EECS 3401 — AI and Logic Prog. — Lecture 15

Adapted from slides of Yves Lesperance

Vitaliy Batusov
vbatusov@cse.yorku.ca

York University

November 11, 2020

- Today: **Classical Planning**
- Required reading: Russell & Norvig Chapter 11 (Ch. 10 in 3-rd ed.)

Planning as a Search Problem

- Already understand: traversing the state space from starting state to goal state
- Planning is one of the applications of search (This lecture)
- But there is more to planning! (Next lecture)

Planning in general

What is planning? Consider a setup:

- The **world state** is described by a Knowledge Base of some kind
- **Actions** can update the KB, thus possibly changing the state of the world

  STRIPS, ADL — existing formal languages for that purpose
- Goal condition is a logical statement (formula)

The the **planning problem** is the task of finding a **sequence of actions** that, when applied to the initial KB, yield an **updated KB which satisfies the goal**.

Classical Planning

- STRIPS[1]: an early and very influential formalism for describing planning domains
- PDDL: current, robust formalsm which fixes semantic problems with STRIPS and adds cool new features
- "Automated Planning and Scheduling" is a sizable sub-area of AI, revolves mostly around PDDL and its extensions

[1] "Stanford Research Institute Problem Solver", 1971

STRIPS, briefly

STRIPS:

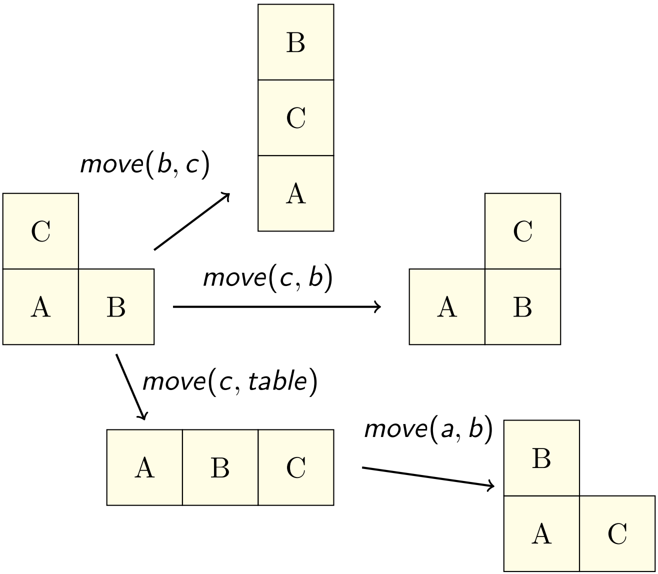
- A modest language (compared to FOL) for state descriptions—operates on propositional variables
- Describes world in terms of a finite set of True/False facts (prop. variables, "atoms")
- A state is given by a subset of the set of all domain atoms. An atom in this subset is true, not in this set is false
- Goal state: a set of atoms that we *want* to be true
- Operators: transform one state to new state. Defined by:
  - A name
  - A set of preconditions — atoms that must be true for the action to be possible
  - An *add-list* — atoms the action makes true
  - A *delete-list* — atoms the action makes false

Planning As Search

This setup is reminiscent of the search problem.

- The initial KB *represents* the initial state
- Each action maps one description of a state to a description of a new one
- The goal state is any state described by a KB which entails the goal statement

Example



Problems

- Search tree is generally quite large

  Randomly reconfiguring just 9 blocks takes thousands of CPU seconds
- The representation suggests some structure: Each action only affects a small set of facts, actions depend on each other via their preconditions
- Planning algorithms are designed to take advantage of the special nature of the representation

Reachability Analysis

- Let's consider just one technique: Relaxed Plan heuristics used with heuristic search
- Idea: consider what happens if we **ignore the delete list of actions**
- This yields a "relaxed problem" that can produce a useful heuristic estimate

- In the relaxed problem, actions add new facts, but never delete old facts
- Then we can do **reachability analysis**, which is much simpler than searching for a solution

Reachability

- Start with the initial state S_0
- Alternate between **state** and **action** layers
- Find all action swhose preconditions are contained in S_0. These actions comprise the first **action layer** A_0.
- The next **state layer** consists of all of S_0 plus the *add-lists* of all of the actions of A_0
- In general,

  A_i is the set of actions whose preconditions are contained in S_i

  S_{i+1} is $S_i \cup \{AddList(A) \mid A \in A_i\}$

STRIPS Blocks World operators

```
pickup(X)
Pre: {clear(X), ontable(X), handempty}
Add: {holding(X)}
Del: {clear(X), ontable(X), handempty}

putdown(X)
Pre: {holding(X)}
Add: {clear(X), ontable(X), handempty}
Del: {holding(X)}
```

STRIPS Blocks World operators

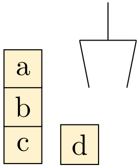
```
unstack(X,Y)
Pre: {clear(X), on(X,Y), handempty}
Add: {holding(X), clear(Y)}
Del: {clear(X), on(X,Y), handempty}

stack(X,Y)
Pre: {holding(X),clear(Y)}
Add: {on(X,Y), handempty, clear(X)}
Del: {holding(X),clear(Y)}
```

