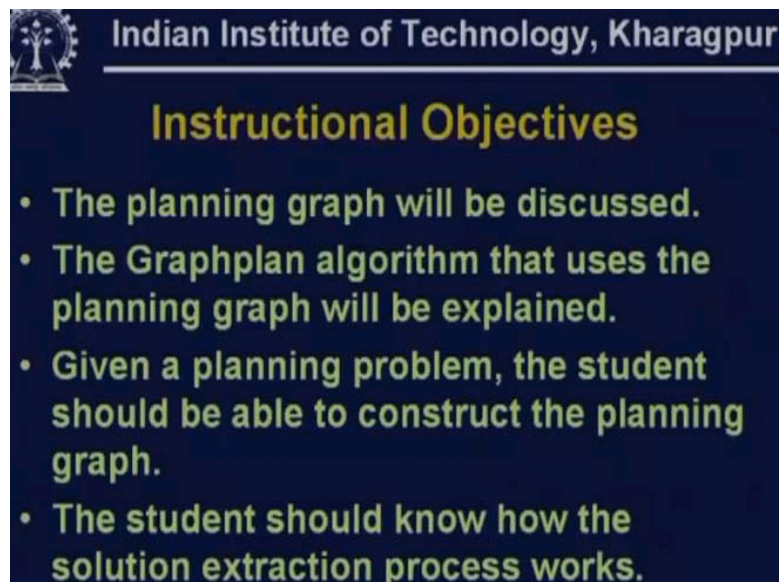


Artificial Intelligence
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture # 24
Planning - 4

Welcome, today we will be having our last lecture on planning. We already had taken three classes on planning and we have looked at plan formulation and several algorithms for planning problems. We have looked at STRIPS, we have looked at forward planning as search problem, we have looked at backward planning or regression and in the last class we had looked at partial order planning. Today we will introduce another way of looking at planning problems and we will introduce the concept of planning graphs and the algorithm graph planning.

(Refer Slide Time: 01:44)



Indian Institute of Technology, Kharagpur

Instructional Objectives

- The planning graph will be discussed.
- The Graphplan algorithm that uses the planning graph will be explained.
- Given a planning problem, the student should be able to construct the planning graph.
- The student should know how the solution extraction process works.

The instructional objective of today's lecture is:

We will discuss the data structure of a planning graph. We will discuss the graph plan algorithm that uses the planning graph and describes an algorithm for finding a plan. Given a planning problem you should be able to construct the corresponding planning graph and starting from the last state you should know how to extract a solution plan from the planning graph. As you have noted a big source of inefficiency in search algorithms is because of the branching factor.

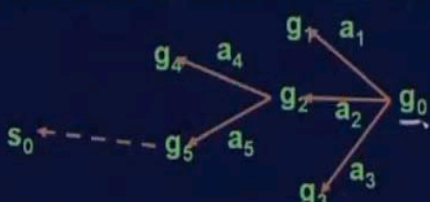
If you are talking about either forward search or backward search what makes search algorithms expensive is the branching factor. If the branching was one at every node then the algorithm would be a linear algorithm. But otherwise if the average branching factor is B and the depth of a search tree is N you know that B^N nodes have to be expanded. So, to make a search problem more effective what we would like to do is to limit the branching factor or limit the number of children of each node.

(Refer Slide Time: 03:33)

Indian Institute of Technology, Kharagpur

Motivation

- A big source of inefficiency in search algorithms is the *branching factor*
 - the number of children of each node
- e.g., a backward search may try lots of actions that can't be reached from the initial state



When you look at backward planning you start from the goal and you find all the actions that could have produced the goal state. And then from each of the previous nodes you do the same thing. The main problem is that you would be considering a lot of actions. And many of these actions may not be reachable, we may not be able to take the action starting from the initial state. So our objective is to find out a subset of the actions which can be activated given the initial state that the agents start from. And in this way we will like to limit the branching factor. This was one of the motivations for this algorithm.

