

Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 7 and 10
A First Course in Artificial Intelligence
Lecture – 61
Goal Stack Planning (GSP)

(Refer Slide Time: 00:14)

Forward and Backward: Exploring the space



	FSSP	BSSP
Start at	Start State	Goal Description
Moves	<i>a is applicable</i> $pre(a) \subseteq S$	<i>a is relevant</i> $\{effect^+(a) \cap G\} \neq \phi \wedge \{effects^-(a) \cap G\} = \phi$
Transition	Progression $S' = \{S \cup effects^+(a)\} \setminus effects^-(a)$ Sound $\pi \leftarrow \pi \circ a$	Regression $G' = \{G \setminus effects^+(a)\} \cup pre(a)$ not Sound $\pi \leftarrow a \circ \pi$
GoalTest	$G \subseteq \gamma(S_0, \pi)$	$\gamma^{-1}(G, \pi) \subseteq S_0$ followed by validity check



So, welcome back. Here we are looking at planning as area of work, which has in some ways emerged out of AI and establish itself as a field in itself. And as I said earlier, we have conferences devoted entirely to planning. So, we started by looking at the simplest of planning domains, the strips like domains and we looked at two algorithms; forward state space planning and backward state space planning.

Let us do a quick recap of these two. We had observed that each of them has some advantage and some disadvantage, like most algorithms have and we always strive to improve upon them. So, how do these two algorithms explore the space? One is called forward state space planning which starts from the start state and walks towards the goal, and backwards state space planning starts from the goal state and walks backwards towards the start state.

Backward state space planning is also kind of associated with backward reasoning which the people who work in logic had said that humans often tend to do goal directed reasoning, and they want to think about the goals that they want to achieve and not about everything else that can happen in the world. And in that sense backward state space planning does exactly that, and that is why we had observed that it has lower branching factor.

So just to recap, the forward state space planning starts from the start state and moves forward, and backward state space planning works with the goal description and moves backwards. A move is selected by forward planning if it is applicable and it is applicable if its preconditions are true in a given state. Remember that the state is described as terms of in terms of sentences, and we will talk a little bit about this in the next few moments.

In backward planning an action is selected if it is relevant, and it is relevant if it produces something that is there in the goal and also it does not delete anything which is there in the goal. So we had expressed this as kind of set properties. If you move forward then we call that move progression. So you are in a given state and you apply an action a and then you go to a new state S' that can be computed by adding the positive effects of a and deleting the negative effects of a .

In regression which is in backward state space planning you move from a goal to a sub goal or a new goal G' by adding the positive effects of a , because that is what anyway the action is going to generate and sorry removing the positive effects of a because that is anyway what the action is going to generate and adding the preconditions of a because that is necessary for that action to be applicable.

But we had seen in the example with backward state space planning that you can regress those set of sentences or formulas which are not necessarily feasible essentially, so we said that regression is not sound. Then backward state space planning constructs a plan from the last action to the first action. So the last action is added first, and that is depicted by this way that the plan is augmented, that you take the old plan and put the new action at the head of the plan.

Forward state space planning does the opposite, it builds a plan from the first action onwards, it takes up current plan and adds a new plan at the end of the sequence. Both the algorithms start with an empty plan.

The termination criteria you would notice is almost identical in both; in forward state space planning you have reached a new state through series of progress movements, and you want to verify that the goal that you want to achieve is true in that new state. The new state is given in this case by the state transition function that we had seen γ that you start with a 0 and apply the plan π ; π is a sequence of actions.

You reach a new state which is depicted by $\gamma S_0 \pi$ and you check whether the goal predicates to the goal description satisfies that particular state. We do something similar for backward state space planning. In backward state space planning we have regressed to a goal which we had called as G_1 , if you remember and that is run by regression from G_n with n actions.

And you want to check whether this new this regressed goal belongs to the start state essentially. If that is the case then you do not need to do anything further, but we had seen that it has to be followed by a validity check.

(Refer Slide Time: 05:34)

Goals: some terminology



The start *state* and the goal *description* are described as *sets of sentences* expressed as *predicates* in *first order logic**

The predicates, like $on(A,B)$ and $holding(D)$ in the blocks world, are defined in the specific PDDL language used to *define the domain*.

We often refer to these predicates as *goals*, in the spirit of *goal-directed planning* in which *each of them is a goal that needs to be true in the goal state*.

The goal of planning $G = \{g_1, g_2, g_3\}$
- for example $\{on(A,B), on(B,C)\}$
- a *partial description of a state*
- represents a *set of states* containing the goals.

A *state* is a set in which *all the goals are true*.

