Plan-Space Planning
Motivation

- Problem with state-space search
  - In some cases we may try many different orderings of the same actions before realizing there is no solution

  dead end — ... — a — b — c
  dead end — ... — b — a — c
  dead end — ... — b — a — c
  dead end — ... — a — c — b — a
  dead end — ... — c — b — a — c

- Least-commitment strategy: don’t commit to orderings, instantiations, etc., until necessary
Outline

- Basic idea
- Open goals
- Threats
- The PSP algorithm
- Long example
- Comments
Plan-Space Planning - Basic Idea

- Backward search from the goal
- Each node of the search space is a *partial plan*
  - A set of partially-instantiated actions
  - A set of constraints
- Make more and more refinements, until we have a solution
- Types of constraints:
  - *precedence constraint*: $a$ must precede $b$
  - *binding constraints*:
    - inequality constraints, e.g., $v_1 \neq v_2$ or $v \neq c$
    - equality constraints (e.g., $v_1 = v_2$ or $v = c$) or substitutions
  - *causal link*:
    - use action $a$ to establish the precondition $p$ needed by action $b$
- How to tell we have a solution: no more *flaws* in the plan
  - Will discuss flaws and how to resolve them
An Example of Partial Plan

Dana Nau: Lecture slides for Automated Planning
Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/
Flaws: 1. Open Goals

- Open goal:
  - An action $a$ has a precondition $p$ that we haven’t decided how to establish

- Resolving the flaw:
  - Find an action $b$
    - (either already in the plan, or insert it)
  - that can be used to establish $p$
    - can precede $a$ and produce $p$
  - Instantiate variables
  - Create a causal link
Flaws: 2. Threats

● Threat: a deleted-condition interaction
  ◆ Action $a$ establishes a precondition (e.g., $pq(x)$) of action $b$
  ◆ Another action $c$ is capable of deleting $p$

● Resolving the flaw:
  ◆ impose a constraint to prevent $c$ from deleting $p$

● Three possibilities:
  ◆ Make $b$ precede $c$
  ◆ Make $c$ precede $a$
  ◆ Constrain variable(s) to prevent $c$ from deleting $p$
The PSP Procedure

\[
PSP(\pi)
\]
\[
flaws \leftarrow \text{OpenGoals}(\pi) \cup \text{Threats}(\pi)
\]
\[
\text{if } flaws = \emptyset \text{ then return}(\pi)
\]
\[
\text{select any flaw } \phi \in flaws
\]
\[
\text{resolvers} \leftarrow \text{Resolve}(\phi, \pi)
\]
\[
\text{if resolvers} = \emptyset \text{ then return}(\text{failure})
\]
\[
\text{nondeterministically choose a resolver } \rho \in \text{resolvers}
\]
\[
\pi' \leftarrow \text{Refine}(\rho, \pi)
\]
\[
\text{return}(PSP(\pi'))
\]

- PSP is both sound and complete
Example

- Similar (but not identical) to an example in Russell and Norvig’s *Artificial Intelligence: A Modern Approach* (1st edition)

- Operators:
  - **Start**
    - Precond: none
    - Effects: At(Home), sells(HWS,Drill), Sells(SM,Milk), Sells(SM,Banana)
  - **Finish**
    - Precond: Have(Drill), Have(Milk), Have(Banana), At(Home)
  - **Go(l,m)**
    - Precond: At(l)
    - Effects: At(m), ¬At(l)
  - **Buy(p,s)**
    - Precond: At(s), Sells(s,p)
    - Effects: Have(p)
Example (continued)

- Initial plan

Start

At(Home), Sells(HWS,Drill), Sells(SM,Milk), Sells(SM,Bananas)

Have(Drill) Have(Milk) Have(Bananas) At(Home)

Finish
Example (continued)

- The only possible ways to establish the Have preconditions
Example (continued)

- The only possible ways to establish the Sells preconditions
The only ways to establish At(HWS) and At(SM)

- Note the threats
Example (continued)

- To resolve the threat to $\text{At}(s_1)$, make $\text{Buy}(\text{Drill})$ precede $\text{Go}(\text{SM})$
  - This resolves all three threats
Example (continued)

- Establish $At(l_1)$ with $l_1$=Home
Example (continued)

- Establish $\text{At}(l_2)$ with $l_2 = \text{HWS}$
Example (continued)

- Establish At(Home) for Finish
  - This creates a bunch of threats
Example (continued)

- Constrain Go($l_3$, Home) to remove threats to At(SM)
  - This also removes the other threats
Final Plan

- Establish $\text{At}(l_3)$ with $l_3 = \text{SM}$
PSP doesn’t commit to orderings and instantiations until necessary

- Avoids generating search trees like this one:

Problem: how to prune infinitely long paths?

- Loop detection is based on recognizing states we’ve seen before
- In a partially ordered plan, we don’t know the states

Can we prune if we see the same action more than once?

... — go(b,a) — go(a,b) — go(b,a) — at(a)

No. Sometimes we might need the same action several times in different states of the world (see next slide)
Example

- 3-digit binary counter starts at 000, want to get to 110
  \[ s_0 = \{d_3=0, d_2=0, d_1=0\} \]
  \[ g = \{d_3=1, d_2=1, d_1=0\} \]

- Operators to increment the counter by 1:
  
  incr0
  
  Precond: \(d_1=0\)
  
  Effects: \(d_1=1\)

  incr01
  
  Precond: \(d_2=0, d_1=1\)
  
  Effects: \(d_2=1, d_1=0\)

  incr011
  
  Precond: \(d_3=0, d_2=1, d_1=1\)
  
  Effects: \(d_3=1, d_2=0, d_1=0\)
A Weak Pruning Technique

- Can prune all paths of length $> n$, where $n = |\{\text{all possible states}\}|$
  - This doesn’t help very much

- I’m not sure whether there’s a good pruning technique for plan-space planning