(Refer Slide Time: 04:44)

Indian Institute of Technology, Kharagpur

Motivation

One way to reduce branching factor:

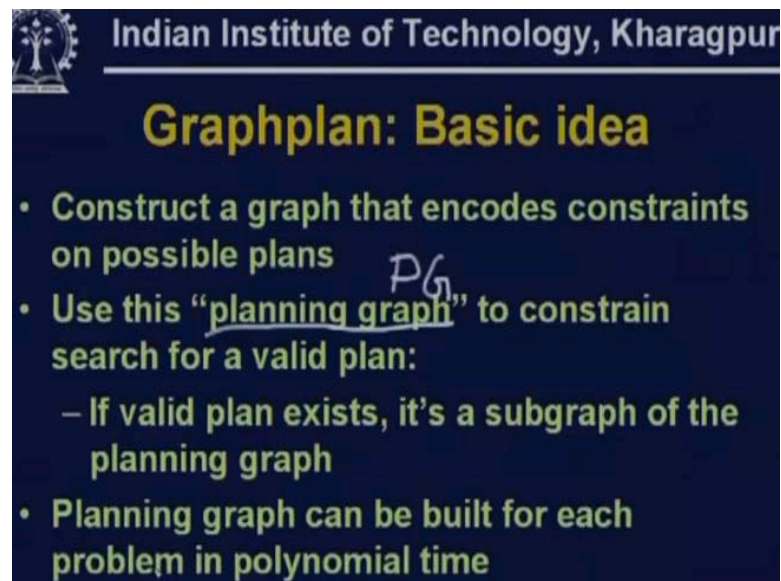
- First create a relaxed problem
 - Remove some restrictions of the original problem
 - The relaxed problem should be easy to solve
 - The solutions to the relaxed problem will include all solutions to the original problem
- Then do a modified version of the original search
 - Restrict its search space to include only those actions that occur in solutions to the relaxed problem

One way to reduce the branching factor is as follows:

First you consider a relaxed problem. So, given the original problem you take a relaxed version of the problem where you let go of some of the constraints. That is, you remove some restrictions of the original problem. And the relaxed problem that you

get should be easy to solve. And the solution to this relax problem will include the solution to the original problem. So once you get this relax problem you can do a modified version of the original search. And you restrict the search space to only those actions that are applicable in the relax problem. Now the basic idea of graph plan is you construct a graph that encodes some constraints on the possible plans. You take some of the constraints from possible plans and incorporate them to create a graph. Now you will use this planning graph which we will refer to subsequently as PG. We will take the planning graph PG to constrain the search for a valid plan.

(Refer Slide Time: 05:55)



Indian Institute of Technology, Kharagpur

Graphplan: Basic idea

- Construct a graph that encodes constraints on possible plans
- Use this "planning graph"^{PG} to constrain search for a valid plan:
 - If valid plan exists, it's a subgraph of the planning graph
- Planning graph can be built for each problem in polynomial time

The way we will construct the planning graph or PG is as follows:

If any valid plan exists it must be a sub-graph of this PG. The partial order plan that we have looked at can be expressed in the form of a directed acyclic graph or a diagram. And we will construct a planning graph such that a valid plan will be a subset of this PG. Now we will show that the planning graph can be built in polynomial time. And once we build the planning graph we are able to obtain restrictions on actions which will make a search for a valid plan easily.

Now the planning graph we will consider will be a directed graph and it will have several levels and in this graph we will have two types of nodes namely proposition nodes and action nodes. Therefore in the planning graph we have proposition nodes and then we have action nodes. And in fact in the graph we will have several levels and the level with the proposition nodes will alternate with a level with the actions. And we will also have three types of edges between levels. There will be the precondition edges from propositions to actions, this proposition is a precondition for this action or these two propositions are the preconditions for this action, these are called precondition edges. We have add edges from action to proposition, if this action is executed this proposition will be added. Then we have delete edges, if this action is executed certain propositions will be deleted. So this may be q and this may be r , this may be $0r$.