For an in depth study of logic see our course
- Artificial Intelligence: Knowledge Representation & Reasoning

Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras



Because we saw, that certain action sequences which can be produced are not valid plans essentially. So, Let us just look at the terminology that we often use in planning. The start state and the goal description are both described in terms of sentences. So the sentence word comes from a formal background in particular first order logic. And a first order logic and other logics also formal languages and every element of that is a sentence.

And in our way of representing things the state and the goal description are both sets of sentences. I would encourage you to look at a course that I offer on artificial intelligence knowledge representation and reasoning for a more detailed study of logic. We will do a little bit towards the end of this course but not too much. So, the predicates or sentences like $on(A,B)$ and $holding(D)$ and so on, are defined in the specific PDDL language that we choose essentially.

So, we had seen that there are variations, that you know there can be for example, metric qualities involved and so on. We often refer to these sentences as goals, in the spirit of goal directed planning in which each of them is a goal that needs to be true in the goal state. So instead of calling them sentence, we will call them goals. It is just a matter of terminology for planning. We have a goal which is made up of a set of goals or individual goals g_1 g_2 and g_3 .

For example, we may have a goal that you want A on B and B on C; these are two individual goals. And the set of goals is the goal of planning that you want to achieve. Now, the a set of such sentences or goals is a partial description of the state because we are not specified, what else is true in the world which we call as a goal state.

So in that sense, because the other things could be anything, we can think of a goal description as a representing a set of goal states. A state, if we want to use a goal terminology is a set of such sentences or goals in which all the goals are true.

(Refer Slide Time: 07:56)

Planning operators: the arrow of time

A planning operator is defined with the *arrow of time* built in implicitly.
- It is essentially *designed to progress in the state space*.



When an *applicable* action *a* is applied in a state *S*

- effect⁺(*a*) are added to *S*, because they *become true*
- effect⁻(*a*) are deleted from *S*, because they *become false*
- the resulting set of goals *S'* define a *valid state*
- progression is *closed* over the state space

A *relevant* action *a* addresses one or more goals in *G*

- regression over *a* adds the pre(*a*) to *G*
- and *carries backwards* the *other goals*
- but the resulting set *G'* *may not be feasible*
- e.g. {**holding(A)**, clear(B), **holding(B)**, clear(C)}
- regression is *not closed* over the state space
- BSSP *requires a check* and possibly extra work



One thing we would like to think a little bit about is, why do we get these spurious actions in backward state space planning? And the reason for that is, if you think a little bit about this, that a planning operator is defined with the arrow of time kind of built in implicitly.

It is essentially designed to progress in the state space. So, what does the planning operator look like? It says that it has a set of preconditions, and the preconditions must be true in the state in which you want to apply the operator.

And then it defines that, if you apply the operator what will be the new state. So in some sense you are going from the previous state to the next state, so the sense of direction is already there in the planning operator.

So any so progression; for example, is aligned with this arrow of time, we can call it that. And it works wonderfully, but regression goes in the opposite direction and we have seen that it does not always work well, and that is also partly because we are only describing the goal state partially.

So, when an applicable action is applied to a state, then its positive effects are added to S, because they become true in the new state, its negative effects are deleted from S, because they become false in the new state, and the resulting state in fact, is a valid state. It means that it is a consistent description of some possible world. Progression is closed and the goal we say, that it is closed over state space that if you progress over an action then the new state that you reach will be a valid state or it will be in the state space.

In backwards planning a relevant action addresses some one or two more or goals of the in the goal set that we have we can call capital G as a goal set and in and the small g lowercase. G as the individual goals and a relevant action addresses one or more individual goals under this thing.

And when it regresses over a goal, when we regress with an action over a goal then it basically adds the preconditions of that action to the set of goal and carries forward the other goals or the other individual goals that are there in the goal set.

In doing so, it does not really bother about applicability and the resultant regressed goal may not be feasible. We saw this as an example, that at some point in backward state space planning, we had ended up with a goal description in which you are holding A and holding B.

Now, given the fact that we were working with a one armed robot this was not possible, but the algorithm has no way of knowing this. I had also mentioned in passing that there are people who had tried to look at this goal descriptions from a orthogonal perspective and trying to see whether they are consistent or not.

So that, you could prune them out while you are doing backward search, but we will not venture into that. As a consequence, situation was not closed over the state space and when our termination criteria was met, which said that the goal description is a subset of the current start state we have to verify that the sequence of actions returned to us by the algorithm is indeed a plan or not.

And verifying whether it is a plan or not is straightforward, because you will just progress from the start state using the plan that has been returned to you, and if you end up in a goal state then it is a valid plan.

