Chapter 5
Plan-Space Planning

Dana S. Nau
CMSC 722, AI Planning
University of Maryland, Fall 2004
Plan Descriptions

- There are 3 types of choices made when planning
  - What action to put into the plan
  - Where to put it in the plan
  - How to instantiate the action’s parameters

- With *lifting* we saw one way to defer one of these choice types

- What about the other types, how can we defer them?
Motivation

- Why would we want to defer them?

- 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  b
dead end — ... — a — c  b
dead end — ... — b — c  a
dead end — ... — c — b  a
```
Motivation

- This situation is similar to the one we saw with regards to instantiating parameters when doing regression planning.

- When doing regression, we are checking that the action is *relevant*.

- However, usually only a few of the action’s parameters need to be instantiated to insure that the action is relevant.

- If we leave those other parameters uninstantiated, we avoid prematurely committing to a particular instantiation.
Motivation

- In planning, we may know that an action must occur before a specific other action, but not necessarily where it should be placed with respect to all of the other actions in the plan.

- Defer constraining order of actions until there is a reason to do so.

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

- How can we do this?
Outline

- Basic idea
- Open goals
- Threats
- The PSP algorithm
- Long example
- Comments
- Review Questions
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
Type of Flaws in Plan

- Open Goal
- Threat
Open Goal

● Flaw:
  ◆ 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
Threat

- **Flaw**: a deleted-condition interaction
  - Action $a$ establishes a condition (e.g., $p(x)$) for action $b$
  - Another action $c$ is capable of deleting this condition $p(x)$
- **Resolving the flaw**:
  - impose a constraint to prevent $c$ from deleting $p(x)$
- **Three possibilities**:
  - Make $b$ precede $c$
  - Make $c$ precede $a$
  - Constrain variable(s) to prevent $c$ from deleting $p(x)$

**Diagram Example**:

- $a(x)$
  - Precond: …
  - Effects: $p(x)$
- $b(x)$
  - Precond: $p(x)$
  - Effects: …
- $c(y)$
  - Precond: …
  - Effects: $\neg p(y)$
The PSP Procedure

- Selection of flaw is non-backtrackable!!
- PSP is both sound and complete

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

- Nodes are partial plans
  - i.e., plans with flaws

- Edges are plan refinement operators
  - operators remove plan flaws
    - add action to remove open goal flaw
    - add constraints to remove open goal or threat flaws
Plan Space Search Space

- Node (Partial Plan) Structure
  - flaws
    - open goals
    - threats
  - actions
  - constraints
    - order
    - binding
    - causal links
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 (root node)

```
Start

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

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

Finish
```
The only possible ways to establish the “Have” preconditions

- At($s_1$), Sells($s_1$, Drill)
- At($s_2$), Sells($s_2$, Milk)
- At($s_3$), Sells($s_3$, Bananas)

Buy(Drill, $s_1$) → Have(Drill), Have(Milk), Have(Bananas), At(Home)
Buy(Milk, $s_2$) → Have(Drill), Have(Milk), Have(Bananas), At(Home)
Buy(Bananas, $s_2$) → Have(Drill), Have(Milk), Have(Bananas), At(Home)
Example (continued)

- The only possible way to establish the “Sells” preconditions
Example (continued)

- The only ways to establish \text{At}(\text{HWS}) \text{ and } \text{At}(\text{SM})
  - Note the threats
Example (continued)

- To resolve the third threat, make Buy(Drill) precede Go(SM)
  - This resolves all three threats
Example (continued)

- Establish $\text{At}(l_1)$ with $l_1=\text{Home}$
Example (continued)

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

- Establish At(Home) for Finish
Example (continued)

- Constrain Go(Home) to remove threats to At(SM)
Final Plan

- Establish $\text{At}(l_3)$ with $l_3=\text{SM}$

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/
Comments

- 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 with $n$ actions, where $n = |\{\text{all possible states}\}|$
  ♦ This doesn’t help very much

● Not sure whether there’s a good pruning technique for plan-space planning

● This means that plan space planners will continue searching for a solution long after a state space planner has found it to be unsolvable
Plan Space Planning Shortcomings

- Past couple of decades has seen significant progress in developing techniques to automatically generate strong state space heuristics.

- We do not yet know how to generate good plan space heuristics.

- Can one adapt current state space heuristic generation methods to plan space search?

- That would be a great research topic for someone thinking of doing an honours, etc. :^)
Plan Space Planning Strengths

- Can have much smaller search space than for state space planners.

- Plans can be generated so that duplicate plans can never occur, thus not needing to store a closed list to check for dupes.

- Since each node is a plan structure, when a goal node is found, the problem’s solution path can be extracted directly from the node.
Why did Plan Space Planning Arise?

- What is the problem with state space planning that plan space planning is trying to solve?
- How is it trying to overcome this problem?
- Was it successful, why or why not?
Summary Questions

- Plan space planning tries to defer planning decisions until they’re necessary, what makes them necessary?

- What two things enable plan space planners to defer planning decisions?
Summary Questions

- Looking at the individual types of plan modifications, how would you characterize the main difference between the granularity of the decisions made by a forward state space planner and that of those made by a plan space planner?

- Does this remind you of any other type of task?
Summary Questions

- What is the structure of a plan space node?
- What do the edges in the plan space search do?
- What signals that the plan solves the problem?
Summary Questions

- What is the goal description for a plan space search problem?

- Can plan space planners work backwards from that goal description?

- If so, how, if not, why not?
Summary Questions

● What types of search strategy do not make sense for plan space planners and why?

● What types of search strategies make sense for plan space planners?