

# Planning and Optimization

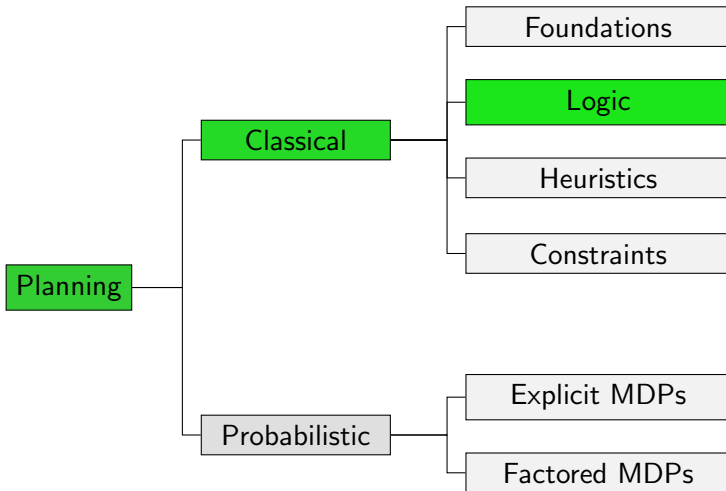
## B2. Progression and Regression Search

Malte Helmert and Gabriele Röger

Universität Basel

October 5, 2020

# 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

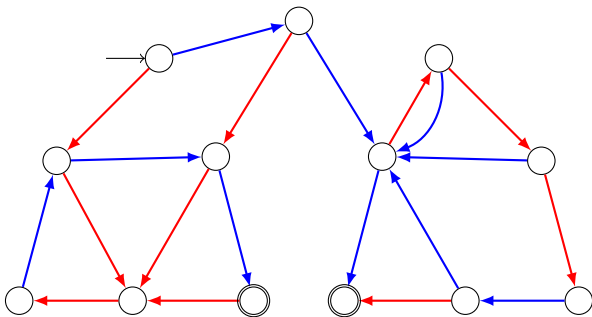
search space for progression in a planning task  $\Pi = \langle V, I, O, \gamma \rangle$   
(search states are world states  $s$  of  $\Pi$ ;  
actions of search space are operators  $o \in O$ )

- **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  $\Pi$
- **h( $s$ )**  $\rightsquigarrow$  estimates cost from  $s$  to  $\gamma$  ( $\rightsquigarrow$  [Parts C–F](#))



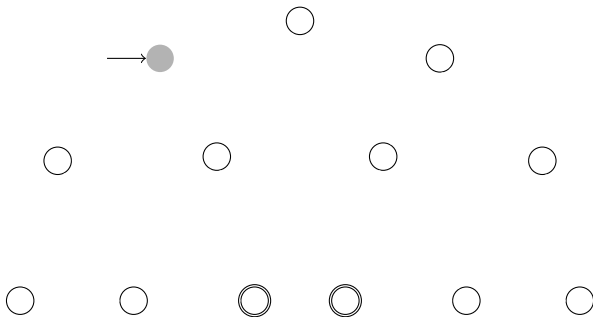
# Progression Planning Example

Example of a progression search



# Progression Planning Example

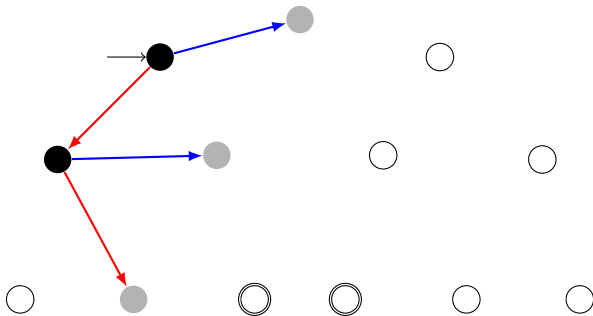
Example of a progression search





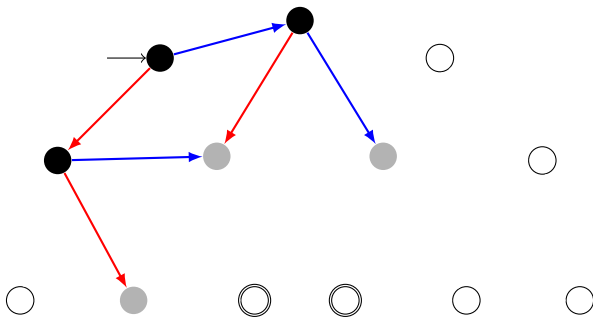
# Progression Planning Example

Example of a progression search



# Progression Planning Example

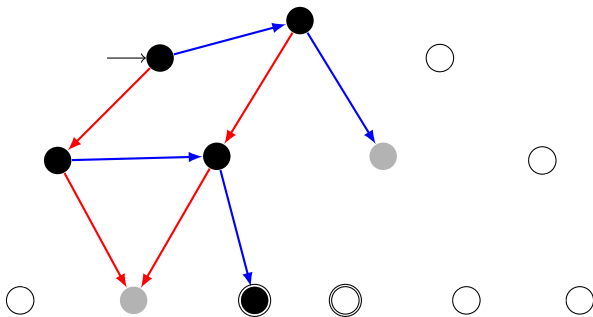
Example of a progression search





# Progression Planning Example

Example of a progression search



# 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**  $s$
- ↪ 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.

↪ formal definition in next chapter

**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**:  
 $\varphi$  represents  $\{s \in \mathcal{S} \mid s \models \varphi\}$
- many basic search operations like detecting duplicates are NP-complete or coNP-complete

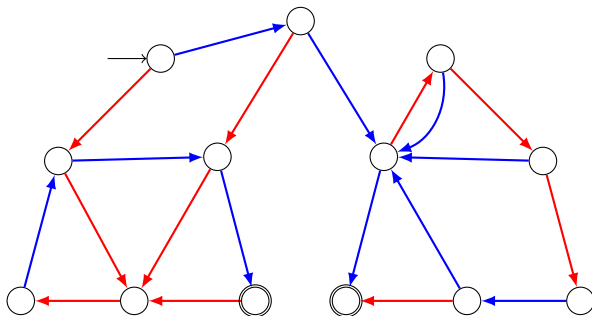
# Search Space for Regression

## Search Space for Regression

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

- **init()**  $\rightsquigarrow$  returns  $\gamma$
- **is\_goal( $\varphi$ )**  $\rightsquigarrow$  tests if  $I \models \varphi$
- **succ( $\varphi$ )**  $\rightsquigarrow$  returns all pairs  $\langle o, \text{regr}(\varphi, o) \rangle$   
where  $o \in O$  and  $\text{regr}(\varphi, o)$  is defined
- **cost( $o$ )**  $\rightsquigarrow$  returns  $\text{cost}(o)$  as defined in  $\Pi$
- **h( $\varphi$ )**  $\rightsquigarrow$  estimates cost from  $I$  to  $\varphi$  ( $\rightsquigarrow$  [Parts C–F](#))

# Regression Planning Example (Depth-first Search)





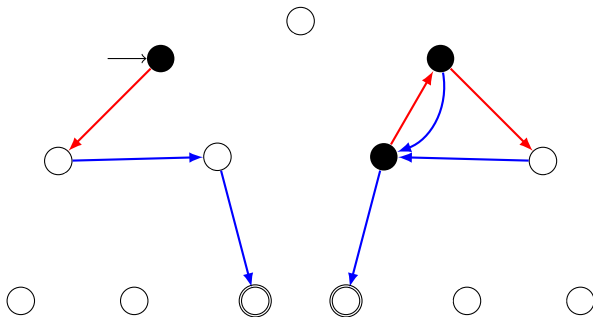






# Regression Planning Example (Depth-first Search)

$$\begin{aligned} \varphi_1 &= \text{regr}(\gamma, \rightarrow) & \varphi_3 &\xrightarrow{\text{red}} \varphi_2 \xrightarrow{\text{blue}} \varphi_1 \xrightarrow{\text{blue}} \gamma \\ \varphi_2 &= \text{regr}(\varphi_1, \rightarrow) \\ \varphi_3 &= \text{regr}(\varphi_2, \rightarrow), I \models \varphi_3 \end{aligned}$$



# Regression for STRIPS Tasks

# Regression for STRIPS Planning Tasks

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

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

↪ Outcome of this is **regression** of  $\varphi$  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  $\varphi$  with respect to  $o$  is

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

**Note:**  $\text{sregr}(\varphi, o)$  is again a conjunction of atoms, or  $\perp$ .

## 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  $\varphi$ , all states  $s$  and all STRIPS operators  $o$ ,

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

This is indeed true. We do not prove it now because we prove this property for general regression (not just STRIPS) later.

# 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.
- The theory for **general regression** is more complex. This is the topic of the following chapters.