Artificial Intelligence-BSCE-306L

Module 6

Planning

Dr . Saurabh Agrawal

Faculty Id: 20165

School of Computer Science and Engineering

VIT, Vellore-632014

Tamil Nadu, India

Outline

- Classical Planning **(RN_C_10.1)**
- Planning as State-space search **(RN_C_10.2)**
- Forward search **(RN_C_10.2.1)**
- Backward search **(RN_C_10.2.2)**
- Planning graphs **(RN_C_10.3)**
- Hierarchical Planning **(RN_C_11.2)**
- Planning and acting in Nondeterministic domains **(RN_C_11.3)**
- Sensor-less Planning **(RN_C_11.3.1)**
- Multiagent planning **(RN_C_11.4)**

Classical Planning

- **What is planning?**
- "Devising a plan of action to achieve one's goals"

Planning = How do I get from here to there?

- Planning systems are problem-solving algorithms that operate on explicit propositional or relational representations of states and actions
- Planning problem: find a plan that is guaranteed (from any of the initial states) to generate a sequence of actions that leads to one of the goal states
- Planning problems often have large state spaces

❑ Automated Planning

❑ We will look at two popular and effective current approaches to automated classical planning:

- Forward state-space search with heuristics
- Translating to a Boolean satisfiability problem

❑ There are also other approaches

- e.g. planning graphs: data structures to give better heuristic estimates than other methods, and also used to search for a solution over the space formed by the planning graph

Classical Planning

- **Representing Planning Problems**
- Recall search based problem-solving agents
  - Find sequences of actions that result in a goal state BUT deal with *atomic states so need good domain specific* heuristics to perform well
- Planning represented by factored representation
  - Represent a state by a collection of variables
- Planning Domain Definition Language (PDDL)
  - Allows expression of all actions with one schema
  - Inspired by earlier STRIPS planning language

Classical Planning

- **Defining a Search Problem :** Define a search problem through:

1. Initial state
2. Actions available in a state
3. Result of action
4. Goal test

❑PDDL – Representing States (I)

- ❑A state is represented by a conjunction of fluents
- ❑These are ground, functionless atoms
  - ▪Example: At(Truck1,Manchester) $\land$ At(Truck2,Warrington)
- ❑Closed world assumption (no facts = false)
- ❑Unique names assumption (Truck1 distinct from Truck2)

Classical Planning

❑PDDL – Representing States (II)

❑Not allowed:

❑At(x,y) non-ground (i.e. variables alone)

- ¬ Poor negation
- At(Father(Fred),Liverpool) uses function

❑A state is treated as either

- conjunction of fluents, manipulated by logical inference
- set of fluents, manipulated with set operations

Classical Planning

❑PDDL – Representing Actions

❑Actions described by a set of action schemas that implicitly define Actions(s) and Result(s,a) functions

❑Classical planning: most actions leave most states unchanged

▪Relates to the Frame Problem: issue of what changes and what stays the same as a result of actions

❑PDDL specifies the result of an action in terms of *what changes* – *don't need to mention* everything that stays the same

❑Action Schema (I)

- Represents a set of ground actions
- Contains action name, list of variables used, precondition and effect
- Example: action schema for flying a plane from one location to another

Action(Fly(p,from,to),

PRECOND: At(p,from) ∧ Plane(p) ∧

Airport(from) ∧ Airport(to)

EFFECT: ¬ At(p,from)∧ At(p,to))

❑Action Schema (II)

❑Free to choose whatever values we want to instantiate variables

❑Precondition and effect of an action are each conjunctions of literals (positive or negated atomic sentences)

- Precondition defines states in which action can be executed
- Effect defines result of action

❑Sometimes we want to *propositionalise a PDDL problem* (replace each action schema with a set of ground actions) and use a propositional solver (e.g. SATPLAN) to find a solution

- More on this later…

❑Action Schema (III)

❑Action a can be executed in state s if s entails the precondition of a

$(a \in Actions(s)) \Leftrightarrow s \models Precond(a)$

where any variables in a are universally quantified

▪Example:

$\forall p, from, to\ (Fly(p, from, to) \in Actions(s)) \Leftrightarrow$

$s \models (At(p, from) \land Plane(p) \land Airport(from)$

$\land Airport(to))$

❑We say that a is applicable in s if the preconditions are satisfied by s

❑Action Schema (IV)

❑Result of executing action *a in state s (s')*

▪Result(s,a)=(s-Del(a)) U Add(a)

❑Delete list (Del(a)): fluents that appear as negative literals in action’s effect