Example

S_0:

on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty

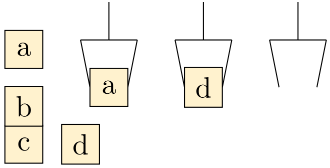


A_0:

unstack(a,b)
pickup(d)

S_1:

on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty,
holding(a),
clear(b),
holding(d)



Example

S_1:

```
on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty,
holding(a),
clear(b),
holding(d)
```

A_1:

```
putdown(a),
putdown(d),
stack(a,b),
stack(a,a),
stack(d,b),
stack(d,a),
pickup(d),
...
unstack(b,c)
...
```

S_2:

...

Reachability

- We continue until the goal G is contained in the state layer, or until the state layer no longer changes
- Intuitively,
  - The actions at level A_i are the actions that could be executed at the i-th step of *some* plan
  - The facts in level S_i are the facts that could be made true after *some* i-step plan
- This is generally false, but not always. More often, this is too optimistic—good enough for a heuristic.

Heuristics from Reachability Analysis

- Grow the levels until the goal is contained in the final state level S_k
- If the state level stops changing and the goal is not present, the goal is **cannot be achieved**. (Because the goal is a set of positive facts, and in STRIPS, all preconditions are positive facts as well.)

Heuristics from Reachability Analysis

- Let `CountActions`(G, S_k) be a procedure that computes the # of actions in relaxed plan to reach G
- Partition G into facts in S_{k-1} and elements in S_k only. These sets are the *previously-achieved* (G_{PA}) and *just-achieved* (G_{JA}) parts of G.
- Find a **minimal** set of actions A whose add-lists cover the G_{JA}.
- Replace the just-achieved part of G with the preconditions of A: Let $G' = G_{PA} \cup Pre(A)$
- Now, return `CountActions`(G', S_{k-1}) + [# of actions needed to cover G_{JA}]

Example

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$S_2 = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$G = \{f_1, f_5, f_6\}$

`CountActions`(G, S_2):

$G_{PA} = \{f_5, f_1\}$ already in S_1

$G_{JA} = \{f_6\}$ new in S_2

$A = \{a_3\}$ adds all in G_{JA}

$G' = G_{PA} \cup Pre(A) = \{f_5, f_1, f_2, f_4\}$

Return `CountActions`$(G', S_1) + 1$

Example

(Now at level S_1)

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$S_2 = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$G' = \{f_5, f_1, f_2, f_4\}$

`CountActions`(G', S_1):

$G_{PA} = \{f_1, f_2\}$ already in S_0

$G_{JA} = \{f_4, f_5\}$ new in S_1

$A = \{a_1, a_2\}$ adds all in G_{JA}

$G'' = G_{PA} \cup Pre(A) = \{f_1, f_2\}$

Return `CountActions`$(G'', S_0) + 2 = 2$

So, in total, `CountActions`$(G, S_2) = 1 + 2 = 3$

Using the Heuristic

To use `CountActions` as a heuristic:

- Build a layered structure from state S that reaches the goal
- Compute `CountActions` to see how many actions are required in a relaxed plan
- Use this count as our heuristic estimate of the distance of S to the goal
- (This heuristic tends to work better as a best-first search, i.e., when the cost of getting to the current state is ignored)

Admissibility

- An optimal-length plan in the relaxed problem (actions have no delete-lists) will be a lower bound on the optimal length of a plan in the real problem
- However, `CountActions` does **not** compute the length of the optimal relaxed plan
- **The choice of which action set to use** to achieve G_{JA} is not necessarily optimal
- In fact, it is NP-hard to compute the optimal-length plan even in the relaxed plan space
- Thus, `CountActions` will not be admissible
- **However**, empirically, refinements of this heuristic perform very well on a number of sample planning domains

Closed World Assumption

- Classical Planning = no incomplete or uncertain knowledge
- The Closed World Assumption underlies Classical Planning both in knowledge representation and reasoning
- The KB is a **list** of positive ground atomic facts
- CWA:
  - If a ground atomic fact is not in our list of "known" facts, its negation must be true
  - The constants mentioned in KB are all the domain objects

State of the Art in Planning

- There are annual AI planning systems competitions (e.g., IPC)
- PDDL is the standard language in the area, several dialects
- Several state of the art (classical) planning systems perform well on large real-world problems

Current Research in Planning

- Classical Planning makes very strong assumptions: complete information, deterministic actions, static single-agent world
- Much recent work in planning addresses more general forms of planning that drop some or all of these assumptions
- In such cases, a solution may be a *branching* plan (branching on observations/action outcomes), a finite state automaton, or a policy
- Hierarchical planning/abstraction is useful to address large real-world problems

End of Lecture

- Next time: **Actions, Situations, and GOLOG**