# Planning

<table>
<thead>
<tr>
<th>Planning</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition and Assumptions</td>
<td>2</td>
</tr>
<tr>
<td>Actions</td>
<td>3</td>
</tr>
<tr>
<td>Delivery Robot Example</td>
<td>4</td>
</tr>
<tr>
<td>Explicit State Space Representation</td>
<td>5</td>
</tr>
<tr>
<td>Feature-Based Representation of Actions</td>
<td>6</td>
</tr>
<tr>
<td>Example Feature-Based Representation</td>
<td>7</td>
</tr>
<tr>
<td>STRIPS Representation</td>
<td>8</td>
</tr>
<tr>
<td>Planning Problem</td>
<td>9</td>
</tr>
</tbody>
</table>

| Forward Planning                             | 10   |
| Forward Planning                             | 10   |
| Forward Planning Example                     | 11   |
| Forward Planning Comments                    | 12   |

| Regression Planning                          | 13   |
| Regression Planning                          | 13   |
| Goals, Subgoals and Edges                    | 14   |
| Regression Example                           | 15   |
| Regression Planning Comments                 | 16   |

| Planning as a CSP                            | 17   |
| Planning as a CSP                            | 17   |
| Action Variables                             | 18   |
| Constraints                                   | 19   |
| CSP for Delivery Robot                        | 20   |
| Example Constraints                           | 21   |

| Partial Order Planning                       | 22   |
| Partial Order Planning                       | 22   |
| Partial Plan Search                          | 23   |

Initial Plan ................................................. 24
Intermediate Plan ............................................ 25
Flaws in Intermediate Plan ......................... 26
Almost Done Plan ............................................ 27
Flaws in Almost Done Plan ............................... 28
Comments ..................................................... 29
Planning

Definition and Assumptions

- Planning is finding actions to achieve goals.
- Initial assumptions:
  - The world is deterministic.
  - There are no events outside of the control of the agents that change the state of the world.
  - The agent knows what state it is in.
  - Time progresses discretely from one state to the next.
- Goals are features of states that need to be achieved or maintained.

Actions

- A deterministic action is a partial function from states to states.
- The preconditions of an action specify when the action can be performed.
- The effect of an action specifies the resulting state.

Delivery Robot Example

Features:
- RLoc: Rob’s location
- RH: Rob has coffee
- SWC: Sam wants coffee
- MW: Mail is waiting
- RHM: Rob has mail

Actions:
- mc: move clockwise
- mcc: move counterclockwise
- nm: no move
- puc: pickup coffee
- dc: deliver coffee
- pum: pickup mail
- dm: deliver mail

Explicit State Space Representation

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Resulting State</th>
</tr>
</thead>
<tbody>
<tr>
<td>lab, rhc, swc, mw, rhm</td>
<td>mc</td>
<td>mr, rhc, swc, mw, rhm</td>
</tr>
<tr>
<td>lab, rhc, swc, mw, rhm</td>
<td>mcc</td>
<td>off, rhc, swc, mw, rhm</td>
</tr>
<tr>
<td>off, rhc, swc, mw, rhm</td>
<td>dm</td>
<td>off, rhc, swc, mw, rhm</td>
</tr>
<tr>
<td>off, rhc, swc, mw, rhm</td>
<td>mcc</td>
<td>cs, rhc, swc, mw, rhm</td>
</tr>
</tbody>
</table>

Features:
- RLoc: Rob’s location
- RH: Rob has coffee
- SWC: Sam wants coffee
- MW: Mail is waiting
- RHM: Rob has mail

Actions:
- mc: move clockwise
- mcc: move counterclockwise
- nm: no move
- puc: pickup coffee
- dc: deliver coffee
- pum: pickup mail
- dm: deliver mail

Feature-Based Representation of Actions

For each action:
- A precondition specifies when the action can be carried out.

For each feature:
- Causal rules specify when the feature gets a new value.
- Frame rules specify when the feature keeps its value.

