Contents

Lifted Planning Algorithms
- Lifted backwards search
- The STRIPS algorithm
- Plan-Space-Planning algorithms
Today: Consider the lifted propositional representation

Types: location, package
Objects: P1, P2 - package, A, B, C - location
Predicates: at(P - package, L - location),
truckAt(L - location), inTruck(P - package), road(...)
I = truckAt(A), at(P1, A), at(P2, B), road(A, B), road(B, C)
G = at(P1, C), at(P2, C)
Operators:

- **move(l1 - location, l2 - location):**
  \[
  \{
  \text{road}(l1,l2), \text{truckAt}(l1)
  \},
  \{
  \neg \text{truckAt}(l1), \text{truckAt}(l2)
  \}
  \]

- **load(p - package, l - location):**
  \[
  \{
  \text{at}(p,l), \text{truckAt}(l)
  \},
  \{
  \neg \text{at}(p,l), \text{inTruck}(p)
  \}
  \]

- **unload(p - package, l - location):**
  \[
  \{
  \text{inTruck}(p), \text{truckAt}(l)
  \},
  \{
  \text{at}(p,l), \neg \text{inTruck}(p)
  \}
  \]
Lifted Backwards Search

- Goal: decrease the branching factor of search
- Like backwards search from lecture 3 but some of the operators partially grounded
  - some variables remain unassigned
  - move(?l1, C) – move to C from somewhere
  - unload(?p, B) – unload some package at B
- We will try to ground as little as possible – least commitment strategy
Lifted Backwards Search

\[
\text{Lifted-backward-search}(O, s_0, g)
\]
\[
\pi \leftarrow \text{the empty plan}
\]
\[
\text{loop}
\]
\[
\text{if } s_0 \text{ satisfies } g \text{ then return } \pi
\]
\[
A \leftarrow \{(o, \theta)| o \text{ is a standardization of an operator in } O, \theta \text{ is an mgu for an atom of } g \text{ and an atom of effects } (o), \text{and } \gamma^{-1}(\theta(g), \theta(o)) \text{ is defined}\}
\]
\[
\text{if } A = \emptyset \text{ then return failure}
\]
\[
\text{nondeterministically choose a pair } (o, \theta) \in A
\]
\[
\pi \leftarrow \text{the concatenation of } \theta(o) \text{ and } \theta(\pi)
\]
\[
g \leftarrow \gamma^{-1}(\theta(g), \theta(o))
\]

- standardization = copy with fresh variables
- mgu = most general unifier (keep as many variables unassigned as possible)
- branching factor will get (significantly) decreased, still need backtracking though
The STRIPS Algorithm

- STRIPS = Stanford Research Institute Problem Solver
- Introduced the “PDDL” formalism, Fikes and Nilsson 1971

Shakey the first general-purpose mobile robot

- STRIPS is a special variant of lifted backwards search
- Goal: decrease branching factor even further
The original STRIPS algorithm is a lifted version of the algorithm below.

\[
\text{Ground-STRIPS}(O, s, g) \\
\pi \leftarrow \text{the empty plan} \\
\text{loop} \\
\quad \text{if } s \text{ satisfies } g \text{ then return } \pi \\
\quad A \leftarrow \{a \mid a \text{ is a ground instance of an operator in } O, \\
\quad \quad \quad \text{and } a \text{ is relevant for } g\} \\
\quad \text{if } A = \emptyset \text{ then return failure} \\
\quad \text{nondeterministically choose any action } a \in A \\
\quad \pi' \leftarrow \text{Ground-STRIPS}(O, s, \text{precond}(a)) \\
\quad \text{if } \pi' = \text{failure} \text{ then return failure} \\
\quad ;; \text{if we get here, then } \pi' \text{ achieves } \text{precond}(a) \text{ from } s \\
\quad s \leftarrow \gamma(s, \pi') \\
\quad ;; s \text{ now satisfies } \text{precond}(a) \\
\quad s \leftarrow \gamma(s, a) \\
\quad \pi \leftarrow \pi \cdot \pi'. a
\]

\[
g_2 = (g - \text{effects}(a_2)) \cup \text{precond}(a_2) \\
\pi' = \langle a_6, a_4 \rangle \text{ is a plan for } \text{precond}(a_2) \\
s = \gamma(\gamma(s_0, a_6), a_4) \text{ is a state satisfying } \text{precond}(a_2)
\]
The STRIPS Algorithm Properties

- STRIPS tries to solve each goal separately
- The current goal is the preconditions of the last selected action
- It works if the goals can be solved in some linear order

- Is the STRIPS algorithm sound? (always correct answers)
- Is the STRIPS algorithm complete? (always finds a solution if one exists)
- Can STRIPS find an optimal plan?
The STRIPS Algorithm Properties

