GraphPlan

Dana S. Nau

CMSC 421/620, Spring 2005
GraphPlan

- Plan-space planning was state-of-the-art until about 10 years ago.

- In 1995, someone discovered an algorithm that’s usually much faster.
  - The Graphplan algorithm.
Graphplan

procedure Graphplan:

- for $k = 0, 1, 2, \ldots$

  - **Graph expansion:**
    - create a “planning graph” that contains $k$ “levels”
    - Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence

- If it does, then
  - do solution extraction:
    - backward search, modified to consider only the actions in the planning graph
    - if we find a solution, then return it
The Planning Graph

- Alternating layers of ground literals and actions
  - All actions that might possibly occur at each time step
  - All of the literals asserted by those actions

Maintenance actions: propagate literals to the next level. These represent what happens if no action in the final plan affects the literal.
Example

due to Dan Weld (U. of Washington)

Suppose you want to prepare dinner as a surprise for your sweetheart (who is asleep)

\[ s_0 = \{ \text{garbage, cleanHands, quiet} \} \]
\[ g = \{ \text{dinner, present, } \neg \text{garbage} \} \]

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>cook()</td>
<td>cleanHands</td>
<td>dinner</td>
</tr>
<tr>
<td>wrap()</td>
<td>quiet</td>
<td>present</td>
</tr>
<tr>
<td>carry()</td>
<td>none</td>
<td>( \neg )garbage, ( \neg )cleanHands</td>
</tr>
<tr>
<td>dolly()</td>
<td>none</td>
<td>( \neg )garbage, ( \neg )quiet</td>
</tr>
</tbody>
</table>

Also have the maintenance actions: one for each literal
Example (continued)

- state-level 0: 
  \{all atoms in \(s_0\)\} \(\cup\) 
  \{negations of all atoms not in \(s_0\)\}

- action-level 1: 
  \{all actions whose preconditions are satisfied in \(s_0\)\}

- state-level 1: 
  \{all effects of all of the actions in action-level 1\}

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>cook()</td>
<td>cleanHands</td>
<td>dinner</td>
</tr>
<tr>
<td>wrap()</td>
<td>quiet</td>
<td>present</td>
</tr>
<tr>
<td>carry()</td>
<td>none</td>
<td>(\neg)garbage, (\neg)cleanHands</td>
</tr>
<tr>
<td>dolly()</td>
<td>none</td>
<td>(\neg)garbage, (\neg)quiet</td>
</tr>
</tbody>
</table>

Also have the maintenance actions:

- \(\neg\)dinner
- \(\neg\)present
Mutual Exclusion

- Two actions at the same action-level are mutex if
  - *Inconsistent effects*: an effect of one negates an effect of the other
  - *Interference*: one deletes a precondition of the other
  - *Competing needs*: they have mutually exclusive preconditions
- Otherwise they don’t interfere with each other
  - Both may appear in a solution plan
- Two literals at the same state-level are mutex if
  - *Inconsistent support*: one is the negation of the other, or all ways of achieving them are pairwise mutex
Example (continued)

- Augment the graph to indicate mutexes
- *carry* is mutex with the maintenance action for *garbage* (inconsistent effects)
- *dolly* is mutex with *wrap*
  - interference
- ~*quiet* is mutex with *present*
  - inconsistent support
- each of *cook* and *wrap* is mutex with a maintenance operation

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>cook()</td>
<td>cleanHands</td>
<td>dinner</td>
</tr>
<tr>
<td>wrap()</td>
<td>quiet</td>
<td>present</td>
</tr>
<tr>
<td>carry()</td>
<td><em>none</em></td>
<td>~garbage, ~cleanHands</td>
</tr>
<tr>
<td>dolly()</td>
<td><em>none</em></td>
<td>~garbage, ~quiet</td>
</tr>
</tbody>
</table>

Also have the maintenance actions
Example (continued)

- Check to see whether there’s a possible plan
- Recall that the goal is
  - \{ \neg \text{garbage, dinner, present} \}
- Note that
  - All are possible in \( s_1 \)
  - None are mutex with each other
- Thus, there’s a chance that a plan exists
- Try to find it
  - Solution extraction
procedure Solution-extraction(g, j)
    if j = 0 then return the solution
    for each literal l in g
        nondeterministically choose an action
        to use in state s_{j-1} to achieve l
    if any pair of chosen actions are mutex
        then backtrack
    g' := {the preconditions of
            the chosen actions}
    Solution-extraction(g', j-1)
end Solution-extraction
Example (continued)

- Two sets of actions for the goals at state-level 1
- Neither works: both sets contain actions that are mutex
Recall what the algorithm does

procedure Graphplan:

- for $k = 0, 1, 2, \ldots$
  - **Graph expansion:**
    » create a “planning graph” that contains $k$ “levels”
  - Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence
  - If it does, then
    » do solution extraction:
      • backward search, modified to consider only the actions in the planning graph
      • if we find a solution, then return it
Example (continued)

- Go back and do more graph expansion
- Generate another action-level and another state-level
Example (continued)

- Solution extraction
- Twelve combinations at level 4
  - Three ways to achieve $\neg \text{garb}$
  - Two ways to achieve dinner
  - Two ways to achieve present

<table>
<thead>
<tr>
<th>state-level 0</th>
<th>action-level 1</th>
<th>state-level 1</th>
<th>action-level 2</th>
<th>state-level 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{garb}$</td>
<td>carry</td>
<td>$\neg \text{garb}$</td>
<td>carry</td>
<td>$\neg \text{garb}$</td>
</tr>
<tr>
<td>cleanH</td>
<td>$\neg \text{garb}$</td>
<td>cleanH</td>
<td>$\neg \text{cleanH}$</td>
<td>cleanH</td>
</tr>
<tr>
<td>quiet</td>
<td>$\neg \text{cleanH}$</td>
<td>quiet</td>
<td>$\neg \text{quiet}$</td>
<td>quiet</td>
</tr>
<tr>
<td>wrap</td>
<td>$\neg \text{quiet}$</td>
<td>wrap</td>
<td>$\neg \text{dinner}$</td>
<td>dinner</td>
</tr>
<tr>
<td>present</td>
<td>$\neg \text{dinner}$</td>
<td>present</td>
<td>$\neg \text{present}$</td>
<td>$\neg \text{present}$</td>
</tr>
</tbody>
</table>
Several of the combinations look OK at level 2

Here’s one of them
- Call Solution-Extraction recursively at level 2
- It succeeds
- Solution whose parallel length is 2
Comparison with PSP

- Unlike PSP, Graphplan creates ground instances of everything
  - Many of them may be irrelevant
- But the backward-search part of Graphplan—which is the hard part—will only look at the actions in the planning graph
  - smaller search space than PSP; thus faster