(Refer Slide Time: 11:33)

Goal Stack Planning (GSP)

FSSP - searches for *applicable actions* from the start state S
- constructs the plan from the start state
- the *first action found* is the *first action in the plan*
- suffers from *high branching* since S is a full description

BSSP - searches for *relevant actions* from the goal description G
- construct the plan from the the goal description
- the *first action found* is the *last one in the plan*
- suffers from *spurious sub-goals*

Goal Stack Planning is an algorithm designed to
- search in a *goal directed backward* manner
- to take advantage of low branching
- but *construct plans in a forward* manner
- adding actions that are *applicable*



Now, let us look at an algorithm called gold stack planning. It is kind of combines the best features of the two algorithms that we have seen. Forward state space planning searches for applicable actions from the start state. It constructs a plan from the start state, we have been

observing this. An important point is that the first action that this algorithm finds is the first action in the plan as well. In other words, it will be the first action that will be executed when the plan is executed.

But, it has this problem that it was suffering from high branching factor, because given state was a complete description of the domain of the world and many many actions would have been possible, and forward state space planning would have to consider all of them. So the branching factor was high here. Backward state space planning searches for relevant actions from the goal description G . It constructs a plan from the in a backward fashion. The first action that is it finds is the last one in the plan.

So in that sense, backward state space planning first finds the last action that will achieve something in the goal state regresses to a new goal; let us call it G prime, looks for an action that will achieve something in G prime regresses to another goal Let us call it G double prime and so on. So, the actions that it finds are in the reverse order, it finds the last action first and then so on. But we saw that it suffers from spurious sub goals.

So, this algorithm goal stack planning is designed to search in a goal directed backward manner, which means; it will be able to take advantage of the low branching that backward state space planning has. And, if you remember we had said that the branching factor branching is low because a goal is only a partial description of what you want to achieve and it. So, backward state spaces focuses only on the goal what you want to achieve.

Goal stack planning on the other hand will construct the plans in a forward manner. Which means, that when it adds a action to a plan, it would be sure that the action is applicable and we have said that the process of progression is sound essentially. So, it will construct the plan from the first action onwards, but it will search for that plan from the last action or the last goal that has to be the final goal that has to be achieved.

(Refer Slide Time: 14:11)

GSP: Linear Planning

The basic idea of goal stack planning is to

- break up a compound goal into individual goals
- and solve them serially one by one
- producing a plan that is a sequence of actions
- a linear plan

It is best suited to domains where goals

- can be solved serially one at a time
- for example, cook three dishes
- each goal can be solved independently

GSP employs a stack in which

- goals are pushed in (in a backward manner)
- along with actions that could achieve them



We say, that goal stack planning does what we call as linear planning. And, let me explain that, the basic idea of goal stack planning is to break up a compound goal. So for example, $g_1 g_2 g_3$ or whatever into individual goals and solve them serially one by one. So what we would say, is that for example, if you have to achieve three goals g_1 and g_2 and g_3 , then first solve; let us say, g_2 in some order then maybe you solve g_3 and then maybe you solve g_1 .

So, you solve the goals individually one by one and in that sense we are solving them in a linear fashion. We also call it a linear planning algorithm because the plan that it produces is a sequence of actions, it is a linear plan. Now for the small domain that we have been working with in which a one arm robot is doing something, you can only produce linear plans

because the robot can do only one thing at a time. It can pick up a block, or put down a block, or unstack a block, or stack a block or something else, but only one at a time.

If we were to augment this, and maybe we will do this in one of these next classes, that if we had two arm robot which was now in the same blocks for domain then you can imagine that it could do two actions at the same time and we would have to revise this notion of what is a plan, and it would no longer necessarily be a linear plan. It mean it will have, perhaps a partial order or something like that. This notion of a linear plan is also challenged if we move on to richer domains.

For example, if you have working with derivative actions or temporal planning as it is called, then very often if you have more than one action possible in parallel. They will have different durations and therefore, this whole notion of stepping through time would kind of not be applicable anymore. So, this goal stack planning is best suited to domains where the goals can be solved serially one at a time.

For example, if you have to cook three dishes; then you are saying I will make the first dish, then I will make the second dish, and then I will make the third dish. And each of these three dishes can presumably be solved independently.

Now, goal stack planning employs the stack as a data structure in which you push the goals that you want to solve in a backward manner. So, if your initial goal is $g_1 g_2 g_3$, you push all the three goals in some order into the stack then keep popping them and seeing how to solve each individual goal separately.

