Planning Problems

Want a sequence of actions to turn a start state into a goal state

Unlike generic search, states and actions have internal structure, which allows better search methods

This slide deck courtesy of Dan Klein at UC Berkeley
State Space

Representation
States described by propositions or ground predicates
Sparse encoding (database semantics): all unstated literals are false
Unique names: each object has its own single symbol

On(C, A)
On(A, Table)
On(B, Table)
Clear(C)
Clear(B)
Actions

On(C, A)
On(A, Table)
On(B, Table)
Clear(C)
Clear(B)

ACTION: Move(b,x,y)
PRECONDITIONS: On(b,x), Clear(b), Clear(y)
POSTCONDITIONS: On(b,y), Clear(x)
\neg On(b,x), \neg Clear(y)

ACTION: Move(C,A,Table)
PRECONDITIONS: On(C,A), Clear(C), Clear(Table)
POSTCONDITIONS: On(C,Table), Clear(A)
\neg On(C,A), \neg Clear(Table)
Actions

On(C, A)
On(A, Table)
On(B, Table)
Clear(C)
Clear(B)

ACTION: MoveToBlock(b,x,y)
PRECONDITIONS: On(b,x), Clear(b), Clear(y),
Block(b), Block(y), (b≠x), (b≠y), (x≠y)
POSTCONDITIONS: On(b,y), Clear(x)
¬On(b,x), ¬Clear(y)

ACTION: MoveToTable(b,x)
PRECONDITIONS: On(b,x), Clear(b), Block(b), Block(x), (b≠x)
POSTCONDITIONS: On(b,Table), Clear(x)
¬On(b,x)
Start and Goal States

Start State

On(C, A)
On(A, Table)
On(B, Table)
Clear(C)
Clear(B)
Block(A)
...

Goal State

On(B, C)
On(A, B)

Important: goal satisfied by any state which entails goal list

[MoveToTable(C,A), Move(B,Table,C), Move(A,Table,B)]
Planning Problems

Action schema, instantiates to give specific ground actions

ACTION: MoveToTable(b,x)
PRECONDITIONS: On(b,x), Clear(b), Block(b), Block(x), (b ≠ x)
POSTCONDITIONS: On(b,Table), Clear(x) ¬On(b,x)
Practice

Problem 10.2: “Applicable”
Problem 10.3a,b: Representation
Where do they come from?
Could they be learned?
Kinds of Plans

**Start State**
- On(C, A)
- On(A, Table)
- On(B, Table)
- Clear(C)
- Clear(B)
- Block(A)
- ...

**Sequential Plan**
- MoveToTable(C,A) > Move(B,Table,C) > Move(A,Table,B)

**Partial-Order Plan**
- MoveToTable(C,A)
- Move(B,Table,C)
- > Move(A,Table,B)
Forward Search

Start State

On(C, A)
On(A, Table)
On(B, Table)
Clear(C)
Clear(B)
Block(A)
...

MoveToTable(C,A)
MoveToBlock(C,A,B)
MoveToBlock(B,Table,C)

On(C, A)
On(A, Table)
On(B, Table)
Clear(C)
Clear(B)
Block(A)
...
+Clear(A)
+On(C, Table)

Applicable actions
Backward Search

ACTION: MoveToBlock(b,x,y)
PRECONDITIONS: On(b,x), Clear(b), Clear(y),
Block(b), Block(y), (b ≠ x), (b ≠ y),
(x ≠ y)
POSTCONDITIONS: On(b,y), Clear(x)
¬On(b,x), ¬Clear(y)

MoveToBlock(A,Table,B)
MoveToBlock(A,x',B)

On(B, C)
On(A, B)
+On(A,Table)
+Clear(A)
+Clear(B)
+...

\[ g' = (g - ADD(a)) \cup Precond(a) \]
Heuristics: Ignore Preconditions

Relax problem by ignoring preconditions

Can drop all or just some preconditions
Can solve in closed form or with set-cover methods

Action(Slide(t, s₁, s₂),
  PRECOND: On(t, s₁) ∧ Tile(t) ∧ Blank(s₂) ∧ Adjacent(s₁, s₂)
  EFFECT: On(t, s₂) ∧ Blank(s₁) ∧ ¬On(t, s₁) ∧ ¬Blank(s₂))
Heuristics: No-Delete

Relax problem by not deleting falsified literals

Can’t undo progress, so solve with hill-climbing (non-admissible)

ACTION: MoveToBlock(b, x, y)
PRECONDITIONS: On(b, x), Clear(b), Clear(y), Block(b), Block(y), (b ≠ x), (b ≠ y), (x ≠ y)
POSTCONDITIONS: On(b, y), Clear(x)
    ¬On(b, x), ¬Clear(y)
Heuristics: Independent Goals

Independent subgoals?