- STRIPS tries to solve each goal separately
- The current goal is the preconditions of the last selected action
- It works if the goals can be solved in some linear order
- Is the STRIPS algorithm sound? (always correct answers)
- Is the STRIPS algorithm complete? (always finds a solution if one exists)
- Can STRIPS find an optimal plan?
The Sussman anomaly

Consider the following Blocksworld problem:

Optimal plan: unstack(C,A), put(C), pickup(B), stack(B,C), pickup(A), stack(A,B)

It is impossible for STRIPS to find an optimal plan
  - No matter what we choices we make
Let’s try

Predicates:
- onTable(x), on(x,y), clear(x), holding(x), handEmpty
- Initial state: on(C,A), onTable(A), onTable(B), clear(C), clear(B), handEmpty
- Goal: on(A,B), on(B,C)
Let’s try

**Classical Operators**

unstack\((x,y)\)
- **Precond:** on\((x,y)\), clear\((x)\), handempty
- **Effects:** \(\neg\)on\((x,y)\), \(\neg\)clear\((x)\), \(\neg\)handempty, holding\((x)\), clear\((y)\)

stack\((x,y)\)
- **Precond:** holding\((x)\), clear\((y)\)
- **Effects:** \(\neg\)holding\((x)\), \(\neg\)clear\((y)\), on\((x,y)\), clear\((x)\), handempty

pickup\((x)\)
- **Precond:** ontable\((x)\), clear\((x)\), handempty
- **Effects:** \(\neg\)ontable\((x)\), \(\neg\)clear\((x)\), \(\neg\)handempty, holding\((x)\)

putdown\((x)\)
- **Precond:** holding\((x)\)
- **Effects:** \(\neg\)holding\((x)\), ontable\((x)\), clear\((x)\), handempty
STRIPS is incomplete

STRIPS cannot solve the following problem:

- The register assignment problem
  - swap the values of two variables (registers)
  - registers: R1, R2, R3 values: V0, V1, ..., V5
  - initial state: value(R1, V3), value(R2, V5), value(R3, V0)
  - goal: value(R1, V5), value(R2, V3)
  - operator: assign(r1, v1, r2, v2):
    value(r1, v1), value(r2, v2) → value(r1, v2)

- Proof: see homework assignment nr. 3
Plan Space Planning

To introduce PSP we use the excellent slides of prof. Dana S. Nau

https://www.cs.umd.edu/~nau/planning/slides/chapter05.pdf
Chapter 5
Plan-Space Planning

Dana S. Nau
University of Maryland

5:10 PM    February 6, 2012
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

  ![Diagram showing state-space search]

- 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 \(v = c\)) and/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

\[\begin{align*}
\text{foo}(x) &; \quad \text{Precond: \ldots} \\
&; \quad \text{Effects: } p(x) \\
\end{align*}\]

\[\begin{align*}
\text{bar}(y) &; \quad \text{Precond: } \neg p(y) \\
&; \quad \text{Effects: \ldots} \\
\end{align*}\]

\[\begin{align*}
\text{baz}(z) &; \quad \text{Precond: } p(z) \\
&; \quad \text{Effects: \ldots} \\
\end{align*}\]
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 and/or constrain variable bindings
  - 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 is both sound and complete
- It returns a partially ordered solution plan
  - Any total ordering of this plan will achieve the goals
  - Or could execute actions in parallel if the environment permits it

```
PSP(π)
flaws ← OpenGoals(π) ∪ Threats(π)
if flaws = ∅ then return(π)
select any flaw φ ∈ flaws
resolvers ← Resolve(φ, π)
if resolvers = ∅ then return(failure)
nondeterministically choose a resolver ρ ∈ resolvers
π' ← Refine(ρ, π)
return(PSP(π'))
end
```
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)

*Start* and *Finish* are dummy actions that we’ll use instead of the initial state and goal.
Example (continued)

- Need to give PSP a plan $\pi$ as its argument
  - Initial plan: **Start**, **Finish**, and an ordering constraint

**Effects:** $\text{At(Home)}$, $\text{Sells(HWS,Drill)}$, $\text{Sells(SM,Milk)}$, $\text{Sells(SM,Bananas)}$

**Precond:** $\text{Have(Drill)}$, $\text{Have(Milk)}$, $\text{Have(Bananas)}$, $\text{At(Home)}$
Example (continued)

- The first three refinement steps
  - These are the only possible ways to establish the Have preconditions

Why don’t we use Start to establish At(Home)?
Example (continued)

- Three more refinement steps
  - The only possible ways to establish the Sells preconditions

