Automated Planning

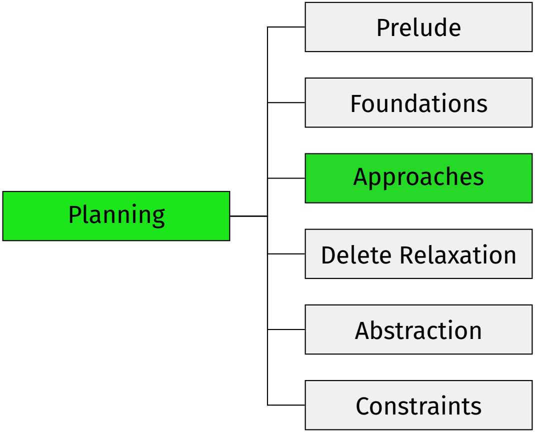
C2. Progression and Regression Search

Jendrik Seipp

Linköping University

based on slides from the AI group at the University of Basel

Content of this Course



Introduction

Search Direction

Search direction

- one dimension for classifying search algorithms
- **forward** search from initial state to goal based on **progression**
- **backward** search from goal to initial state based on **regression**
- **bidirectional** search

In this chapter we look into progression and regression planning.

Reminder: Interface for Heuristic Search Algorithms

Abstract Interface Needed for Heuristic Search Algorithms

- init() $\rightsquigarrow$ returns initial state
- is_goal(s) $\rightsquigarrow$ tests if s is a goal state
- succ(s) $\rightsquigarrow$ returns all pairs $\langle a, s' \rangle$ with $s \xrightarrow{a} s'$
- cost(a) $\rightsquigarrow$ returns cost of action a
- h(s) $\rightsquigarrow$ returns heuristic value for state s

Progression

Planning by Forward Search: Progression

Progression: Computing the successor state $s[\![o]\!]$ of a state s with respect to an operator o.

Progression planners find solutions by forward search:

- start from initial state
- iteratively pick a previously generated state and progress it through an operator, generating a new state
- solution found when a goal state generated

pro: very easy and efficient to implement

Search Space for Progression

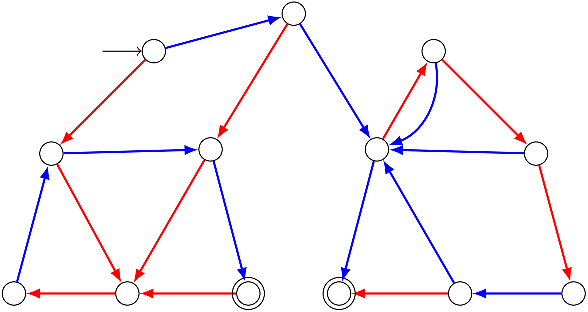
Search Space for Progression

search space for progression in a planning task $\Pi = \langle V, I, O, \gamma \rangle$
(search states are world states s of Π;
actions of search space are operators $o \in O$)

- init() $\rightsquigarrow$ returns I
- is_goal(s) $\rightsquigarrow$ tests if $s \models \gamma$
- succ(s) $\rightsquigarrow$ returns all pairs $\langle o, s[\![o]\!] \rangle$ where $o \in O$ and o is applicable in s
- cost(o) $\rightsquigarrow$ returns $cost(o)$ as defined in Π
- h(s) $\rightsquigarrow$ estimates cost from s to γ

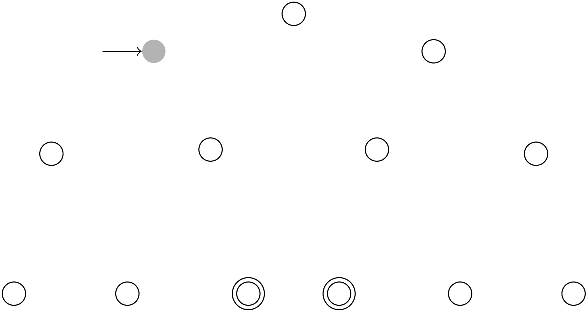
Progression Planning Example (Breadth-first Search)

