Planning
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.
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:**
- \( R_{Loc} \): Rob’s location
- \( R_{HC} \): Rob has coffee
- \( S_{WC} \): Sam wants coffee
- \( M_{W} \): Mail is waiting
- \( R_{HM} \): 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><code>lab, rhc, swc, mw, rhm</code></td>
<td><code>mc</code></td>
<td><code>mr, rhc, swc, mw, rhm</code></td>
</tr>
<tr>
<td><code>lab, rhc, swc, mw, rhm</code></td>
<td><code>mcc</code></td>
<td><code>off, rhc, swc, mw, rhm</code></td>
</tr>
<tr>
<td><code>off, rhc, swc, mw, rhm</code></td>
<td><code>dm</code></td>
<td><code>off, rhc, swc, mw, rhm</code></td>
</tr>
<tr>
<td><code>off, rhc, swc, mw, rhm</code></td>
<td><code>mcc</code></td>
<td><code>cs, rhc, swc, mw, rhm</code></td>
</tr>
<tr>
<td><code>off, rhc, swc, mw, rhm</code></td>
<td><code>mc</code></td>
<td><code>lab, rhc, swc, mw, rhm</code></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Features:
- **rloc**: Rob’s location
- **rhc**: 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

---

**Planning Assumptions**

**Actions**

**Example**

**Feature-Based Example**

**Feature-Based STRIPS Planning Problem**

**Forward Planning**

**Regression Planning**

**Planning as a CSP**

**Partial Order Planning**
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 ($puc$):

$$\text{Act} = puc \rightarrow \text{RLoc} = cs \land \neg rhc$$

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

- $RLoc = off \land \text{Act} = mcc \rightarrow RLoc' = cs$
- $RLoc = mr \land \text{Act} = mc \rightarrow RLoc' = cs$
- $RLoc = cs \land \text{Act} \neq mcc \land \text{Act} \neq mc \rightarrow RLoc' = cs$

Rules for “robot has coffee” ($rhc$)

$$rhc \land \text{Act} \neq dc \rightarrow rhc'$$

$$\text{Act} = puc \rightarrow rhc'$$
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:** \(RLoc = cs \land \neg rhc\)
- **Effect:** \(rhc\)

**Action: Deliver coffee** \((dc)\):

- **Precondition:** \(off \land rhc\)
- **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.
Forward Planning Example

### Planning

#### Forward Planning

**Definition**

**Example**

**Comments**

#### Regression Planning

#### Planning as a CSP

#### Partial Order Planning

---

**Actions**

- **mc**: move clockwise
- **mac**: move anticlockwise
- **nm**: no move
- **puc**: pick up coffee
- **dc**: deliver coffee
- **pum**: pick up mail
- **dm**: deliver mail

**mc**

**Locations:**

- **cs**: coffee shop
- **off**: office
- **lab**: laboratory
- **mr**: mail room

**Feature values**

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

---

**Diagram**

[Diagram showing the planning process with actions and locations]
The search graph can be constructed on demand: you only construct reachable states.

Forward search can use knowledge specified as:

- a heuristic function that estimates the number of steps to the goal
- domain-specific pruning of neighbors:
  - don’t pick-up coffee unless Sam wants coffee
  - unless the goal involves time constraints, don’t do the “no move” action.
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.
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 Example

Planning
Forward Planning
Regression Planning
Definition
Goals and Edges
Example
Comments
Planning as a CSP
Partial Order Planning

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

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

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

**Example**

<table>
<thead>
<tr>
<th>Action</th>
<th>Goal State</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc</td>
<td>[cs, rhc]</td>
</tr>
<tr>
<td>mac</td>
<td>[mr, rhc]</td>
</tr>
<tr>
<td>puc</td>
<td>[mr, rhc]</td>
</tr>
<tr>
<td>dc</td>
<td>[off, rhc]</td>
</tr>
<tr>
<td>pum</td>
<td>[off, rhc]</td>
</tr>
<tr>
<td>dm</td>
<td>[off, rhc]</td>
</tr>
<tr>
<td>mc</td>
<td>[cs, rhc]</td>
</tr>
<tr>
<td>mac</td>
<td>[lab, rhc]</td>
</tr>
</tbody>
</table>

**Comments**

- Planning as a CSP
- Partial Order Planning

---

CS 3793 Artificial Intelligence
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).
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 variable, robot delivers mail.
- **Move**: variable with domain \{mc, mcc, 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 $k$. 
CSP for Delivery Robot