So, along with the goals you would as we will see the algorithm in a short while, we also push in actions which are required to solve that goal. But we push the action into the stack and we do not add it to the plan because first we want to make sure that the conditions for action being applicable are true. So, first we would like to make sure that the current state is indeed a state in which the action is applicable, and only then you would add the action to the plan.

(Refer Slide Time: 17:31)

Function PushSet(G)



A basic operation in GSP is to

- push a compound goal $G = \{g_1, g_2, \dots, g_n\}$
- along with the individual goals g_1, g_2, \dots, g_n
- to be solved in *last in first out* order
- possibility of using a heuristic function
- or some form of reasoning

The reason one has to insert the compound goal too is to check whether after having solved the individual goals independently the compound goal has indeed been solved

- we will illustrate this need with an example



So, we will use as part of our algorithm a function which I called as push set; it basically says it is a push action in which you put something push something onto the stack, but what you are pushing is the set is the compound goal which is made up of a set of individual goals. So the compound goal is uppercase G, capital G here; the individual goals are g_1, g_2 up to g_n . So you push the entire compound goal into this onto the stack.

When you push the individual goals in some order g_1, g_2, g_n , and remember that a stack has a property of being a last in first out data structure. So the first goal that you push in out of this g_1 to g_n would be the last one to be addressed and the last one to be pushed in would be the first one to be addressed.

So, the goal ordering may matter as we will see. Now you could use the heuristic function here, to determine in what order you should put the goals in or you could do some kind of reasoning, which looks at the state looks at the current state looks at the goal state.

And so we have three states that at any point of time; we have the start state we from where we started, we have the goal description where we want to end up and we may have a current state at any point of time. So you could do some kind of reasoning, as to in the current state which of the possible goals can be solved more easily and use that to order the goals essentially.

The reason why you have to insert the compound goal which is the set of all the individual goals is to check that when we are about to terminate whether all the goals have been achieved or not.

We saw in backward state space planning that you could end up with a sequence of actions, but if you were to execute those sequence of action then all the goals would not have been achieved. And we will see this illustrate this with an example, it is easier done with an example.

(Refer Slide Time: 19:41)

When a goal is popped from the stack...

The algorithm goal stack planning maintains the current state S at all times.



When a goal g is popped from the stack, there are two possibilities

1. $g \in S$

When the goal is true in the current state nothing needs to be done.

2. $g \notin S$

When the goal is not true

- GSP pushes a relevant action a onto the stack
- Followed by $\text{PushSet}(\text{pre}(a))$
- i.e. the preconditions of a as a compound goal
- and the individual goals in $\text{pre}(a)$ in some order

If an action a is popped then

it *must be applicable* and can be added to the plan

- $\pi \leftarrow \pi \circ a$
- $S' \leftarrow \text{Progress}(S, a) = \{S \cup \text{effects}^+(a)\} \setminus \text{effects}^-(a)$



So the algorithm is basically as followed, we said that we will push the goals into the stack and then we will pop stuff out of the stack essentially one by one. And when a goal is popped from a stack there are two possibilities; and we are talking about individual goals here, so this g_1, g_2, g_3 that we mentioned. Either that goal is true in the given state. So as I said, the algorithm maintains the current state S at all times.

If the goal that we are trying to solve is a member of that state description which is a set of sentences then we do not have to do anything and we can move on to the next goal if there is one. But, if the goal is not present in the state description; that means, the goal is not true in the current state, then what goal stack planning does? It pushes a relevant action onto the stack.

So, let us call that action a , it pushes a relevant action a onto the stack followed by its preconditions. And we use this function which set we define, so we will first put in the conjunct or the compound goal and then we would put in the individual preconditions one by one. In some order, again as I said maybe you could use some heuristics or some form of reasoning to decide which one to do. The other thing that you can pop because you are pushing actions also onto the stack is action.

Now, observe that we have said that we push an action a onto the stack followed by push set which pushes the compound goals and the individual goals. So first action is pushed in then the compound goal, then the individual goals. So, if we have managed to pop out all the individual goals and found that they are true in the current state and then the compound goal also would be true in the current state. Then the action would be applicable essentially.

So, at the point when action a is popped out it must be applicable in the current state and we can add it to the plan, and as we have said earlier we add it as the last section in the plan being constructed. And we progressed to the new state using this action and that becomes the current state. So that is the basic algorithm that works for goal stack planning. So, let us quickly describe the algorithm in a little bit detail and then we will see a couple of examples as to how it works.