Example of a progression search



Progression Planning Example (Breadth-first Search)

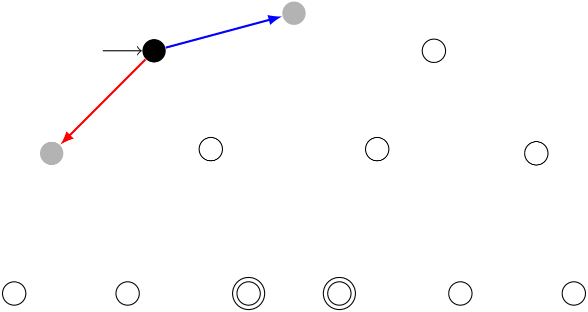
Example of a progression search



Grey states are generated but not expanded. Black states are expanded.

Progression Planning Example (Breadth-first Search)

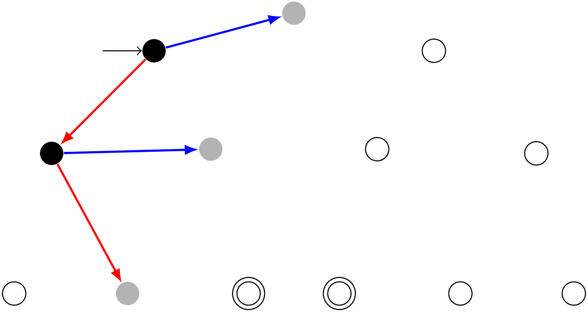
Example of a progression search



Grey states are generated but not expanded. Black states are expanded.

Progression Planning Example (Breadth-first Search)

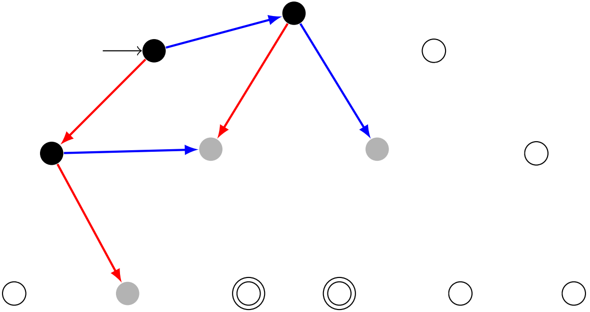
Example of a progression search



Grey states are generated but not expanded. Black states are expanded.

Progression Planning Example (Breadth-first Search)

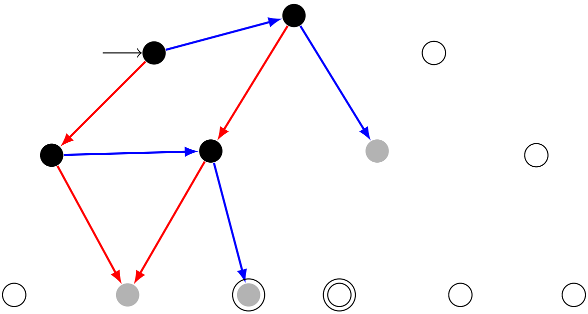
Example of a progression search



Grey states are generated but not expanded. Black states are expanded.

Progression Planning Example (Breadth-first Search)

Example of a progression search



Grey states are generated but not expanded. Black states are expanded. BrFS checks goal when generating successors, so the goal state is not expanded.

Regression

Forward Search vs. Backward Search

Searching planning tasks in forward vs. backward direction is not symmetric:

- forward search starts from a single initial state; backward search starts from a set of goal states
- when applying an operator o in a state s in forward direction, there is a unique successor state s'; if we just applied operator o and ended up in state s', there can be several possible predecessor states s

$\rightsquigarrow$ in most natural representation for backward search in planning, each search state corresponds to a set of world states

Planning by Backward Search: Regression

Regression: Computing the possible predecessor states $regr(S', o)$ of a set of states S' ("subgoal") given the last operator o that was applied.