Example Feature-Based Representation

Precondition of pick-up coffee (puc):

Act = puc \rightarrow RLoc = cs \land \neg rhc

Rules for next location = coffee shop (RLoc = cs):

- RLoc = off \land Act = mc \rightarrow RLoc’ = cs
- RLoc = mr \land Act = mc \rightarrow RLoc’ = cs
- RLoc = cs \land Act \neq mc \land Act \neq mc \rightarrow RLoc’ = cs

Rules for “robot has coffee” (rhc)

rhc \land Act \neq dc \rightarrow rhc’
Act = puc \rightarrow rhc’
**STRIPS Representation**

For each action:
- A **precondition** specifies when the action can be carried out.
- An **effect** assigns values to features that are changed by this action.

**Action**: Pick-up coffee \((puc)\):
- Precondition: \(R\text{Loc} = \text{cs} \land \neg rhc\)
- Effect: \(rhc\)

**Action**: Deliver coffee \((dc)\):
- Precondition: \(\text{off} \land rhc\)
- Effect: \(\neg rhc \land \neg swc\)

---

**Planning Problem**

Given:
- A description of the effects and preconditions of the actions
- A description of the initial state
- A goal to achieve

find a sequence of actions that is possible and will result in a state satisfying the goal.

---

**Forward Planning**

Idea: search in the state-space graph.
- The nodes represent the states
- The arcs correspond to the actions: The edges from a state \(s\) represent all of the actions that are legal in state \(s\).
- A plan is a path from the state representing the initial state to a state that satisfies the goal.
Regression Planning

Idea: search backwards from the goal description: nodes correspond to goals and subgoals, and arcs to actions that achieve goals.

- Nodes are partial assignments of values to features.
- Edges correspond to actions that can achieve one of the assignments.
- The edge points to a node that includes the preconditions of the action.
- The initial node is the goal to be achieved.
- Search succeeds if a node is true of the initial state.

Goals, Subgoals and Edges

- A node $g$ represents goals (or subgoals) to be achieved: represented as a value assignment to one or more features:
  \[ X_i = v_i, X_j = v_j, \ldots \]
- An action from $g$ includes part of $g$ as an effect, with no effect that contradicts $g$.
- The edge goes to a node $g'$ that must contain:
  - The preconditions of the action
  - All elements of $g$ not in the action’s effect
  - $g'$ must not have contradictions.

Regression Planning Comments

- You can define a heuristic function that estimates how difficult it is to achieve a node from the initial state.
- You can use domain-specific knowledge to remove impossible goals.
- Whether forward or regression is more efficient depends on the branching factor and how good the heuristics are.
- Forward planning is unconstrained by the goal (except as a source of heuristics).
- Regression planning is unconstrained by the initial state (except as a source of heuristics).

Regression Example

Actions
- mc: move clockwise
- mac: move anticlockwise
- puc: pick up coffee
- dc: deliver coffee
- pm: pick up mail
- dm: deliver mail

Locations:
- cs: coffee shop
- of: office
- lab: laboratory
- mr: mail room

Feature values:
- rhc: robot has coffee
- swc: Sam wants coffee
- mw: mail waiting
- rhm: robot has mail

Diagram:

- [cs, rhc] → [off, rhc] via dc
- [off, rhc] → [lab, rhc] via [mac]
- [off, rhc] → [mr, rhc] via [mac]
- [cs] → [rhc] via puc
- [rhc] → [rhc] via mc
Planning as a CSP

### Planning as Constraint Satisfaction Problems

- **Idea:** Create a CSP for a limited-length plan.
  - If length $k$ fails, increment $k$ and try again.
- **Algorithm:**
  - Choose a plan length $k$ (also called the horizon).
  - Create a variable for each feature and each time from 0 to $k$.
  - Create a variable for each action for each time in the range 0 to $k - 1$.
  - Add constraints between features and actions, and solve.
- **Very effective with a specialized algorithm.**

### Action Variables

- **PUC:** Boolean var, robot picks up coffee.
- **DelC:** Boolean var, robot delivers coffee.
- **PUM:** Boolean var, robot picks up mail.
- **DelM:** Boolean var, robot delivers mail.
- **Move:** variable with domain $\{mc, nmc, nm\}$ specifies whether the robot moves clockwise, counterclockwise or doesn’t move.

### Constraints

- **State constraints** between variables at the same time step.
- **Precondition constraints** between state vars at time $t$ and action vars at time $t$.
- **Effect constraints** between state vars at time $t$, action vars at time $t$, and state vars at time $t + 1$.
- **Action constraints** specify which actions cannot co-occur (also called mutex constraints).
- **Initial state constraints** on the state at time 0.
- **Goal constraints** specify that goals are satisfied at time $k$.
Partial Order Planning

□ Forward, regression and CSP planners commit to unnecessary action orderings.
□ Idea: Maintain a partial ordering between actions and only commit to an ordering between actions when forced.
□ A partial-order plan is a partial ordering of actions \((\text{act}_0 < \text{act}_1)\) represents \(\text{act}_0\) before \(\text{act}_1\). The problem is solved when every total ordering is a solution.
□ Algorithm Idea: Start with an unfinished plan and search over ways to fix it.

Partial Plan Search

Procedure \(\text{Partial-Plan-Search}(s, g, A)\)

Inputs: \(s, g, A\): initial state, goal, actions
\(\text{start} \leftarrow \) pseudo-action with \(s\) as effect
\(\text{finish} \leftarrow \) pseudo-action with \(g\) as precondition
insert plan \(\text{start} < \text{finish}\) into \(\text{Frontier}\)
while \(\text{Frontier}\) is not empty
\(p \leftarrow \) remove a plan from \(\text{Frontier}\)
if \(p\) has no flaws then return \(p\)
select a flaw \(w\) in \(p\)
for each fix \(x\) for \(w\) in \(p\)
\(p' \leftarrow \) copy of \(p\) including \(x\)
insert \(p'\) into \(\text{Frontier}\)
return null

Initial Plan

\(\text{start}\)
\(cs, \text{rhc, swc, mw, rhm}\)
\(\text{lab, swc}\)
\(\text{finish}\)

flaw: open precondition: \(\text{lab}\) not achieved
fix: add \text{move} from \text{off} to \text{lab}

flaw: open precondition: \(\text{swc}\) not achieved
fix: add \text{dc} action

Intermediate Plan

\(\text{start}\)
\(cs, \text{rhc, swc, mw, rhm}\)
\(\text{off}\)
\(\text{move}\)
\(\text{dc}\)
\(\text{lab}\)
\(\text{lab, swc}\)
\(\text{finish}\)

CS 3793 Artificial Intelligence Planning – 22

CS 3793 Artificial Intelligence Planning – 23

CS 3793 Artificial Intelligence Planning – 24

CS 3793 Artificial Intelligence Planning – 25
Flaws in Intermediate Plan
flaw: open precondition: off of move not achieved
fix: add move from cs to off
flaw: open precondition: off of dc not achieved
fix: use same move from cs to off
flaw: open precondition: rhc of dc not achieved
fix: add puc action
flaw: open preconditions of puc and new move
fix: use effects of start (use initial state)

Flaws in Almost Done Plan
flaw: conflict: the first move conflicts with cs
staying true between start and puc
fix: order first move after puc
flaw: conflict: the second move conflicts with off
staying true between first move and dc.
fix: order second move after dc

Comments
- The above example doesn’t show the search over all the fixes that don’t work.
- Works well if plans for different subgoals do not interact much.
- Compared to forward/regression planning, adds search levels for reusing actions and resolving conflicts.
- In practice, CSP planners are more efficient than partial order/forward/regression planning.