(Refer Slide Time: 22:30)


A planning problem Algorithm GSP

```
GSP(givenState, givenGoal, actions)
1  S ← givenState; plan ← (); stack ← emptyStack
2  PushSet(givenGoal, stack)
3  while not Empty(stack)
4      do x ← Pop(stack)
5      if x is an action a
6          then plan ← (plan ◦ a)
7              S ← Progress(S, a)
8      else if x a compound goal G and G is not true
9          then PushSet(G, stack)
10     else if x is a goal g and g ∉ S
11         then CHOOSE a relevant action a that achieves g
12             if none then return FAILURE
13             Push(a, stack)
14             PushSet(Pre(a), stack)
15 return plan
```

Backward search

Forward plan construction

Non-deterministic choice - replace with backtracking



Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT M

So, the algorithm takes three arguments. In fact, a planning problem is defined by these three arguments. A planning problem is defined by specifying the set of actions or operators that are available to the planner, the initial state that the planner is in, and the given goal description that you want to achieve. So these three things; the initial state, the goal description and the set of actions defines a planning problem. And goal stack planning algorithm takes a planning problem as an input.

So, let us call *S* the given state, because when it starts constructing the plan that is where it is initially it has an empty plan and there is nothing in the stack. So, as we said its main action is that you push the goal set or the given goal which is compound goal onto the stack, and you can see that this is the flavour of backward reasoning that you are trying to figure out. What are the relevant actions that will finally, let me achieve the goal that I want to achieve. To

facilitate that, we push the goals onto the stack. And then of course, we will pop them out one by one and try to achieve them.

So, the algorithm proceeds that as long as there is something in the stack. So what are what is in the stack? It is either some goal that you want to achieve or it is some action which has been selected which is relevant to some goal that you want to achieve. And you want to make sure that the preconditions for that action are true, which means that the action is applicable. And you will only schedule the action or add the action to the plan, when you are sure that its preconditions are true.

So the stack basically contains goals and actions, and you will pop them and one of the two will come out. And once you have addressed all the goals, and when you say, if you rule out the actions that we had proposed there is nothing else to do. So, this algorithm just repeats itself as long as the stack is not empty. We have already pushed at this point the initial goal set given to us, the given goal onto the stack and then it will start by addressing the initial goals essentially.

So when you do the pop action first time, the last of the individual goals that we had inserted into the stack or pushed on to the stack will pop out and we will see what to do with it. So, let us say x is the element that is popped from the stack it could either be individual goal or it could be a compound goal or it could be an action. And let us see what we do in each of the three cases.

So, if it is an action a , then we have just argued that it must be applicable, so we simply add it to the plan. We extend the plan by adding a at the end of the plan and we also progress using this progress function that we have defined, where you add the positive effects of an action and delete the negative effects of the action.

And we go to this new state and we were calling this new state S in the style of programming. So, if it is not an action, it could be a compound goal G . So, if the compound goal is true then

you do not have to do anything because it simply states that it is there in the current state and nothing else need to be done.

But if it is not true, even if one of the sub goals in that is not true then or if or even if one of the individual goals that is not true, we push the entire compound goal onto the stack using push set; remember we said we will put first the compound goal onto the stack and then the individual goals. The third thing can happen is that, what you have popped out is the individual goals which we have represented as small g . Again, if it is true in the state which means g belong to S , you do not have to do anything.

But, if it is not true in the state then you have to choose a relevant action a that achieved g ok. So in this algorithm here, I have written the word choose and by this I mean, that somehow you make a non deterministic choice as to which of the two relevant actions to pick. So, for example, if the goal that you want to achieve is holding, let us say block b , then holding b could be in this world that we have defined achieved either by picking a b from the table or unstacking b from something else.

And here, I am simply saying that you somehow know what is the right choice to make. Maybe you can look at the given state and decide that or do some other form of reasoning or use some heuristics.

It may not be applicable always in the given state. As we will see in an example. But, I have written this in a non deterministic fashion, because I did not want to write the backtracking part of the algorithm. Otherwise, you would choose, let us say action a_1 and try out something, if it fails then you come back and try action a_2 and so on.

And that can be implemented using some sort of a backtracking algorithms. But we will not we will skip that details here; we will assume that somehow we have chosen the right action. If there is no right action if there is no action to address the goal g , then of course you cannot find the plan and you return failure. Otherwise, you push that action onto the stack and then you also push the preconditions of those actions onto the stack.

And, so remember that first you push the action, then you put the preconditions. So, first the preconditions would be popped out, and if they are true only then the action would be popped out and executed. That will make sure that the action that is popped out will be applicable. And that is all in the algorithm you just keep repeating the cycle you keep pushing things, either a goal, compound goal along with its individual goals or an action and keep popping them as they proceed.

So this is the basic algorithm for goal stack planning, and we will come back and look at an example and see when it works and when it is not so good in the next session.

So, see you in the next session.