(Refer Slide Time: 08:18)

Indian Institute of Technology, Kharagpur

Planning graph

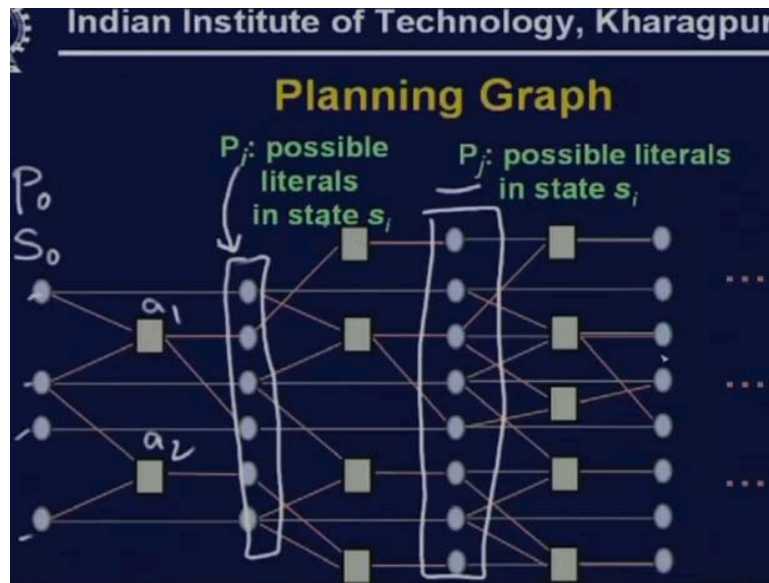
- Directed, leveled graph
 - 2 types of nodes:
 - Proposition: P
 - Action: A
 - 3 types of edges (between levels)
 - Precondition: $P \rightarrow A$
 - Add: $A \rightarrow P$
 - Delete: $A \rightarrow P$
- Proposition and action levels alternate
- Action level includes actions whose preconditions are satisfied in previous level plus no-op actions (to solve frame problem).

So we may have add edges and delete edges from action nodes to proposition nodes and we will have precondition edges from proposition nodes to action nodes. And in the planning graph the proposition levels and action levels will alternate. The action level will include actions whose preconditions are satisfied in the previous level plus maintenance actions. The propositions at a particular level will denote those propositions that can be valid at a particular state.

In a state where the subset of these propositions will be valid what are the actions that one can take from the state?

The actions that one can take from the state are those for which the preconditions are satisfied. Also, certain propositions will get carried over to the next state by virtue of the frame axioms as we have discussed earlier. As we have noted that there can be many propositions in the world and unless some proposition is explicitly added or deleted the other propositions which hold in a particular state continue to hold at the next state. So, corresponding to these propositions we will have maintenance actions or no-op actions. Let us look at an example of the planning graph and then look at the algorithm graph plan.

(Refer Slide Time: 11:33)