❑Add list (Add(a)): fluents that appear as positive literals in action’s effect

❑Note that time is implicit: preconditions have time t, effects have t+1

Classical Planning

❑Planning Domain

❑A set of action schemas defines a planning domain

❑A specific problem within a domain is defined by adding initial state and goal

- Initial state: conjunction of ground atoms
- Goal: conjunction of literals (positive or negative) that may contain variables
- e.g. $At(p,LPL) \wedge Plane(p)$

❑Problem solved when we find sequence of actions that end in a state that entails the goal

- e.g. $Plane(Plane1) \wedge At(Plane1,LPL)$ entails the goal $At(p,LPL) \wedge Plane(p)$

Classical Planning

❑Example: Air Cargo Transport

Init($At(C_1,SFO) \land At(C_2,JFK) \land At(P_1,SFO) \land At(P_2,JFK) \land$
$Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land$
$Airport(JFK) \land Airport(SFO)$)

Goal($At(C_1,JFK) \land At(C_2,SFO)$)

Action($Load(c,p,a)$,
PRECOND: $At(c,a) \land At(p,a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c,a) \land In(c,p)$)

Action($Unload(c,p,a)$,
PRECOND: $In(c,p) \land At(p,a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c,a) \land \neg In(c,p)$)

Action($Fly(p,from,to)$,
PRECOND: $At(p,from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p,from) \land At(p,to)$)

❑Example: Air Cargo Transport

❑Problem defined with 3 actions

❑Actions affect 2 predicates

❑When a plane flies from one airport to another, all cargo inside goes too

- in PDDL we have no explicit universal quantifier to say this as part of the Fly action
- so instead we use the load/unload actions:
  - ❑cargo seizes to be At the old airport when it is loaded
  - ❑and only becomes At the new airport when it is unloaded

❑A solution plan:

[Load(C1,P1,SFO),Fly(P1,SFO,JFK),Unload(C1,P1,JFK),

Load(C2.P2.JFK),Fly(P2,JFK,SFO),Unload(C2,P2,SFO)].

❑Problem – spurious actions like *Fly(P1,JFK,JFK) have contradictory effects*

- Add inequality preconditions ∧ *(from ≠ to)*

Planning as State-space search

- Now we turn our attention to planning algorithms.
- We saw how the description of a planning problem defines a search problem: we can search from the initial state through the space of states, looking for a goal.
- One of the nice advantages of the declarative representation of action schemas is that we can also search backward from the goal, looking for the initial state.
- Figure 10.5 compares forward and backward searches.

