### Planning as Search

<table>
<thead>
<tr>
<th>State Space</th>
<th>Plan Space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm</strong></td>
<td><strong>Progression</strong></td>
</tr>
<tr>
<td><strong>Node</strong></td>
<td><strong>World State</strong></td>
</tr>
<tr>
<td><strong>Edge</strong></td>
<td><strong>Apply Action</strong></td>
</tr>
<tr>
<td></td>
<td>If prec satisfied,</td>
</tr>
<tr>
<td></td>
<td>Add adds,</td>
</tr>
<tr>
<td></td>
<td>Delete deletes</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Partial Plans
Expressive action representation: UCPOP

- **Negated goals:**
  - Same as positive goals
  - CWA for initial state (i.e. assume false if prop. not present)

- **Actions with variables:**
  - Use unification instead of matching
  - Maintain Bindings in Partial plan

- **Conditional effects:**
  - If conditional effect used for causal links, achieve antecedent
  - Threat resolution by “confrontation”, i.e., negate antecedent

- **Disjunctive preconditions:**
  - Choose one to work on

- **Universal quantification:**
  - Assume finite, static universe → finite universal base (UB)
  - To achieve universally quantified precondition, achieve its UB
  - Use effect literal from UB, to satisfy goal (incrementally expand UB)
  - Consider threats from universally quantified variables.
GraphPlan

- Planning graph
  - Encodes constraints on possible plans
    - Alternate proposition and action node layers
      - connected by preconditions and effect edges
    - Mutual exclusion constraints
  - Polynomial-time construction
  - Constrains search for a valid plan
- Finds “shortest parallel plan”
- Sound, complete and will terminate with failure if there is no plan
Mutual Exclusion relations

Inconsistent Effects

Competing Needs

Interference (prec-effect)

Inconsistent Support
GraphPlan algorithm

- Grow the planning graph (PG) until all goals are reachable and not mutex. (If PG levels off first, fail)
- Search the PG for a valid plan
- If non found, add a level to the PG and try again
If goals are present & non-mutex:
Choose action to achieve each goal
Add preconditions to next goal set
Planning as $X, X \in \{\text{SAT, CSP, ILP, …}\}$

- Compile planning into a computational substrate that is (at least) NP-hard.

- Planning as:
  - **SAT**: Propositional Satisfiability
    - OBDD: Ordered Binary Decision Diagrams (Cimatti et al, 98)
  - **CSP**: Constraint Satisfaction
    - GP-CSP (Do & Kambhampati 2000)
  - **ILP**: Integer Linear Programming
    - Kautz & Walser 1999, Vossen et al 2000
  - ...

...
Planning as SAT

- Bounded-length planning can be formalized as propositional satisfiability (SAT)
- Plan = model (truth assignment) that satisfies logical constraints representing:
  - Initial state
  - Goal state
  - Domain axioms: actions, frame axioms, ...

for a fixed plan length

- Logical spec such that **any** model is a valid plan
Architectures of a SAT-based planner

Problem Description
- Init State
- Goal State
- Actions

Compiler (encoding)

Simplifier (polynomial inference)

Solver (SAT engine/s)

Plan

Decoder

Satisfying model

Increment plan length
If unsatisfiable

CNF

mapping
Graphplan-based Encoding

- Goal holds at last layer
- Initial state holds at first layer
- Fact $\Rightarrow$ Act1 $\lor$ Act2
- Act1 $\Rightarrow$ Pre1 $\land$ Pre2
- $\neg$Act1 $\lor$ $\neg$Act2

[Kautz & Selman AAAI 96]
Algorithms for SAT

- Systematic (Complete: prove sat and unsat)
  - Davis-Putnam (1960)
  - DPLL (Davis Logemann Loveland, 1962)
  - Satz (Li & Anbulagan 1997)
  - Rel-Sat (Bayardo & Schrag 1997)
  - Chaff (Moskewicz et al 2001; Zhang & Malik CADE 2002)

