Classical STRIPS Planning

Alan Fern *

* Based in part on slides by Daniel Weld.
Stochastic/Probabilistic Planning: Markov Decision Process (MDP) Model

Percepts

World

Actions

Goal

maximize expected reward over lifetime

perfect

fully observable

sole source of change

stochastic

instantaneous
Classical Planning Assumptions

Percepts

- perfect
- fully observable

World

Actions

- sole source of change
- deterministic
- instantaneous

Goal

achieve goal condition
Why care about classical planning?

• Places an emphasis on analyzing the combinatorial structure of problems
  ▲ Developed a many powerful ideas in this direction
  ▲ MDP research has *mostly* ignored this type of analysis

• Classical planners tend scale much better to large state spaces by leveraging those ideas

• **Replanning:** many stabilized environments ~satisfy classical assumptions (e.g. robotic crate mover)
  ▲ It is possible to handle minor assumption violations through replanning and execution monitoring
  ▲ The world is often not so random and can be effectively thought about deterministically
Why care about classical planning?

- Ideas from classical planning techniques often form the basis for developing non-classical planning techniques
  - Recent work uses classical planners as a component of probabilistic planning [Yoon et. al. 2008] (i.e. reducing probabilistic planning to classical planning)
  - Powerful domain analysis techniques from classical planning have been integrated into MDP planners
Representing States

**World states** are represented as sets of facts.

We will also refer to facts as propositions.

Closed World Assumption (CWA):
Fact not listed in a state are assumed to be false. Under CWA we are assuming the agent has full observability.
Representing Goals

**Goals** are also represented as sets of facts.

For example $\{ \text{on}(A,B) \}$ is a goal in the blocks world.

A **goal state** is any state that contains all the goal facts.

State 1 is a goal state for the goal $\{ \text{on}(A,B) \}$.
State 2 is not a goal state for the goal $\{ \text{on}(A,B) \}$.
A STRIPS action definition specifies:
1) a set \( \text{PRE} \) of preconditions facts
2) a set \( \text{ADD} \) of add effect facts
3) a set \( \text{DEL} \) of delete effect facts

**PutDown**(A,B):

\[
\begin{align*}
\text{PRE}: & \{ \text{holding}(A), \text{clear}(B) \} \\
\text{ADD}: & \{ \text{on}(A,B), \text{handEmpty}, \text{clear}(A) \} \\
\text{DEL}: & \{ \text{holding}(A), \text{clear}(B) \}
\end{align*}
\]
A STRIPS action is **applicable** (or allowed) in a state when its preconditions are contained in the state.

Taking an action in a state $S$ results in a new state $S \cup ADD – DEL$ (i.e. add the add effects and remove the delete effects)

**PutDown(A,B):**

**PRE:** \{ holding(A), clear(B) \}

**ADD:** \{ on(A,B), handEmpty, clear(A) \}

**DEL:** \{ holding(A), clear(B) \}
A STRIPS planning problem specifies:
1) an initial state \( S \)
2) a goal \( G \)
3) a set of STRIPS actions

Objective: find a “short” action sequence reaching a goal state, or report that the goal is unachievable

Example Problem:

Initial State

<table>
<thead>
<tr>
<th>holding(A)</th>
<th>clear(B)</th>
<th>onTable(B)</th>
</tr>
</thead>
</table>

Goal

on(A,B)

Solution: \((\text{PutDown}(A,B))\)

PutDown(A,B):
- **PRE:** \{ holding(A), clear(B) \}
- **ADD:** \{ on(A,B), handEmpty, clear(A) \}
- **DEL:** \{ holding(A), clear(B) \}

PutDown(B,A):
- **PRE:** \{ holding(B), clear(A) \}
- **ADD:** \{ on(B,A), handEmpty, clear(B) \}
- **DEL:** \{ holding(B), clear(A) \}

STRIPS Actions
Propositional Planners

- For clarity we have written propositions such as on(A,B) in terms of objects (e.g. A and B) and predicates (e.g. on).

- However, the planners we will consider ignore the internal structure of propositions such as on(A,B).

- Such planners are called **propositional planners** as opposed to first-order or relational planners.