Planning as State-space search

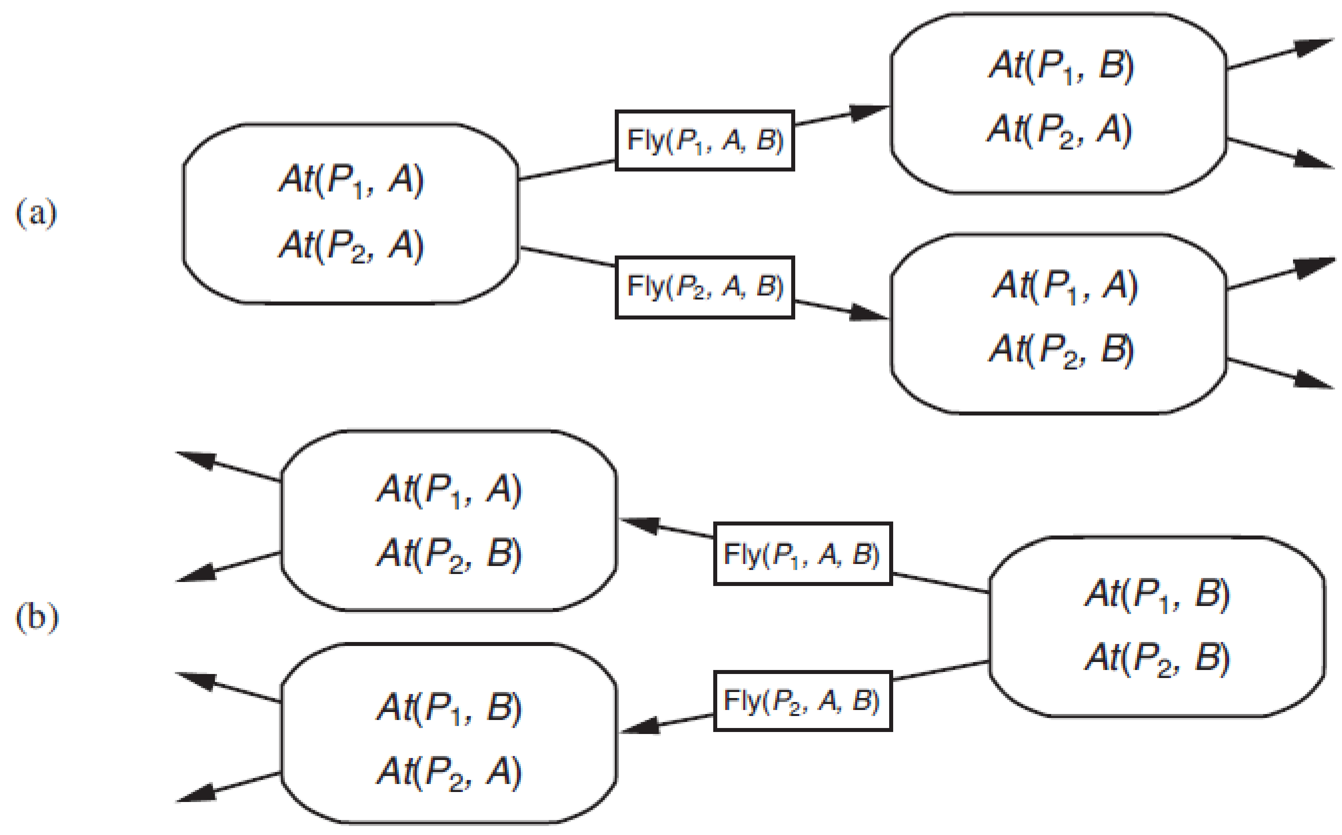

Figure 10.5 Two approaches to searching for a plan. (a) Forward (progression) search through the space of states, starting in the initial state and using the problem's actions to search forward for a member of the set of goal states. (b) Backward (regression) search through sets of relevant states, starting at the set of states representing the goal and using the inverse of the actions to search backward for the initial state.

Forward (progression) State-space search

- Now that we have shown how a planning problem maps into a search problem, we can solve planning problems with any of the heuristic search algorithms or a local search algorithm (provided we keep track of the actions used to reach the goal).
- First, forward search is prone to exploring irrelevant actions.
- Consider the noble task of buying a copy of *AI: A Modern Approach from an online bookseller.*
- *Suppose there is an* action schema Buy(isbn) with effect Own(isbn).
- ISBNs are 10 digits, so this action schema represents 10 billion ground actions. An uninformed forward-search algorithm would have to start enumerating these 10 billion actions to find one that leads to the goal.

Forward (progression) State-space search

- Second, planning problems often have large state spaces.
- Consider an air cargo problem with 10 airports, where each airport has 5 planes and 20 pieces of cargo.
- The goal is to move all the cargo at airport A to airport B.
- There is a simple solution to the problem: load the 20 pieces of cargo into one of the planes at A, fly the plane to B, and unload the cargo.
- Finding the solution can be difficult because the average branching factor is huge: each of the 50 planes can fly to 9 other airports, and each of the 200 packages can be either unloaded (if it is loaded) or loaded into any plane at its airport (if it is unloaded).
- So in any state there is a minimum of 450 actions (when all the packages are at airports with no planes) and a maximum of 10,450 (when all packages and planes are at the same airport).
- On average, let's say there are about 2000 possible actions per state, so the search graph up to the depth of the obvious solution has about 2000^{41} nodes.

Forward (progression) State-space search

- Clearly, even this relatively small problem instance is hopeless without an accurate heuristic.
- Although many real-world applications of planning have relied on domain-specific heuristics, it turns out that strong domain-independent heuristics can be derived automatically; that is what makes forward search feasible.

Backward (regression) relevant states search

- In regression search we start at the goal and apply the actions backward until we find a sequence of steps that reaches the initial state.
- It is called **relevant-states search because we** only consider actions that are relevant to the goal (or current state).
- As in belief-state search , there is a set of relevant states to consider at each step, not just a single state.
- We start with the goal, which is a conjunction of literals forming a description of a set of states—for example, the goal $\neg Poor \wedge Famous$ describes those states in which Poor is false, Famous is true, and any other fluent can have any value.
- If there are n ground fluents in a domain, then there are 2^n ground states (each fluent can be true or false), but 3^n descriptions of sets of goal states (each fluent can be positive, negative, or not mentioned).