- Stochastic (incomplete: cannot prove unsat)
  - GSAT (Selman et al 1992)
  - Walksat (Selman et al 1994)

- Randomized Systematic
  - Randomized Restarts (Gomes et al 1998)
    - Cutoff and restart search after a fixed number of backtracks
      → Provably Eliminates heavy tails
Representing the Planning Graph as a CSP

(a) Planning Graph

(b) DCSP

Variables: $G_1, \ldots, G_4, P_1 \ldots P_6$

Domains:
- $G_1: \{A_1\}$
- $G_2: \{A_2\}$
- $G_3: \{A_3\}$
- $G_4: \{A_4\}$
- $P_1: \{A_5\}$
- $P_2: \{A_6, A_{11}\}$
- $P_3: \{A_7\}$
- $P_4: \{A_8, A_9\}$
- $P_5: \{A_{10}\}$
- $P_6: \{A_{10}\}$

Constraints (normal):
- $P_1 = A_5 \Rightarrow P_4 \neq A_9$
- $P_2 = A_6 \Rightarrow P_4 \neq A_8$
- $P_2 = A_{11} \Rightarrow P_3 \neq A_7$

Constraints (Activity):
- $G_1 = A_1 \Rightarrow Active\{P_1, P_2, P_3\}$
- $G_2 = A_2 \Rightarrow Active\{P_4\}$
- $G_3 = A_3 \Rightarrow Active\{P_5\}$
- $G_4 = A_4 \Rightarrow Active\{P_1, P_6\}$

Init State: $Active\{G_1, G_2, G_3, G_4\}$
Transforming a DCSP to a CSP

Variables: $G_1, \ldots, G_4, P_1 \ldots P_6$

Domains:
- $G_1: \{A_1\}$
- $G_2: \{A_2\}$
- $G_3: \{A_3\}$
- $G_4: \{A_4\}$
- $P_1: \{A_5\}$
- $P_2: \{A_6, A_{11}\}$
- $P_3: \{A_7\}$
- $P_4: \{A_8, A_9\}$
- $P_5: \{A_{10}\}$
- $P_6: \{A_{10}\}$

Constraints (normal):
- $P_1 = A_5 \Rightarrow P_4 \neq A_9$
- $P_2 = A_6 \Rightarrow P_4 \neq A_8$
- $P_2 = A_{11} \Rightarrow P_3 \neq A_7$

Constraints (Activity):
- $G_1 = A_1 \Rightarrow \text{Active}\{P_1, P_2, P_3\}$
- $G_2 = A_2 \Rightarrow \text{Active}\{P_4\}$
- $G_3 = A_3 \Rightarrow \text{Active}\{P_5\}$
- $G_4 = A_4 \Rightarrow \text{Active}\{P_1, P_6\}$

Init State: $\text{Active}\{G_1, G_2, G_3, G_4\}$

(a) DCSP

Variables: $G_1, \ldots, G_4, P_1 \ldots P_6$

Domains:
- $G_1: \{A_1, \bot\}$
- $G_2: \{A_2, \bot\}$
- $G_3: \{A_3, \bot\}$
- $G_4: \{A_4, \bot\}$
- $P_1: \{A_5, \bot\}$
- $P_2: \{A_6, A_{11}, \bot\}$
- $P_3: \{A_7, \bot\}$
- $P_4: \{A_8, A_9, \bot\}$
- $P_5: \{A_{10}, \bot\}$
- $P_6: \{A_{10}, \bot\}$

Constraints (normal):
- $P_1 = A_5 \Rightarrow P_4 \neq A_9$
- $P_2 = A_6 \Rightarrow P_4 \neq A_8$
- $P_2 = A_{11} \Rightarrow P_3 \neq A_7$

Constraints (Activity):
- $G_1 = A_1 \Rightarrow P_1 \neq \bot \land P_2 \neq \bot \land P_3 \neq \bot$
- $G_2 = A_2 \Rightarrow P_4 \neq \bot$
- $G_3 = A_3 \Rightarrow P_5 \neq \bot$
- $G_4 = A_4 \Rightarrow P_1 \neq \bot \land P_6 \neq \bot$