Regression planners find solutions by backward search:

- start from set of goal states
- iteratively pick a previously generated subgoal (state set) and regress it through an operator, generating a new subgoal
- solution found when a generated subgoal includes initial state

pro: can handle many states simultaneously
con: basic operations complicated and expensive

Search Space Representation in Regression Planners

identify state sets with logical formulas (again):

- each search state corresponds to a set of world states ("subgoal")
- each search state is represented by a logical formula:
  φ represents $\{s \in S \mid s \models \varphi\}$
- many basic search operations like detecting duplicates
  are NP-complete or coNP-complete

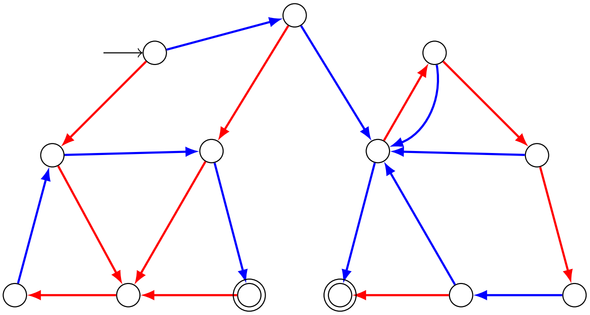
Search Space for Regression

Search Space for Regression

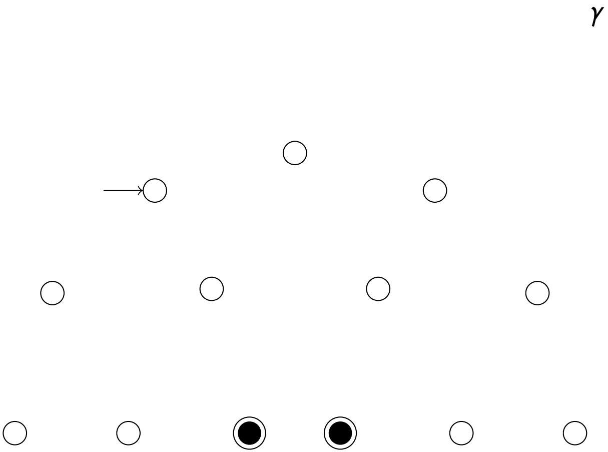
search space for regression in a planning task $\Pi = \langle V, I, O, \gamma \rangle$
(search states are formulas φ describing sets of world states;
actions of search space are operators $o \in O$)

- init() $\rightsquigarrow$ returns γ
- is_goal(φ) $\rightsquigarrow$ tests if $I \models \varphi$
- succ(φ) $\rightsquigarrow$ returns all pairs $\langle o, regr(\varphi, o) \rangle$ where $o \in O$ and $regr(\varphi, o)$ is defined
- cost(o) $\rightsquigarrow$ returns $cost(o)$ as defined in Π
- h(φ) $\rightsquigarrow$ estimates cost from I to φ

Regression Planning Example (Depth-first Search)



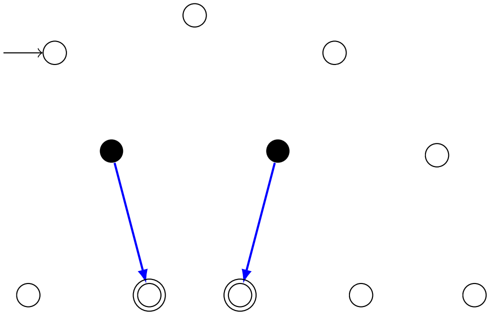
Regression Planning Example (Depth-first Search)



Regression Planning Example (Depth-first Search)