Backward (regression) relevant states search

- In general, backward search works only when we know how to regress from a state description to the predecessor state description.
- For example, it is hard to search backwards for a solution to the n-queens problem because there is no easy way to describe the states that are one move away from the goal.
- Happily, the PDDL representation was designed to make it easy to regress actions—if a domain can be expressed in PDDL, then we can do regression search on it.
- Given a ground goal description g and a ground action a, the regression from g over a gives us a state description g' defined by.

$$g' = (g - ADD(a)) \cup Precond\ (a)$$

Backward (regression) relevant states search

- That is, the effects that were added by the action need not have been true before, and also the preconditions must have held before, or else the action could not have been executed.
- Note that DEL(a) does not appear in the formula; that's because while we know the fluents in DEL(a) are no longer true after the action, we don't know whether or not they were true before, so there's nothing to be said about them.
- To get the full advantage of backward search, we need to deal with partially uninstantiated actions and states, not just ground ones.
- For example, suppose the goal is to deliver a specific piece of cargo to SFO: At(C_2, SFO). That suggests the action Unload(C_2, p', SFO):

$$
\begin{aligned}
&Action(Unload(C_2, p', SFO),\\
&\quad \text{PRECOND:} In(C_2, p') \land At(p', SFO) \land Cargo(C_2) \land Plane(p') \land Airport(SFO)\\
&\quad \text{EFFECT:} At(C_2, SFO) \land \neg In(C_2, p')\,.
\end{aligned}
$$

Backward (regression) relevant states search

- (Note that we have **standardized variable names (changing p to p'** in this case) so that there will be no confusion between variable names if we happen to use the same action schema twice in a plan.
- This represents unloading the package from an *unspecified plane at SFO; any plane will do,* but we need not say which one now.
- We can take advantage of the power of first-order representations: a single description summarizes the possibility of using *any of the planes by* implicitly quantifying over p'.
- The regressed state description is

$$g' = In(C_2, p') \land At(p', SFO) \land Cargo(C_2) \land Plane(p') \land Airport(SFO)\,.$$

- The final issue is deciding which actions are candidates to regress over.
- In the forward direction we chose actions that were **applicable—those actions that could be the next step in the** plan.
- In backward search we want actions that are **relevant—those actions that could be the** *last step in a plan leading up to the current goal state.*

Backward (regression) relevant states search

- For an action to be relevant to a goal it obviously must contribute to the goal: at least one of the action's effects (either positive or negative) must unify with an element of the goal.
- What is less obvious is that the action must not have any effect (positive or negative) that negates an element of the goal.
- Now, if the goal is $A \wedge B \wedge C$ and an action has the effect $A \wedge B \wedge \neg C$ then there is a colloquial sense in which that action is very relevant to the goal—it gets us two-thirds of the way there.
- But it is not relevant in the technical sense defined here, because this action could not be the *final step of a solution—we would always need at least* one more step to achieve C.

Backward (regression) relevant states search

- Given the goal At(C_2, SFO), several instantiations of Unload are relevant: we could chose any specific plane to unload from, or we could leave the plane unspecified by using the action Unload(C_2, p', SFO).
- We can reduce the branching factor without ruling out any solutions by always using the action formed by substituting the most general unifier into the (standardized) action schema.
- As another example, consider the goal Own(0136042597), given an initial state with 10 billion ISBNs, and the single action schema

$$A = Action(Buy(i), \text{PRECOND}: ISBN(i), \text{EFFECT}: Own(i))$$

Backward (regression) relevant states search

- As we mentioned before, forward search without a heuristic would have to start enumerating the 10 billion ground Buy actions.
- But with backward search, we would unify the goal Own(0136042597) with the (standardized) effect Own(I'), yielding the substitution θ = {i'/0136042597}.
- Then we would regress over the action Subst(θ, A') to yield the predecessor state description ISBN (0136042597).
- This is part of, and thus entailed by, the initial state, so we are done.

Backward (regression) relevant states search

We can make this more formal. Assume a goal description g which contains a goal literal g_i and an action schema A that is standardized to produce A'. If A' has an effect literal e'_j where $Unify(g_i, e'_j) = \theta$ and where we define $a' = \text{SUBST}(\theta, A')$ and if there is no effect in a' that is the negation of a literal in g, then a' is a relevant action towards g.

Backward search keeps the branching factor lower than forward search, for most problem domains. However, the fact that backward search uses state sets rather than individual states makes it harder to come up with good heuristics. That is the main reason why the majority of current systems favor forward search.

Note for Students

- This power point presentation is for lecture, therefore it is suggested that also utilize the text books and lecture notes.
- Also Refer the solved and unsolved examples of Text and Reference Books.