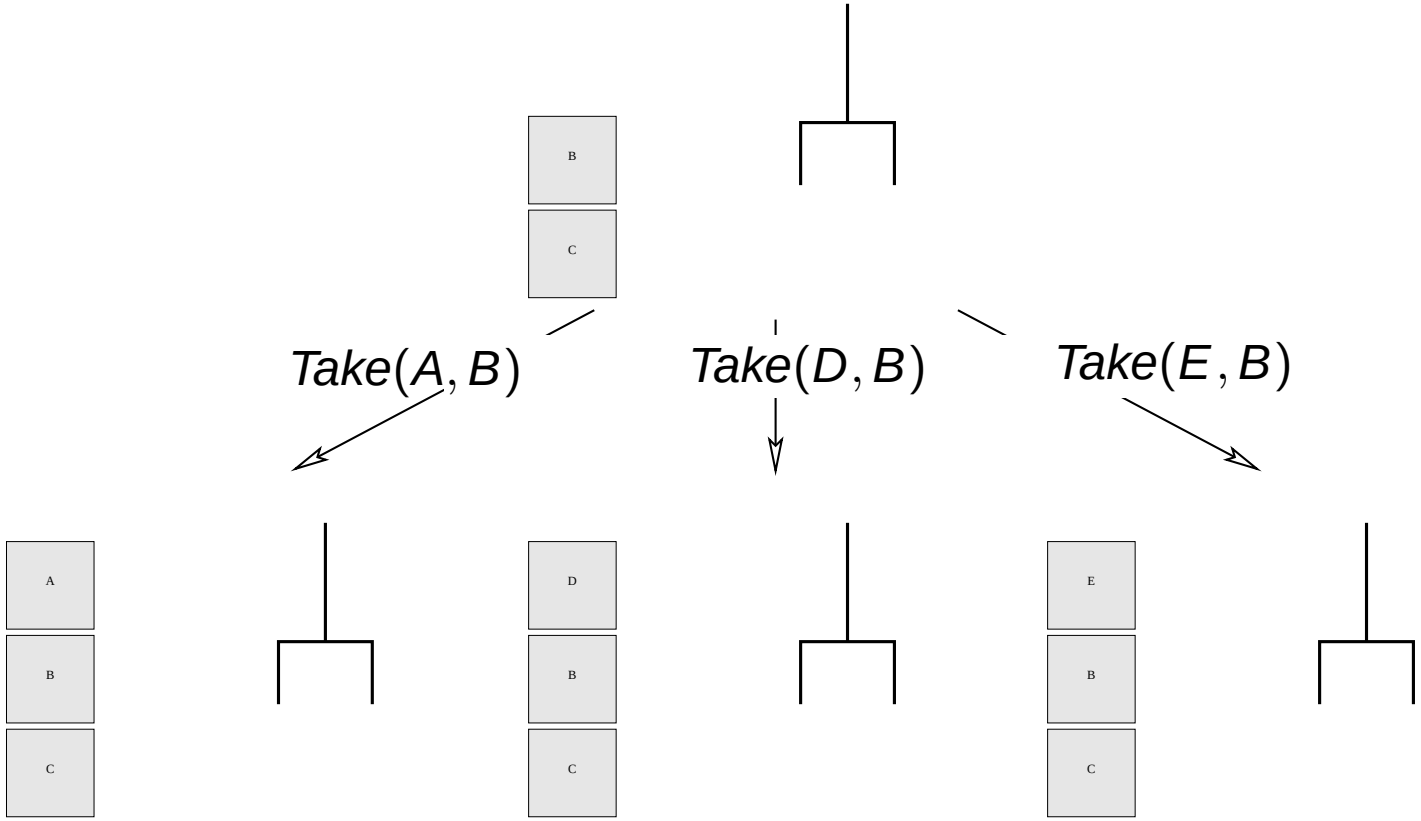
[3] small modifications should be made for this strategy

[4] because the algorithm detects loops

Lifting

The branching factor can be substiancially reduced if we <u>partially instanciate</u> the operators

Example:



Lifting

The branching factor can be substiancially reduced if we <u>partially instanciate</u> the operators

Example: