State-Space Planning (Recap)

- Planning as state-space search
  - Forward (progression) from initial state
  - Backward (regression) from goal
- Action is *fully specified* at time of introduction to plan
  - All variables bound to specific objects
  - Ordering in sequence with existing actions
GraphPlan

- Blum & Furst, 1995.
- Radical reformulation of plan search space
  - Applies to propositional encodings
    - Must propositionalize knowledge
  - Compacted state space with plan graph rep’n
  - Demonstrated dramatic performance improvements

Planning Graphs

- Interleave levels of literals and actions, related by support, precondition, and mutex

```
<table>
<thead>
<tr>
<th>0</th>
<th>t-1</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Propositions true at time 0              Propositions true at time t-1
Actions                                          Propositions achievable at time t
GraphPlan Algorithm

- zeroth level ← initial state
- Repeat until solution or “level off”:
  - Extend planning graph by considering application of actions
  - Attempt to extract solution

Example: “Dinner Date”

- Object: Take out garbage, fix dinner, wrap present.
- Goal: \( \neg \text{garb} \land \text{dinner} \land \text{present} \)
- Actions: \((\text{act \ [preconds] \ [effects]}))\)
  - cook [cleanH] [dinner]
  - wrap [quiet] [present]
  - carry [] [\( \neg \text{garb} \land \neg \text{cleanH} \)]
  - dolly [] [\( \neg \text{garb} \land \neg \text{quiet} \)]
- Initial: \( \text{garb} \land \text{cleanH} \land \text{quiet} \)

from Weld, “Recent advances in AI planning”, AI Magazine, 1999.
Constructing the Plan Graph

Propositions true at time 0

Mutex
- Actions
  - Inconsistent effects
    - Effects conflict
  - Interference
    - Effects conflict with Preconditions
  - Competing needs
    - Preconditions conflict
- Propositions
  - Complements
  - Inconsistent support
    - Actions that generate propositions are mutex

Solution?

- Can achieve \(\neg\text{garb}, \text{dinner}, \text{present}\) separately
- Two “solns”:
  - \{carry, cook, wrap\}
  - \{dolly, cook, wrap\}
- But:
  - \text{mutex}(\text{carry, cook})
  - \text{mutex}(\text{dolly, wrap})
Extend the Plan Graph

Solution!
Graph Plan: Summary

- We still have to do a TreeSearch to extract a solution from the graph.
  - So why did we build the graph in the first place?

Planning in the real world

- Major challenges:
  - Major parts of the world are unknown
    - How do we generate plans when we don’t know what’s around the corner?
  - Very large search spaces
    - Continuous-valued actions
    - Interactions of multiple agents
Conditional Planning

- Uncertainty at plan construction time may be resolved by execution time
  - Example: shopping plan, don’t know how much milk costs
  - Problem: cannot specify complete guaranteed plan without specifying amount of money for Pay action
  - Solution: find out the price when we get to Busch’s

Conditional Planning

- Identify conditions that will be observable at execution time
- Include construct in plan that selects course of action based on condition:
  ```
  if (OnSale(Milk))
    Pay($0.99)
  else
    Pay($1.49)
  ```

- What if we don’t know (yet) whether the milk is on sale?
  - The conditional is undefined!
  - Is there an action that would resolve the uncertainty?
Information-Gathering Action

CheckOnSale (x, y)
Precond: InStore(x)
Effect: KnowsWhether(“OnSale(y)”)  

☐ KnowsWhether: eligible for conditioning
☐ A legal (conditional) plan:

Move(John, Store)
CheckOnSale(John, Milk)

If ( OnSale(Milk) )
   Pay(John, Store, $0.99)
else
   Pay(John, Store, $1.49)

Continuous replanning

☐ Alternative:
  ☐ Construct plan for expected case
   □ “There are no obstacles”...
   □ “The milk is on sale”...

☐ Update plan as information is revised
   □ There’s a log in the road!
   □ The milk isn’t on sale

☐ Creates real-time issues
Online replanning

Very large search spaces

- How many actions for path planning with a car?
  - Infinite number of possible actions!
  - Finding “perfect” solutions (e.g., a perfect parallel parking maneuver) is:
    - Very difficult
    - Generally unnecessary

- How do we efficiently search the space?
  - Make good use of our computational resources to find the best possible plan
### Deterministic Motion Planning

- We can sometimes consider only a few discrete (say 5) actions

- Shortcomings
  - Algorithm is no longer complete (or optimal).
  - If we had more CPU time, could we compute a better plan?

### Non-Deterministic Planning

- How do we explore a large search tree, finding a good enough answer while making use of what CPU time we have available?

- Consider sequences of random actions
  - Build a tree of actions
    - Node = state
    - Edge = action
  - A plan is a sequence of actions, i.e., a path from the root (initial state) to a leaf near the goal

- Skeleton algorithm:
  - while time remains
    - Select a node (call it parent) in the tree
    - Generate a new action a
    - Create new node: child = propagate(parent, a)
    - If action is safe
      - Add node to tree
    - return best plan found so far

Implementation here is critical…

Analogous to fringe.get()
Non-Deterministic Planning Variants

- We’ll consider three basic variants
  - **Random**: Pick a parent node at random. Pick an action at random.
  - **RRT**: Pick a destination at random. Find parent node that is closest to destination. Compute action that goes towards destination
  - **RRT-Biased**: Pick the destination randomly, but in a biased way (i.e., prefer directions that are heuristically likely to work out)

Random Policy

- **Random**: Pick a parent node at random. Pick an action at random.
Random Policy: Analysis

- Complete?
- Optimal?
- Practical?

RRT Policy

- **RRT**: Pick a destination at random. Find parent node that is closest to destination. Compute action that goes towards destination.
RRT Policy (Biased Sampling)

- **RRT-Biased**: Pick the destination randomly, but in a biased way (i.e., prefer directions that are heuristically likely to work out)

RRT Discussion

- Results in trees with specific structure:
  - In 2D path planning, results in *planar* trees
  - Paths don’t overlap each other

  - This means, even in limit of $t \to \infty$, that not all paths will be computed any more
    - No longer complete!

- However, very useful in practice
  - Explores search space very rapidly
  - Good diversity of solutions
  - Random restarts “fixes” completeness problem
Continuous Replanning

Continuous Replanning
Multi-Agent Planning

- **Adversarial**
  - Mini-max style searches (we saw this before)

- **Cooperative**
  - Just a bigger search space
  - Suppose depth-limited search of depth $d$, with one robot that can move N, S, E, or W. Complexity?
  - What if we have N robots?

- **Coexist**
  - Model other agents’ actions as uncertain

Pursuit/Evasion
Multi-agent exploration

Next time: Probability

- Suppose the $P(\text{heads})$ of a coin is $p$.
- What is the probability of $N$ heads in a row?
- What’s the probability of at least 1 tails in $N$ trials?
- Suppose a coin comes up heads, heads, heads. What is the probability that the fourth toss is also heads?
- Suppose $x$ and $y$ are two random variables. Define conditional, marginal, joint probability.
Review Questions