Planning

Chapter 10, Sections 1–4
Outline

♦ Planning Domain Definition Language (PDDL)
♦ Forward and backward state-space search
♦ GraphPlan
♦ SatPlan
♦ Partial order planning
Automated Planning

Planning research has been central to AI from the beginning, partly because of practical interest but also because of the “intelligence” features of human planners.

◊ Large logistics problems, operational planning, robotics, scheduling etc.
◊ A number of international Conferences on Planning
◊ Bi-annual Planning competition
Automated Planning

The setting: a single agent in a fully observable, deterministic and static environment.

Propositional logic can express small domain planning problems, but becomes impractical if there are many actions and states (combinatorial explosion).

Example: In the wumpus world the action of a forward-step has to be written for all four directions, for all \( n^2 \) locations, and for each time step \( T \).

The Planning Domain Definition Language (PDDL) is a subset of FOL and more expressive than propositional logic. It allows for factored representation.
Planning Domain Definition Language (PDDL)

PDDL is derived from the STRIPS planning language.

– Initial and goal states.
– A set of ACTIONS($s$) in terms of preconditions and effects.
– Closed world assumption: Unmentioned state variables are assumed false.

Example:

ACTION: Fly($from$, $to$
PRECONDITION: At($p$, $from$), Plane($p$), Airport($from$), Airport($to$)
EFFECT: ¬At($p$, $from$), At($p$, $to$)
PDDL/STRIPS operators

Tidily arranged actions descriptions, restricted language

**ACTION:** Buy($x$)
**PRECONDITION:** At($p$), Sells($p$, $x$)
**EFFECT:** Have($x$)

[Note: this abstracts away many important details of buying!]

Restricted language $\Rightarrow$ efficient algorithm
  - Precondition: conjunction of positive literals
  - Effect: conjunction of literals

A complete set of STRIPS operators can be translated into a set of successor-state axioms
Example: Air cargo transport

A classical transportation problem: Loading and unloading cargo and flying between different airports.

Actions: Load(cargo, plane, airport), Unload(cargo, plane, airport),
          Fly(plane, airport, airport)

Predicates: In(cargo, plane), At(cargo ∨ plane, airport)

Example solution:
          Load(C1, P1, SFO), Fly(P1, SFO, JFK), Unload(C1, P1, JFK),
          Load(C2, P2, JFK), Fly(P2, JFK, SFO), Unload(C2, P2, SFO).
Example: The blocks world

Cube-shape blocks sitting on a table or stacked on top of each other.

Actions: PutOn(block, block), PutOnTable(block)

Predicates: On(block, block ∨ table), Clear(block ∨ table)
How difficult is planning?

Does there exist a plan that achieves the goal? **PlanSat**

Does there exist a solution of length at most $k$? **Bounded PlanSat**

PlanSat and Bounded PlanSat are PSPACE-complete.
  - i.e., difficult!

PlanSat without negative preconditions and without negative effects is in P.
  - i.e., solveable!
State-space search

◊ **Forward (progression):**
  state-space search considers actions that are **applicable**

◊ **Backward (regression):**
  state-space search considers actions that are **relevant**

Neither of them is efficient without good heuristics!
Heuristics for forward state-space search

For forward state-space search there are a number of domain-independent heuristics:

◇ Relaxing actions:
  – Ignore-preconditions heuristic
  – Ignore-delete-lists heuristic

◇ State abstractions:
  – Reduce the state space

Programs that has won the bi-annual Planning competition has often used
  – FF (fast forward) search with heuristics, or
  – planning graphs, or
  – SAT.
Planning graphs

The main disadvantage of state-space search is the size of the search tree (exponential). Also, the heuristics are not admissible in general.

The planning graph is a polynomial size approximation of the complete tree. Search on this graph is an admissible heuristic.

The planning graph is organized in alternating levels of possible states $S_i$ and applicable actions $A_i$. Links between levels represent preconditions and effects whereas links within the levels express conflicts (mutex-links).
Planning graphs

A planning problem with $l$ literals and $a$ actions has a polynomial size planning graph:

- Levels $S_i$ contain at most $l$ nodes and $l^2$ mutex links
- Levels $A_i$ contain at most $a + l$ nodes and $(a + l)^2$ mutex links
- At most $2(al + l)$ links between levels for preconditions and effects
- Therefore, a graph with $n$ levels has size $O(n(a + l)^2)$
The GraphPlan algorithm

The GraphPlan algorithm expands the graph with new levels $S_i$ and $A_i$ until there are no mutex links between the goals. To extract the actual plan, the algorithm searches backwards in the graph.