- Thus it will make no difference to the planner if we replace every occurrence of “on(A,B)” in a problem with “prop1” (and so on for other propositions).

- It feels wrong to ignore the existence of objects. But currently propositional planners are the state-of-the-art.
For convenience we typically specify problems via action schemas rather than writing out individual STRIPS actions.

**Action Schema: (x and y are variables)**

<table>
<thead>
<tr>
<th><strong>PutDown(x,y):</strong></th>
<th><strong>PutDown(B,A):</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PRE:</strong> { holding(x), clear(y) }</td>
<td><strong>PRE:</strong> { holding(B), clear(A) }</td>
</tr>
<tr>
<td><strong>ADD:</strong> { on(x,y), handEmpty, clear(x) }</td>
<td><strong>ADD:</strong> { on(B,A), handEmpty, clear(B) }</td>
</tr>
<tr>
<td><strong>DEL:</strong> { holding(x), clear(y) }</td>
<td><strong>DEL:</strong> { holding(B), clear(A) }</td>
</tr>
</tbody>
</table>

Each way of replacing variables with objects from the initial state and goal yields a “ground” STRIPS action.

Given a set of schemas, an initial state, and a goal, propositional planners compile schemas into ground actions and then ignore the existence of objects thereafter.
STRIPS Versus PDDL

• Your book refers to the PDDL language for defining planning problems rather than STRIPS

• The Planning Domain Description Language (PDDL) was defined by planning researchers as a standard language for defining planning problems
  ▲ Includes STRIPS as special case along with more advanced features
  ▲ Some simple additional features include: type specification for objects, negated preconditions, conditional add/del effects
  ▲ Some more advanced features include allowing numeric variables and durative actions

• Most planners you can download take PDDL as input
  ▲ Majority only support the simple PDDL features (essentially STRIPS)
  ▲ PDDL syntax is easy to learn from examples packaged with planners, but a definition of the STRIPS fragment can be found at:
Properties of Planners

• A planner is **sound** if any action sequence it returns is a true solution

• A planner is **complete** if it outputs an action sequence or “no solution” for any input problem

• A planner is **optimal** if it always returns the shortest possible solution

Is optimality an important requirement?
Is it a reasonable requirement?
Planning as Graph Search

• It is easy to view planning as a graph search problem
• Nodes/vertices = possible states
• Directed Arcs = STRIPS actions
• Solution: path from the initial state (i.e. vertex) to one state/vertices that satisfies the goal
Search Space: Blocks World

Graph is finite

Initial State

Goal State
Planning as Graph Search

• Planning is just finding a path in a graph
  ▶ Why not just use standard graph algorithms for finding paths?

• **Answer:** graphs are exponentially large in the problem encoding size (i.e. size of STRIPS problems).
  ▶ But, standard algorithms are poly-time in graph size
  ▶ So standard algorithms would require exponential time

• Can we do better than this?
PlanSAT

**Given:** a STRIPS planning problem
**Output:** “yes” if problem is solvable, otherwise “no”

- PlanSAT is decidable.
  - Why?

- In general PlanSAT is PSPACE-complete!
  Just finding a plan is hard in the worst case.
  - even when actions limited to just 2 preconditions and 2 effects

**Does this mean that we should give up on AI planning?**

NOTE: PSPACE is set of all problems that are decidable in polynomial space. PSPACE-complete is widely believed to strictly contain NP.
Satisficing vs. Optimality

• While just finding a plan is hard in the worst case, for many planning domains, finding a plan is easy.

• However finding optimal solutions can still be hard in those domains.
  † For example, optimal planning in the blocks world is NP-complete.

• In practice it is often sufficient to find “good” solutions “quickly” although they may not be optimal.
  † This is often referred to as the “satisficing” objective.
  † For example, producing approx. optimal blocks world solutions can be done in linear time. How?
Satisficing

• Still finding satisficing plans for arbitrary STRIPS problems is not easy.
  
  Must still deal with the exponential size of the underlying state spaces

• Why might we be able to do better than generic graph algorithms?

• **Answer:** we have the compact and structured STRIPS description of problems

  Try to leverage structure in these descriptions to intelligently search for solutions

• We will now consider several frameworks for doing this, in historical order.