Partition goal literals
Find plans for each subset
cost(all) < cost(any)?
cost(all) < sum-cost(each)?
Planning “Tree”

Start: HaveCake

Goal: AteCake, HaveCake

Action: Eat
Pre: HaveCake
Add: AteCake
Del: HaveCake

Action: Bake
Pre: ¬HaveCake
Add: HaveCake

{Eat} 

Have=T, Ate=F

{Eat}

Have=F, Ate=T

{Bake}

Have=T, Ate=T

{Eat}

Have=F, Ate=T

{Eat}

Have=F, Ate=F

{Eat}

Have=T, Ate=F

{Eat}

Have=F, Ate=T
Reachable State Sets

Have=T, Ate=F

\{Eat\} \quad \{} \quad \{} \quad \{Eat\} \quad \{} 

Have=F, Ate=T \quad \text{Have=T, Ate=F} 

\{Bake\} \quad \{} \quad \{} 

\text{Have=T, Ate=T} \quad \text{Have=F, Ate=T} \quad \text{Have=F, Ate=T} \quad \text{Have=T, Ate=F}

\text{Have=F, Ate=T} \quad \text{Have=F, Ate=T} \quad \text{Have=T, Ate=F}
Approximate Reachable Sets

Have=T,
Ate=F

Have={T},
Ate={F}

Have=F,
Ate=T
Have=T,
Ate=F

Have={T,F},
Ate={T,F}

(Have, Ate) not (T,T)
(Have, Ate) not (F,F)

Have=F,
Ate=T
Have=T,
Ate=F

Have={T,F},
Ate={T,F}

(Have, Ate) not (F,F)
Planning Graphs

Start: HaveCake

Goal: AteCake, HaveCake

Action: Eat
Pre: HaveCake
Add: AteCake
Del: HaveCake

Action: Bake
Pre: \neg HaveCake
Add: HaveCake

\begin{align*}
S_0 &\quad \quad A_0 &\quad \quad S_1 \\
\text{HaveCake} &\quad \quad \neg \text{AteCake} &\quad \quad \neg \text{AteCake} \\
&\quad \quad \neg \text{HaveCake} &\quad \quad \text{HaveCake} \\
&\quad \quad \text{AteCake} &\quad \quad \text{HaveCake} \\
\end{align*}
Mutual Exclusion (Mutex)

NEGATION

Literals and their negations can’t be true at the same time

\[ P \quad \neg P \]

\[ \neg AteCake \quad AteCake \]

\[ \neg HaveCake \quad HaveCake \]

\[ S_0 \quad A_0 \quad S_1 \]
Mutual Exclusion (Mutex)

**INCONSISTENT EFFECTS**
An effect of one negates the effect of the other.
Mutual Exclusion (Mutex)

**INCONSISTENT SUPPORT**
All pairs of actions that achieve two literals are mutex

![Diagram showing mutual exclusion](image)

- **HaveCake**
- **Eat**
- **AteCake**
- **¬AteCake**

States:
- **S₀**
- **A₀**
- **S₁**
Planning Graph
Mutual Exclusion (Mutex)

COMPETITION
Preconditions are mutex; cannot both hold

INCONSISTENT EFFECTS
An effect of one negates the effect of the other
Mutual Exclusion (Mutex)

INTERFERENCE
One deletes a precondition of the other

\[ S_1 \]
\[ \neg \text{HaveCake} \]
\[ \text{Eat} \]
\[ \neg \text{HaveCake} \]
\[ \neg \text{AteCake} \]

\[ A_1 \]
\[ \text{Bake} \]
\[ \text{Sell} \]
\[ \text{HaveCake} \]
\[ \neg \text{HaveCake} \]
\[ \neg \text{AteCake} \]

\[ S_2 \]
Planning Graph
Observation 1

Propositions monotonically increase
(always carried forward by no-ops)
Observation 2