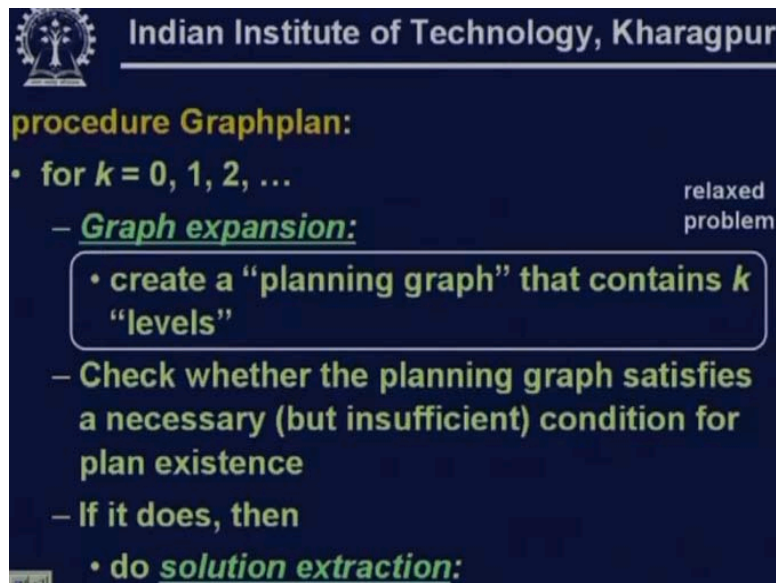


This is an example of a planning graph. This is the initial state or S_0 , these are the propositions which hold at the initial state, these are some actions whose preconditions are satisfied in this initial state and this is the set of propositions that can hold in the next state. So let P_0 be the set of propositions that can hold in state S_0 and let P_i be the possible set of propositions that can hold in state S_i . This set is called P_i . P_i is the possible literals or propositions that can hold in state S_i . This represents P_j and P_j represents possible set of literals that can hold in state S_j and a_i are the set of actions that can be taken at state S_i . This is an example of the planning graph.

These pink edges from a proposition to action means this is a precondition edge and edges from action to propositions mean they are add or delete edges. They are add edges if they go to a positive literal and they are delete edges if they go to a negative literal and this grey edges are the maintenance actions or the no-op actions. So this proposition continues to hold at the next state if the action does not add or delete that. This is an example of a planning graph.

We have come to the procedure graph plan now. In the algorithm for graph plan there will be two phases. There will be one phase for graph expansion and there will be another phase for solution extraction. We start with k is equal to 0 then k is equal to 1 and then k is equal to 2 and so on. For every value of k we first do graph expansion then we check certain conditions in the graph and if the graph needs those conditions we do solution extraction.

(Refer Slide Time: 13:18)



Indian Institute of Technology, Kharagpur

procedure Graphplan:

- for $k = 0, 1, 2, \dots$
 - **Graph expansion:**
 - create a “planning graph” that contains k “levels”
 - Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence
 - If it does, then
 - do **solution extraction:**

relaxed problem

Procedure for graph plan:

For a particular value of k in the graph expansion phase we create a planning graph that contains k levels. This represents a relaxed problem. This planning graph we get incorporates some of the constraints of the problem we have but not necessarily all the constraints so this planning graph is a solution to the relaxed problem. And the actual plan will be a sub-graph of this planning graph.

Once we have created the planning graph up to k levels we check whether the planning graph satisfies a necessary condition for plan existence. Once we have a graph we check first by looking at the last level of the plan whether this plan would satisfy a necessary condition for the final plan. If it does satisfy then we try to extract a solution.

If we succeed in extracting a solution we output the solution. If we do not succeed in extracting a solution from the level k graph we increment the value of k and then go through the graph expansion phase again. Therefore, in the solution extraction phase what we do is we consider only the planning graph that we have and we do a backward search from the goal state. So, in this backward search in the planning graph we consider only those actions that are present in the planning graph and this is what constraints the search space. Now as a result of solution extraction by backward search through the planning graph if we find a solution then we return the solution.

And as we have seen earlier this is an example of the planning graph. These pink edges denote the precondition edges and the ‘add and delete’ edges, the grey edges denote the maintenance edges or the edges corresponding to no-op actions. And P_i is the possible literals in state I , P_j is the possible literals in state j so this is the notation that we could be using.

(Refer Slide Time: 16:18)

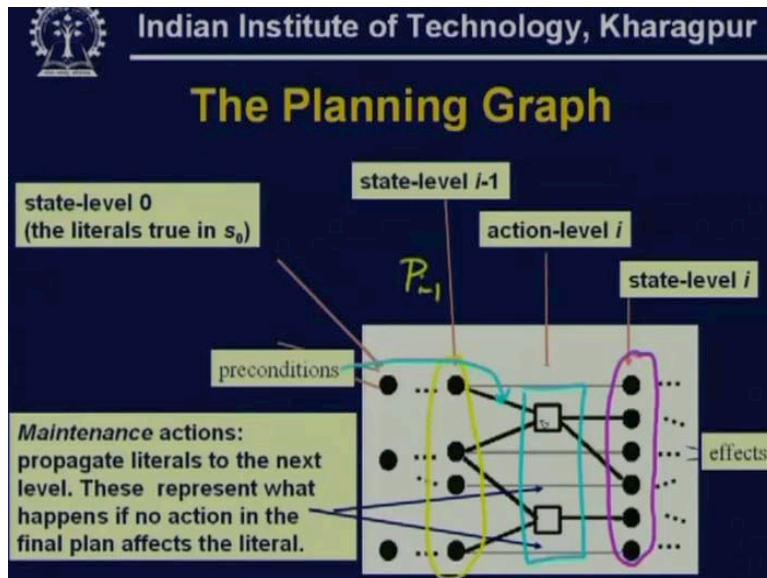
Indian Institute of Technology, Kharagpur