- At(HWS)
- Sells(HWS, Drill)
- At(SM)
- Sells(SM, Milk)
- At(SM)
- Sells(SM, Bananas)
- Buy(Drill, HWS)
- Buy(Milk, SM)
- Buy(Bananas, SM)
- Have(Drill)
- Have(Milk)
- Have(Bananas)
- At(Home)

Finish
Example (continued)

- Two more refinements: the only ways to establish $\text{At(HWS)}$ and $\text{At(SM)}$
  - This time, several threats occur

![Diagram showing a plan with actions and locations]

- $\text{At}(l_1)$
- $\text{Go}(l_1, \text{HWS})$
- $\text{At}(\text{HWS})$
- $\text{Sells}($HWS,Drill$)$
- $\text{Buy}($Drill, HWS$)$
- $\text{Have}($Drill$)$
- $\text{Have}($Milk$)$
- $\text{At}(l_2)$
- $\text{Go}(l_2, \text{SM})$
- $\text{Sells}($SM,Milk$)$
- $\text{Buy}($Milk, SM$)$
- $\text{Have}($Milk$)$
- $\text{Have}($Bananas$)$
- $\text{Sells}($SM,Bananas$)$
- $\text{Buy}($Bananas, SM$)$
- $\text{Have}($Bananas$)$
- $\text{At}(\text{Home})$
- $\text{At}(l_2)$
- $\text{Finish}$
Example (continued)

- Nondeterministic choice: how to resolve the threat to \( \text{At}(s_1) \)?
  - Our choice: make \( \text{Buy(Drill)} \) precede \( \text{Go}(l_2, \text{SM}) \)
  - This also resolves the other two threats (why?)
Example (continued)

- Nondeterministic choice: how to establish $At(l_1)$?
  - We’ll do it from Start, with $l_1=\text{Home}$
  - How else could we have done it?
Example (continued)

- Nondeterministic choice: how to establish At($l_2$)?
  - We’ll do it from Go(Home,HWS), with $l_2 = \text{HWS}$
Example (continued)

- The only feasible way to establish At(Home) for Finish
  - This creates a bunch of threats
Example (continued)

- To remove the threats to $\text{At}(\text{SM})$ and $\text{At}(\text{HWS})$, make them precede $\text{Go}(l_3,\text{Home})$
  - This also removes the other threats
Final Plan

- Establish $At(l_3)$ with $l_3=SM$
- We’re done!

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/
Discussion

- How to choose which flaw to resolve first and how to resolve it?
  - We’ll return to these questions in Chapter 10
- 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?
  - No. Sometimes we might need the same action several times in different states of the world

\[ \text{Example on next slide} \]
Example

- 3-digit binary counter starts at 000, want to get to 111
  \[ s_0 = \{d_3=0, d_2=0, d_1=0\}, \quad \text{i.e.,} \quad 000 \]
  \[ g = \{d_3=1, d_2=1, d_1=1\}, \quad \text{i.e.,} \quad 111 \]

- Operators to increment the counter by 1:
  - incr-xx0-to-xx1
    - Precond: \( d_1 = 0 \)
    - Effects: \( d_1 = 1 \)
  - incr-x01-to-x10
    - Precond: \( d_2 = 0, d_1 = 1 \)
    - Effects: \( d_2 = 1, d_1 = 0 \)
  - incr-011-to-100
    - Precond: \( d_3 = 0, d_2 = 1, d_1 = 1 \)
    - Effects: \( d_3 = 1, d_2 = 0, d_1 = 0 \)

Plan:

Initial state: \[ \begin{array}{ccc} d_3 & d_2 & d_1 \\ 0 & 0 & 0 \end{array} \]

- incr-xx0-to-xx1 \rightarrow \[ \begin{array}{ccc} 0 & 0 & 1 \end{array} \]
- incr-x01-to-x10 \rightarrow \[ \begin{array}{ccc} 0 & 1 & 0 \end{array} \]
- incr-xx0-to-xx1 \rightarrow \[ \begin{array}{ccc} 0 & 1 & 1 \end{array} \]
- incr-011-to-100 \rightarrow \[ \begin{array}{ccc} 1 & 0 & 0 \end{array} \]
- incr-xx0-to-xx1 \rightarrow \[ \begin{array}{ccc} 1 & 0 & 1 \end{array} \]
- incr-x01-to-x10 \rightarrow \[ \begin{array}{ccc} 1 & 1 & 0 \end{array} \]
- incr-xx0-to-xx1 \rightarrow \[ \begin{array}{ccc} 1 & 1 & 1 \end{array} \]
A Weak Pruning Technique

- Can prune all partial plans of \( n \) or more actions, 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
The End

Next week: GraphPlan and SATPlan