Init State: $G_1 \neq \bot \land G_2 \neq \bot \land G_3 \neq \bot \land G_4 \neq \bot$

(b) CSP
HTN Planning

- Capture hierarchical structure of planning domain
- Non-primitive actions and Reduction schemas:
  - Expert knowledge: preferred ways to accomplish a task
  - Reduction schemas: (task, task-network)
- Task Reduction: another plan refinement
- Task hierarchy \(\sim\) context-free grammar
  - Prune plans that do not conform to the grammar in a Partial-Order planner [Barret & Weld, AAAI94]
Task Reduction
Basic HTN Procedure

1. Input a planning problem P
2. If P contains only primitive tasks, then resolve the conflicts and return the result. If the conflicts cannot be resolved, return failure
3. Choose a non-primitive task t in P
4. Choose an expansion for t
5. Replace t with the expansion
6. Find interactions among tasks in P and suggest ways to handle them. Choose one.
7. Go to 2
Refinement Planning

[Kambhampati 96]
OBDD-based Planning

Action 0
Pre: \( \neg x_1 \land \neg x_2 \)
Eff: \( \neg x'_1 \)

Action 1
Pre: \( x_1 \)
Eff: \( (x_1 \land \neg x_2 \rightarrow \neg x'_1) \land (x_1 \land x_2 \rightarrow x'_1) \)

FSM

\[ \begin{align*}
&00 \quad 01 \\
&01 \quad 10 \\
&10 \quad 11
\end{align*} \]

\( P_G(a,x_1,x_2) \)

\[ \begin{align*}
P_V(a,x_1,x_2) &= \exists x'_1, x'_2 \cdot T(a,x_1,x'_1,x_2,x'_2) \land V(x'_1, x'_2) \\
P_G(a,x_1,x_2) &= \exists x'_1, x'_2 \cdot T(a,x_1,x'_1,x_2,x'_2) \land (\neg x'_1 \land x'_2)
\end{align*} \]
Weak, Strong and Strong Cyclic Reachability Goals

1. Weak Solutions: plans that may achieve the goal

2. Strong Solutions: plans that are guaranteed to achieve the goal

3. Strong Cyclic solutions: iterative trial-and-error strategies whose executions always have a possibility of terminating and, when they do, they are guaranteed to achieve the goal.
Weak Solutions

Given: A planning domain $D$ and problem $P$
A plan $\pi$ and the Induced Kripke Structure $K$

The plan $\pi$ is a weak solution to the planning problem $P$ if for all $s \in I$, $K, s \models EF G$.

Planning domain

1. unlock
2. lock
3. unload
4. unlock

plan = \{ <2,load>, <3,lock> \}

Induced Kripke Structure:
**Strong Solutions**

Given: A planning domain $D$ and problem $P$

A plan $\pi$ and the Induced Kripke Structure $K$

The plan $\pi$ is a strong solution to the planning problem $P$ if for all $s \in I$, $K, s \models \text{AF } G$.