Actions monotonically increase
(if they applied before, they still do)
Observation 3

Proposition mutex relationships monotonically decrease
Observation 4

Action mutex relationships monotonically decrease
Observation 5

Claim: planning graph “levels off”

After some time $k$ all levels are identical

Because it’s a finite space, the set of literals cannot increase indefinitely, nor can the mutexes decrease indefinitely

Claim: if goal literal never appears, or goal literals never become non-mutex, no plan exists

If a plan existed, it would eventually achieve all goal literals (and remove goal mutexes – less obvious)

Converse not true: goal literals all appearing non-mutex does not imply a plan exists
Planning graphs enable powerful heuristics

Level cost of a literal is the smallest $S$ in which it appears.

Max-level: goal cannot be realized before largest goal conjunct level cost (admissible).

Sum-level: if subgoals are independent, goal cannot be realized faster than the sum of goal conjunct level costs (not admissible).

Set-level: goal cannot be realized before all conjuncts are non-mutex (admissible).
Graphplan directly extracts plans from a planning graph

Graphplan searches for **layered plans** (often called parallel plans)

More general than totally-ordered plans, less general than partially-ordered plans

A layered plan is a sequence of sets of actions

actions in the same set must be compatible

all sequential orderings of compatible actions gives same result

Layered Plan: (a two layer plan)

\[
\begin{align*}
\{ \text{move}(A,B,\text{TABLE}) \} \cdot \{ \text{move}(B,\text{TABLE},A) \} \\
\{ \text{move}(C,D,\text{TABLE}) \} \cdot \{ \text{move}(D,\text{TABLE},C) \}
\end{align*}
\]
Solution Extraction: Backward Search

Search problem:
- Start state: goal set at last level
- Actions: conflict-free ways of achieving the current goal set
- Terminal test: at $S_0$ with goal set entailed by initial planning state

Note: may need to start much deeper than the leveling-off point!

Caching, good ordering is important
Scheduling

In real planning problems, actions take time, resources
   Actions have a duration (time to completion, e.g. building)
   Actions can consume (or produce) resources (or both)
   Resources generally limited (e.g. minerals, SCVs)

Simple case: known (partial) plan, just need to schedule

Even simpler: no resources, just ordering and duration

<table>
<thead>
<tr>
<th>JOBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[AddEngine1 &lt; AddWheels1 &lt; Inspect1]</td>
</tr>
<tr>
<td>[AddEngine2 &lt; AddWheels2 &lt; Inspect2]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESOURCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>EngineHoists (1)</td>
</tr>
<tr>
<td>WheelStations (1)</td>
</tr>
<tr>
<td>Inspectors (2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddEngine1: DUR=30, USE=EngHoist(1)</td>
</tr>
<tr>
<td>AddEngine2: DUR=60, USE=EngHoist(1)</td>
</tr>
<tr>
<td>AddWheels1: DUR=30, USE=WStation(1)</td>
</tr>
<tr>
<td>AddWheels2: DUR=15, USE=WStation(1)</td>
</tr>
<tr>
<td>Inspect1: DUR=10, USE=Inspectors(1)</td>
</tr>
<tr>
<td>Inspect2: DUR=10, USE=Inspectors(1)</td>
</tr>
</tbody>
</table>
Resource-Free Scheduling

How to minimize total time?

Easy: schedule an action as soon as its parents are completed

\[ ES(START) = 0 \]
\[ ES(a) = \max_{b: b \prec a} ES(b) + DUR(b) \]

Result:

**JOBS**
- AddEngine1 < AddWheels1 < Inspect1
- AddEngine2 < AddWheels2 < Inspect2

**RESOURCES**
- EngineHoists (1)
- WheelStations (1)
- Inspectors (2)

**ACTIONS**
- AddEngine1: DUR=30, USE=EngHoist(1)
- AddEngine2: DUR=60, USE=EngHoist(1)
- AddWheels1: DUR=30, USE=WStation(1)
- AddWheels2: DUR=15, USE=WStation(1)
- Inspect1: DUR=10, USE=Inspectors(1)
- Inspect2: DUR=10, USE=Inspectors(1)
Resource-Free Scheduling