$\varphi_1 = regr(\gamma, \longrightarrow)$

$\varphi_1 \longrightarrow \gamma$

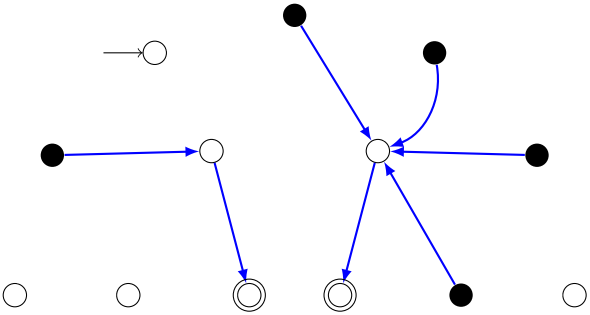


Regression Planning Example (Depth-first Search)

$\varphi_1 = regr(\gamma, \longrightarrow)$

$\varphi_2 = regr(\varphi_1, \longrightarrow)$

$\varphi_2 \longrightarrow \varphi_1 \longrightarrow \gamma$



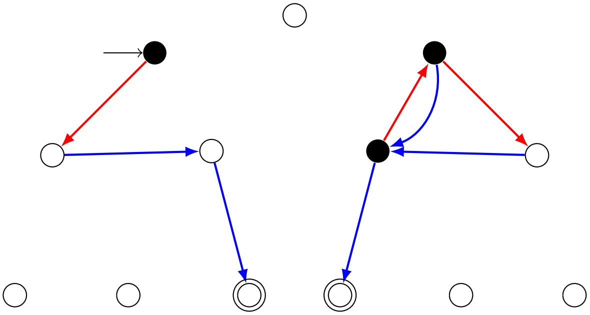
Regression Planning Example (Depth-first Search)

$\varphi_1 = regr(\gamma, \longrightarrow)$

$\varphi_2 = regr(\varphi_1, \longrightarrow)$

$\varphi_3 = regr(\varphi_2, \longrightarrow), I \models \varphi_3$

$\varphi_3 \longrightarrow \varphi_2 \longrightarrow \varphi_1 \longrightarrow \gamma$



Regression for STRIPS Tasks

Regression for STRIPS Planning Tasks

Regression for STRIPS planning tasks is much simpler than the general case:

- Consider subgoal φ that is conjunction of atoms $a_1 \land \cdots \land a_n$ (e.g., the original goal γ of the planning task).
- First step: Choose an operator o that deletes no a_i.
- Second step: Remove any atoms added by o from φ.
- Third step: Conjoin $pre(o)$ to φ.

$\rightsquigarrow$ Outcome of this is regression of φ w.r.t. o.
It is again a conjunction of atoms.

optimization: only consider operators adding at least one a_i

STRIPS Regression

Definition (STRIPS Regression)

Let $\varphi = \varphi_1 \wedge \dots \wedge \varphi_n$ be a conjunction of atoms, and let o be a STRIPS operator which adds the atoms $a_1, \dots, a_k$ and deletes the atoms $d_1, \dots, d_l$.

The STRIPS regression of φ with respect to o is

$$sregr(\varphi, o) := \begin{cases} \bot & \text{if } \varphi_i = d_j \text{ for some } i, j \\ pre(o) \wedge \bigwedge(\{\varphi_1, \dots, \varphi_n\} \setminus \{a_1, \dots, a_k\}) & \text{else} \end{cases}$$

Note: $sregr(\varphi, o)$ is again a conjunction of atoms, or $\bot$.

Does this Capture the Idea of Regression?

For our definition to capture the concept of regression,
it must have the following property:

Regression Property

For all sets of states described by a conjunction of atoms φ, all states s and all STRIPS operators o,

$$s \models sregr(\varphi, o) \quad \text{iff} \quad s[\![o]\!] \models \varphi.$$

This is indeed true. In fact, this property holds not just for STRIPS, but also for general regression.

Summary

Summary

- Progression search proceeds forward from the initial state.
- In progression search, the search space is identical to the state space of the planning task.
- Regression search proceeds backwards from the goal.
- Each search state corresponds to a set of world states, for example represented by a formula.
- Regression is simple for STRIPS operators.