Planning domain

```
1  unlock  2  load  5  fix  lock
   lock    unload  fix   unlock

plan = \{ <2,load>, <3,lock>, <5,fix> \}

Induced Kripke Structure:
```

```
Strong Cyclic Solutions

Given: A planning domain $D$ and problem $P$
A plan $\pi$ and the Induced Kripke Structure $K$

The plan $\pi$ is a strong cyclic solution to the planning problem $P$ if for all $s \in I$, $K, s \models \text{AGEF } G$.

Plan set: $\{\langle 2, \text{load} \rangle, \langle 3, \text{lock} \rangle, \langle 5, \text{fix} \rangle\}$

Induced Kripke Structure:
Planning Decision Problems

- **Plan Existence (PLANSAT):**
  - Given a planning problem instance \( P = (I, O, G) \),
  - Is there a plan that achieves goals \( G \) from initial state \( I \) using operators from \( O \)?

- **Plan Length (PLANMIN):**
  - Given a planning problem instance \( P = (I, O, G) \) and an integer \( k \) (encoded in binary),
  - Is there a plan that achieves goals \( G \) from initial state \( I \) using less than \( k \) operators from \( O \)?
Complexity of Domain-independent Planning

- Undecidable if function symbols allowed

Complexity bounds (decidable case):

- With no restrictions: EXPSPACE
  - Search through all states
  - Each state consumes exponential space
- No delete lists: NEXP
  - Operators only need to appear once
  - Choose among exponentially-many operators
- No negative preconds and no deletes: EXP
  - Plans for different subgoals won't negatively interfere with each other => order does not matter (no choose)
Propositional Planning

- Propositions = 0-ary predicates
- State has $p$ propositions (polynomial)
- Possible States = Powerset{$p$} = $2^p$ (single! exponential)
- Number of Operators is also polynomial

$\Rightarrow$ Reduced complexity:
  - General case: from EXPSPACE to PSPACE
  - No deletes: from NEXP to NP
  - No deletes and no negative preconds: from EXP to P

If you know the operators in advance, this in effect bounds the arity of predicates and operators, with the same result
What does all this mean?

- Domain-independent planning in general is very hard: PSPACE, NP, ...
- Even for very restricted cases:
  - 2 positive preconds, 2 effects (PSPACE)
  - 1 precond, 1 positive effect (NP)
  ... in the worst case ...
- What about the average case, structured domains, real-world problem distributions?
=> Heuristics, reuse solutions, learning
Planning, Execution, and Information Gathering

Things go wrong

**Incomplete information**

Unknown preconditions, e.g., \( \text{Intact}(\text{Spare}) \)?

Disjunctive effects, e.g., \( \text{Inflate}(x) \) causes
\[
\text{Inflated}(x) \lor \text{SlowHiss}(x) \lor \text{Burst}(x) \lor \text{BrokenPump} \lor \ldots
\]

**Incorrect information**

Current state incorrect, e.g., spare NOT intact
Missing/incorrect postconditions in operators

**Qualification problem:**

can never finish listing all the required preconditions and possible conditional outcomes of actions
Solutions

Conformant or sensorless planning
Devise a plan that works regardless of state or outcome
Such plans may not exist

Conditional planning
Plan to obtain information (observation actions)
Subplan for each contingency, e.g.,
\[
[\text{Check}(\text{Tire}_1), \text{if } \text{Intact}(\text{Tire}_1) \text{ then } \text{Inflate}(\text{Tire}_1) \text{ else } \text{Call AAA}]
\]
Expensive because it plans for many unlikely cases

Monitoring/Replanning
Assume normal states, outcomes
Check progress during execution, replan if necessary
Unanticipated outcomes may lead to failure (e.g., no AAA card)

(Really need a combination; plan for likely/serious eventualities, deal with others when they arise, as they must eventually)
Execution Monitoring

“Failure” = preconditions of *remaining plan* not met

Preconditions of remaining plan
  = all preconditions of remaining steps not achieved by remaining steps
  = all causal links *crossing* current time point

On failure, resume POP to achieve open conditions from current state

IPEM (Integrated Planning, Execution, and Monitoring):
  keep updating *Start* to match current state
  links from actions replaced by links from *Start* when done
Sample Conditional Plan

Start

On(Tire1)
Flat(Tire1)
Inflated(Spare)

Check(Tire1)

¬Intact(Tire1)

Remove(Tire1)
(¬Intact(Tire1))

Intact(Tire1)

Flat(Tire1)

Inflate(Tire1)
(¬Intact(Tire1))

On(Spare)
Inflated(Spare)

Puton(Spare)
(¬Intact(Tire1))

Finish
(¬Intact(Tire1))

Finish
(Intact(Tire1))

On(Tire1)

Inflated(Tire1)