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
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)
¬On(b,x), ¬Clear(y)

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

Sparse encoding, but complete state spec

Set of goal states, only requirements specified (think unary constraints)

Which goal first?
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(A, Table, B)]
Move(B, Table, C)
Forward Search

Start State

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

Applicable actions

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)
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)
+...

Relevant actions

\[ g' = (g - ADD(a)) \cup Precond(a) \]
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

\[ \text{Action}(\text{Slide}(t, s_1, s_2)), \]

\[ \text{Precond}: On(t, s_1) \land \text{Tile}(t) \land \text{Blank}(s_2) \land \text{Adjacent}(s_1, s_2) \]

\[ \text{Effect}: On(t, s_2) \land \text{Blank}(s_1) \land \neg On(t, s_1) \land \neg \text{Blank}(s_2) \]
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
\[ \text{cost(all)} < \text{cost(any)}? \]
\[ \text{cost(all)} < \text{sum-cost(each)}? \]
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

{Bake}

Have=F, Ate=T

{Eat}

Have=T, Ate=F

Have=F, Ate=T

{}
Reachable State Sets

- Have=T, Ate=F
  - {Eat}
    - Have=F, Ate=T
      - {Bake}
        - Have=T, Ate=T
        - Have=F, Ate=T
    - {Eat}
      - Have=T, Ate=F
      - Have=F, Ate=T
- Have=F, Ate=T
  - Have=T, Ate=F
    - Have=T, Ate=F
Approximate Reachable Sets

- Have=\{T\}, Ate=\{F\}
- Have=\{T,F\}, Ate=\{T,F\}
- (Have, Ate) not (T,T)
- (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
**Mutual Exclusion (Mutex)**

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

\[ P \]

\[ \neg P \]
Mutual Exclusion (Mutex)

INCONSISTENT EFFECTS
An effect of one negates the effect of the other

S₀
HaveCake
¬AteCake

S₁
HaveCake
¬HaveCake
AteCake
¬AteCake

A₀
Eat
Mutual Exclusion (Mutex)

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

$S_0$  $A_0$  $S_1$  $A_1$  $S_2$

- HaveCake
- Eat
- $\neg$ HaveCake
- AteCake
- $\neg$ AteCake

- Bake
- HaveCake
- $\neg$ HaveCake
- AteCake
- $\neg$ AteCake

- $\neg$ AteCake
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

\[ \text{S}_1 \xrightarrow{\neg \text{HaveCake}} \text{HaveCake} \]
\[ \text{S}_1 \xrightarrow{\neg \text{HaveCake}} \neg \text{HaveCake} \]
\[ \text{S}_1 \xrightarrow{\neg \text{AteCake}} \text{AteCake} \]
\[ \text{S}_1 \xrightarrow{\neg \text{AteCake}} \neg \text{AteCake} \]
\[ \text{A}_1 \xrightarrow{\neg \text{HaveCake}} \text{HaveCake} \]
\[ \text{A}_1 \xrightarrow{\neg \text{HaveCake}} \neg \text{HaveCake} \]
\[ \text{A}_1 \xrightarrow{\neg \text{AteCake}} \text{AteCake} \]
\[ \text{A}_1 \xrightarrow{\neg \text{AteCake}} \neg \text{AteCake} \]
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

\[
\begin{align*}
\text{Layered Plan:} & \quad (a \text{ two layer plan}) \\
\{ & \{ \text{move}(A,B,\text{TABLE}) \} \cdot \{ \text{move}(C,D,\text{TABLE}) \} \}
\end{align*}
\]
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

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

Result:

<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>

\[
\begin{array}{c|c|c}
\text{Start} & \text{Engine1} & \text{Wheel1} \\
30 & 30 & 10 \\
\text{Engine2} & \text{Wheel2} & \text{Insp1} \\
60 & 15 & 10 \\
\text{Insp2} & \text{End} & \\
& & \\
\end{array}
\]
Resource-Free Scheduling

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

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

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)
Adding Resources

For now: consider only released (non-consumed) resources
View start times as variables in a CSP
Before: conjunctive linear constraints
\[ \forall b : b < a \quad ES(a) \geq ES(b) + DUR(b) \]
Now: disjunctive constraints (competition)
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