**Planning**

- Forward Planning
- Regression Planning
- Planning as a CSP

**Action Variables**

- \( RLoc_i \) — Rob’s location
- \( RHC_i \) — Rob has coffee
- \( SWC_i \) — Sam wants coffee
- \( MW_i \) — Mail is waiting
- \( RHM_i \) — Rob has mail

**Constraints**

- \( Move_i \) — Rob’s move action
- \( PUC_i \) — Rob picks up coffee
- \( DelC \) — Rob delivers coffee
- \( PUM_i \) — Rob picks up mail
- \( DelM_i \) — Rob delivers mail
Example Constraints

### Planning

- Forward Planning
- Regression Planning
- Planning as a CSP
- Action Variables
- Constraints
- CSP for Robot
- Example
- Constraints
- Partial Order Planning

### Precondition Constraint

<table>
<thead>
<tr>
<th>$RHC_i$</th>
<th>$RLoc_i$</th>
<th>$PUC_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>cs</td>
<td>true</td>
</tr>
<tr>
<td>any</td>
<td>any</td>
<td>false</td>
</tr>
</tbody>
</table>

### Effect Constraint

<table>
<thead>
<tr>
<th>$RHC_i$</th>
<th>$DC_i$</th>
<th>$PUC_i$</th>
<th>$RHC_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

### Action Constraint

<table>
<thead>
<tr>
<th>$Move_i$</th>
<th>$PUC_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>nm</td>
<td>true</td>
</tr>
<tr>
<td>any</td>
<td>false</td>
</tr>
</tbody>
</table>
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 \((act_0 < act_1\) represents \(act_0\) before \(act_1\)). 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

\begin{align*}
\text{start} & \leftarrow \text{pseudo-action with } s \text{ as effect} \\
\text{finish} & \leftarrow \text{pseudo-action with } g \text{ as precondition}
\end{align*}

insert plan $\text{start} < \text{finish}$ into $\text{Frontier}$

while $\text{Frontier}$ is not empty

\begin{align*}
p & \leftarrow \text{remove a plan from } \text{Frontier} \\
\text{if } p \text{ has no flaws then return } p
\end{align*}

select a flaw $w$ in $p$

for each fix $x$ for $w$ in $p$

\begin{align*}
p' & \leftarrow \text{copy of } p \text{ including } x \\
\text{insert } p' \text{ into } \text{Frontier}
\end{align*}

return null
Initial Plan

Planning
Forward Planning
Regression Planning
Planning as a CSP
Partial Order Planning
Idea
Partial Plan Search
   ▶ Initial Search
Intermediate
Flaws
Almost Done
Flaws
Comments

\[
\text{start} \quad cs, rhc, swc, mw, rhm \quad \text{lab, swc} \quad \text{finish}
\]

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

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

Planning
Forward Planning
Regression Planning
Planning as a CSP
Partial Order Planning
Idea
Partial Plan Search
Initial Search
Intermediate
Flaws
Almost Done
Flaws
Comments

start

\(cs, rhc, swc, mw, rhm\)

off

move

lab

causal link

finish

lab, swc

dc

off, rhc

swc

causal link
Flaws in Intermediate Plan

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

flaw: open precondition: \( \text{off of dc} \) not achieved
fix: use same \text{move from cs to off}

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

flaw: open preconditions of \text{puc} and new \text{move}
fix: use effects of \text{start} (use initial state)
Almost Done Plan

Planning
Forward Planning
Regression Planning
Planning as a CSP
Partial Order Planning
Idea
Partial Plan Search
Initial Search
Intermediate
Flaws
Almost Done
Flaws
Comments

start

\( cs, rhc, swc, mw, rhm \)

\( cs \)  \( cs, rhc \)

move

off

off

off

move

lab

lab

lab, swc

finish

cs, rhc

puc

rhc

rhc

off, rhc

dc

swc

swc
Flaws in Almost Done Plan

- **flaw:** conflict: the first \textit{move} conflicts with \textit{cs} staying true between \textit{start} and \textit{puc}
- **fix:** order first \textit{move} after \textit{puc}

- **flaw:** conflict: the second \textit{move} conflicts with \textit{off} staying true between first \textit{move} and \textit{dc}.
- **fix:** order second \textit{move} after \textit{dc}
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.