Module 9

Planning

Lesson 24

Planning algorithm - I

Version 1 CSE IIT, Kharagpur

9.4 Planning as Search

Planning as Search:

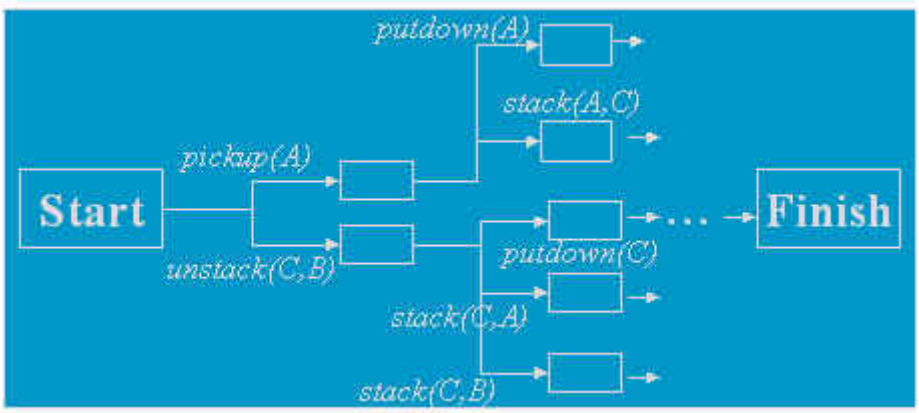
There are two main approaches to solving planning problems, depending on the kind of search space that is explored:

1. Situation-space search
2. Planning-space search

9.4.1 Situation-Space Search

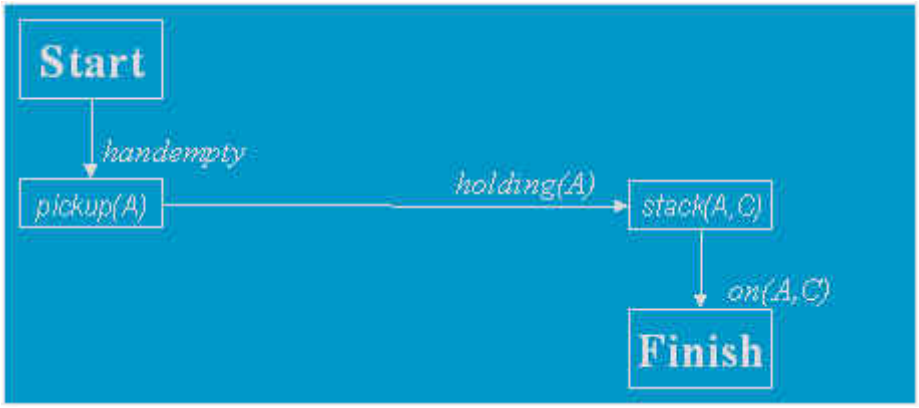
In situation space search

- the search space is the space of all possible states or situations of the world
- initial state defines one node
- a goal node is a state where all goals in the goal state are satisfied
- a solution plan is the sequence of actions (e.g. operator instances) in the path from the start node to a goal node



9.4.2 Plan-Space Search

- the search space is the space of all possible plans
- a node corresponds to a partial plan
- initially we will specify an "initial plan" which is one node in this space
- a goal node is a node containing a plan which is complete, satisfying all of the goals in the goal state
- the node itself contains all of the information for determining a solution plan (e.g. sequence of actions)



9.4.3 Situation-Space Planning Algorithms

There are 2 approaches to situation-space planning:

1. Progression situation-space planning
2. Regression situation-space planning

Progression Planning:

- *Forward-chaining* from initial state to goal state
- Looks just like a state-space search except STRIPS operators are specified instead of a set of next-move functions
- You can use any search method you like (i.e. BFS, DFS, A*)
- **Disadvantage**: huge search space to explore, so usually very inefficient

Algorithm:

1. Start from initial state
2. Find all operators whose preconditions are true in the initial state
3. Compute effects of operators to generate successor states
4. Repeat steps #2-#3 until a new state satisfies the goal conditions

