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
Kinds of Plans

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)
...

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: \(\text{On}(b, x), \text{Clear}(b), \text{Clear}(y),\)

\(\text{Block}(b), \text{Block}(y), (b \neq x), (b \neq y),\)

\((x \neq y)\)

POSTCONDITIONS: \(\text{On}(b, y), \text{Clear}(x)\)

\(\neg\text{On}(b, x), \neg\text{Clear}(y)\)

\(\text{MoveToBlock}(A, \text{Table}, B)\)

\(\text{MoveToBlock}(A, x', B)\)

\(\text{On}(B, C)\)
\(\text{On}(A, B)\)
\(+\text{On}(A, \text{Table})\)
\(+\text{Clear}(A)\)
\(+\text{Clear}(B)\)
\(+\ldots\)

\[g' = (g - \text{ADD}(a)) \cup \text{Precond}(a)\]
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: \neg HaveCake
Add: HaveCake

Have=T, Ate=F

{Eat}

Have=F, Ate=T

{Bake}

Have=F, Ate=T

{Eat}

Have=T, Ate=F

{}
Reachable State Sets

Have=T, Ate=F

{Eat}  {}

Have=F, Ate=T

{Bake}  {}

Have=F, Ate=T

Have=T, Ate=T

Have=T, Ate=F

Have=F, Ate=T

Have=F, Ate=T

Have=T, Ate=F

Have=T, Ate=T

Have=F, Ate=T

Have=T, Ate=F

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={T,F}, Ate={T,F}
- (Have, Ate) not (F,F)
- Have=T, Ate=T
- 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: ¬HaveCake
Add: HaveCake

S₀    A₀    S₁
Mutual Exclusion (Mutex)

NEGATION

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

P
¬P

HaveCake

Eat

HaveCake
¬HaveCake

AteCake
¬AteCake

S₀
A₀
S₁
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.
Planning Graph
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

\[ S_1 \]

- HaveCake
- \( \neg \)HaveCake
- AteCake
- \( \neg \)AteCake

\[ A_1 \]

- Sell
- Bake
- Eat
- AteCake

\[ S_2 \]
Planning Graph

S0
HaveCake
¬AteCake
A0
Eat
Haven't AteCake
S1
HaveCake
¬HaveCake
AteCake
¬AteCake
A1
Eat
Bake
HaveCake
¬HaveCake
AteCake
¬AteCake
S2
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

Diagram of mutex relationships between propositions p, q, and r.
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
Heuristics: Level Costs

- 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

- 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

```
move(A,B,TABLE)
move(C,D,TABLE)
move(B,TABLE,A)
move(D,TABLE,C)
```

**Layered Plan:** (a two layer plan)

\[
\left\{ \text{move}(A,B,\text{TABLE}) \right\} \cdot \left\{ \text{move}(B,\text{TABLE},A) \right\} \left\{ \text{move}(C,D,\text{TABLE}) \right\} \cdot \left\{ \text{move}(D,\text{TABLE},C) \right\}
\]
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

```plaintext
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

- 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:

**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

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

<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>

**Result:**

![Resource-Free Scheduling Diagram](image-url)
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)
  \[
  \begin{align*}
  &\text{if competing}(a, b) \\
  &ES(a) \geq ES(b) + DUR(b) \lor \\
  &ES(b) \geq ES(a) + DUR(a)
  \end{align*}
  \]
- 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