The plan extraction is the difficult part and is usually done with greedy-like heuristics.
SatPlan and CSP

Translate the PDDL description into a SAT problem or a CSP (constraint satisfaction problem).

The goal state as well as all actions have to be propositionalized. Action schemas have to be replaced by a set of ground actions, variables have to be replaced by constants, fluents need to be introduced for each time step, etc.

⇒ combinatorial explosion

In other words, we remove a part of the benefits of the expressiveness of PDDL to gain access to efficient solution methods for SAT and CSP solvers.
Historical remark: Linear planning

Planners in the early 1970s considered totally ordered action sequences
– problems were decomposed in subgoals
– the resulting subplans were stringed together in some order
– this is called linear planning

But, linear planning is incomplete!
– there are some very simple problems it cannot handle
  – e.g., the Sussman anomaly
  – a complete planner must be able to interleave subplans

Enter partial-order planning, state-of-the-art during the 1980s and 90s
– today mostly used for specific tasks, such as operations scheduling
– also used when it is important for humans to understand the plans
– e.g., operational plans for spacecraft and Mars rovers are checked
  by human operators before uploaded to the vehicles
Example: The Sussman anomaly

"Sussman anomaly" problem

Start State

\[ \text{Clear}(x) \ \text{On}(x,z) \ \text{Clear}(y) \]

\[ \text{PutOn}(x,y) \]

\[ \neg \text{On}(x,z) \ \neg \text{Clear}(y) \]

\[ \text{Clear}(z) \ \text{On}(x,y) \]

Goal State

\[ \text{Clear}(x) \ \text{On}(x,z) \]

\[ \text{PutOnTable}(x) \]

\[ \neg \text{On}(x,z) \]

\[ \text{Clear}(z) \ \text{On}(x,\text{Table}) \]

+ several inequality constraints
Example contd.

\begin{center}
\begin{tikzpicture}
  \node (start) at (0,0) {START};
  \node (a) at (1,0) {A};
  \node (b) at (2,0) {B};
  \node (c) at (3,0) {C};

  \draw[->] (start) -- (a);
  \draw[->] (a) -- (b);
  \draw[->] (b) -- (c);

  \node (end) at (5,0) {FINISH};
  \node (a') at (5,1) {A};
  \node (b') at (5,2) {B};
  \node (c') at (5,3) {C};

  \draw[->] (end) -- (a');
  \draw[->] (a') -- (b');
  \draw[->] (b') -- (c');

  \node at (0,-1.5) {On(C,A) \ On(A,Table) \ Cl(B) \ On(B,Table) \ Cl(C)};
\end{tikzpicture}
\end{center}
Example contd.

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

PutOn(B,C)

On(A,B) On(B,C)

FINISH

START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

Cl(B) On(B,z) Cl(C)
Example contd.

\[
\begin{align*}
\text{START} & \quad \text{On}(C,A) \quad \text{On}(A,Table) \quad \text{Cl}(B) \quad \text{On}(B,Table) \quad \text{Cl}(C) \\
\text{PutOn}(A,B) & \quad \text{PutOn}(B,C) \\
\text{FINISH} & \\
\end{align*}
\]

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)
Example contd.

On(A, Table) Cl(B) On(B, Table) Cl(C)

PutOnTable(C)

PutOn(A, B)

PutOn(B, C)

On(A, B) On(B, C)

START

On(C, A) Cl(B) On(C, z)

PutOnTable(C)

Cl(A) On(A, z) Cl(B)

PutOn(A, B)

PutOn(B, C)

On(C, z) Cl(C)

FINISH

PutOn(A, B) clobbers Cl(B) => order after PutOn(B, C)

PutOn(B, C) clobbers Cl(C) => order after PutOnTable(C)

Artificial Intelligence, spring 2013, Peter Ljunglöf; based on AIMA Slides ©Stuart Russel and Peter Norvig, 2004