Note there is always a critical path
All other actions have slack
Can compute slack by computing latest start times

\[
LS(\text{END}) = ES(\text{END}) \\
LS(a) = \min_{b:a<b} LS(b) - DUR(a)
\]

Result:

\begin{align*}
&\text{Engine1: DUR=30, USE=EngHoist(1)} \\
&\text{Engine2: DUR=60, USE=EngHoist(1)} \\
&\text{Wheel1: DUR=30, USE=WStation(1)} \\
&\text{Wheel2: DUR=15, USE=WStation(1)} \\
&\text{Inspect1: DUR=10, USE=Inspectors(1)} \\
&\text{Inspect2 DUR=10, USE=Inspectors(1)}
\end{align*}
Adding Resources

For now: consider only released (non-consumed) resources

View start times as variables in a CSP

Before: conjunctive linear constraints

\[ \forall b : b \prec a \quad ES(a) \geq ES(b) + DUR(b) \]

Now: disjunctive constraints (competition)

\[
\text{if competing}(a, b) \\
ES(a) \geq ES(b) + DUR(b) \lor \\
ES(b) \geq ES(a) + DUR(a)
\]

In general, no efficient method for solving optimally
Adding Resources

One greedy approach: min slack algorithm

- Compute ES, LS windows for all actions
- Consider actions which have all preconditions scheduled
- Pick the one with least slack
- Schedule it as early as possible
- Update ES, LS windows (recurrences now must avoid reservations)
Resource Management

Complications:

- Some actions need to happen at certain times
- Consumption and production of resources
- Planning and scheduling generally interact
Prodigy

- A classical STRIPS-style planner
  - Domain Representation: objects, operators
  - Problem Representation: initial state, goal state

- Operators have preconditions and effects
Example – Blocksworld

<table>
<thead>
<tr>
<th>(On A B)</th>
<th>Initial State</th>
<th>Goal State</th>
</tr>
</thead>
<tbody>
<tr>
<td>(On B Table)</td>
<td></td>
<td>(On C A)</td>
</tr>
<tr>
<td>(On C Table)</td>
<td></td>
<td>(On B C)</td>
</tr>
<tr>
<td>(Clear A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Clear C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Clear Table)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Arm–empty)</td>
<td></td>
<td>[+] whatever</td>
</tr>
</tbody>
</table>

Operators:  
(Pickup x)  
preconds: (Clear x)  
(Arm–empty)  
adds: (Holding x)  
if (On x y), (Clear y)  
dels: (Arm–empty)  
if (On x y), (On x y)  

(Putdown x y)  
preconds: (Holding x)  
(Clear y)  
adds: (On x y)  
if (y != Table), (Clear y)  
dels: (Holding x)  
(Arm–empty)
Prodigy/Blocksworld (cont.)

[Diagram with blocks A, B, C in different configurations.]

Putdown C A
(Holding C)
(Clear A)

(On C A)
(On B C)
Prodigy/Blocksworld (cont.)

A
B
C

(Clear C)

(Holding B)
Prodigy/Blocksworld (cont.)

Pickup A
(Arm−empty)

Putdown A Table
Prodigy/Blocksworld (cont.)

Putdown A Table
Prodigy/Blocksworld (cont.)

A B C A
B C B C A
A

C
A

B
C A
C
Prodigy/Blocksworld (cont.)
Issues in Planning

- Representations
- Algorithms
- Conditional effects
- Dynamic worlds
- Mixing planning and execution
- Learning
- Large-scale applications

Fairly mature field
Example – Blocksworld

(On A B)
(On B Table)
(On C Table)
(Clear A)
(Clear C)
(Clear Table)
(Arm−empty)

Initial
State

Goal
State

B
C
A

Operators:  (Pickup x)

preconds:  (Clear x)
          (Arm−empty)

adds:    (Holding x)
         if (On x y), (Clear y)

dels:    (Arm−empty)
         if (On x y), (On x y)

(Putdown x y)

preconds:  (Holding x)
          (Clear y)

adds:    (On x y)
         (Arm−empty)

dels:    (Holding x)
         if (y != Table), (Clear y)