The work through of the progression algorithm for the Blocks World example is shown below:

Step	State	Applicable Operators	Operator Applied
#1	ontable(A) Λ ontable(B) Λ on(C, B) Λ clear(A) Λ clear(C) Λ handempty	pickup(A) unstack(C,B)	pickup(A)
#2	~ontable(A) Λ	putdown(A) stack(A,C)	stack(A,C)

	ontable(B) Λ on(C, B) Λ ~clear(A) Λ clear(C) Λ ~handempty Λ holding(A)		
#3	ontable(B) Λ on(C, B) Λ on(A, C) Λ clear(A) Λ ~clear(C) Λ handempty Λ ~holding(A)	Matches goal state so STOP!	

Regression Planning

- *Backward-chaining* from goal state to initial state
- Regression situation-space planning is usually more efficient than progression because many operators are applicable at each state, yet only a small number of operators are applicable for achieving a given goal
- Hence, regression is more goal-directed than progression situation-space planning
- **Disadvantage***:* cannot always find a plan even if one exists!

Algorithm:

1. Start with goal node corresponding to goal to be achieved
2. Choose an operator that will *add* one of the goals
3. Replace that goal with the operator's preconditions
4. Repeat steps #2-#3 until you have reached the initial state
5.

While backward-chaining is performed by STRIPS in terms of the generation of goals, sub-goals, sub-sub-goals, etc., operators are used in the forward direction to generate successor states, starting from the initial state, until a goal is found.

The work through of the regression algorithm for the Blocks World example is shown below.

Step	State	Stack	Plan	Note
#1	ontable(A) Λ ontable(B)	achieve(on(A,C))		Stack contains original goal. State contains the initial state

	Λ on(C, B) Λ clear(A) Λ clear(C) Λ handempty			description.
#2	Same.	achieve(clear(C), holding(A), apply(Stack(A,C)) achieve(on(A,C))		Choose operator Stack to solve goal popped from top of goal stack.
#3	Same.	achieve(holding(A)) achieve(clear(C)) achieve(clear(C), holding(A), apply(Stack(A,C)) achieve(on(A,C))		Order sub-goals arbitrarily.
#4	Same.	achieve(ontable(A), clear(A), handempty), apply(pickup(A)) achieve(holding(A)) achieve(clear(C)) achieve(clear(C), holding(A), apply(Stack(A,C)) achieve(on(A,C))		Choose operator *pickup* to solve goal popped from top of goal stack.
#5	ontable(B) Λ on(C, B) Λ clear(C) Λ holding(A)	achieve(clear(C)) achieve(clear(C), holding(A), apply(Stack(A,C)) achieve(on(A,C))	Pickup(A)	Top goal is true in current state, so pop it and apply operator *pickup(A).*
#6	ontable(B) Λ on(C, B) Λ on(A,C) Λ clear(A) Λ handempty	achieve(on(A,C))	pickup(A) stack(A,C)	Top goal *achieve(C)* true so pop it. Re-verify that goals that are the preconditions of the *stack(A,C)* operator still true, then pop that and the operator is applied.
#7	Same.	<empty>		Re-verify that original goal is true in current state, then pop and halt with empty goal stack and state description satisfying original goal.

Goal Interaction

Most planning algorithms assume that the goals to be achieved are independent or nearly independent in the sense that each can be solved separately and then the solutions concatenated together. If the order of solving a set of goals (either the original goals or a set of sub-goals which are the preconditions of an operator) fails because solving a latter goal undoes an earlier goal, then this version of the STRIPS algorithm **fails.** Hence, situation-space planners do not allow for interleaving of steps in any solution it finds.

Principle of Least Commitment

The principle of least commitment is the idea of never making a choice unless required to do so. The advantage of using this principle is you won't have to backtrack later!
In planning, one application of this principle is to never order plan steps unless it's necessary for some reason. So, partial-order planners exhibit this property because constraint ordering steps will only be inserted when necessary. On the other hand, situation-space progression planners make commitments about the order of steps as they try to find a solution and therefore may make mistakes from poor guesses about the right order of steps.