The Planning Graph

- Alternating layers of ground literals and actions
 - All actions that might possibly occur at each time step
 - All of the literals asserted by those actions

Therefore in a planning graph there are alternating layers of ground literals and actions. So at the action level i we have all actions that might possibly occur at that time step and at P_i we have all the literals that are asserted by these actions. So, in the planning graph suppose this is the state level i minus 1 which consists of the propositions P_{i-1} and suppose we have the state level i consisting of the literals P_i and this is the action level i , now this is state level 0, these are the propositions which hold in the initial state which are known to us and these are precondition edges.

(Refer Slide Time: 17:11)



So this is an action, this action has this and this has the precondition, this action has this and this has the precondition and these are the effect edges. So these three are the effect edges and these are the precondition edges and these grey lines correspond to maintenance actions. That is, this proposition is carried over to the next state if an action does not affect that proposition.

So how do we construct the planning graph?

At level P_1 we have all the literals which are given in the initial state. In a planning problem you are given all the propositions that hold in the initial state and you are given a partial description of the goal state and you are required to find a plan which achieves the goal state.

(Refer Slide Time: 18:55)

Indian Institute of Technology, Kharagpur

Constructing the planning graph

- Level P_1 : all literals from the initial state
- Add an action in level A_i if all its preconditions are present in level P_i
- Add a precondition in level P_i if it is the effect of some action in level A_{i-1} (including maintenance actions or no-ops)
- Maintain a set of exclusion relations to eliminate incompatible propositions and actions (thus reducing the graph size)

$P_1 A_1 P_2 A_2 \dots P_{n-1} A_{n-1} P_n$

Therefore P_1 will include all literals in the initial state. Then we will add an action in level A_i . Suppose we have P_i we will see how to construct A_i . We will add an action in level A_i if all the preconditions of that action are present at level P_i . We will add a precondition in level P_i if it is the effect of some action in level A_i minus 1. So a proposition will be added in level P_i if it is the effect of some action in the level A_i minus 1. And because we also have these maintenance actions we will carry over the propositions from P_i minus 1 to P_i using maintenance actions and due to other actions we have some other propositions as effects which we will add to level i .

In addition, we will maintain a set of exclusion relations called mutex relations. So these mutex relations will be used to eliminate incompatible propositions and actions. And because of these mutex relations we will be able to reduce the graph size **work**. So the graph consists of alternate levels. We have a proposition level P_1 followed by an action level A_1 then a proposition level P_2 followed by an action level A_2 and so on. This is a planning graph.

Now what would be the mutual exclusion relations we will consider?

We say two actions are mutex or two propositions are mutex. So we have mutex relations between two propositions and we also have mutex relations between two actions. Now two actions are mutex at some stage if no valid plan could contain both. So if at a stage we find that both the actions cannot be together present in a valid plan we say that those two actions are mutex.

Similarly, at a particular level if we find that two propositions cannot occur in a valid plan we will call them to be mutex. Now we will look at certain conditions under

which we can label two actions as mutex and then we will also find conditions under which we can label two propositions as mutex. So we will study three different mutex situations between two actions.

(Refer Slide Time: 22:11)

Indian Institute of Technology, Kharagpur

Mutual Exclusion relations

- Two actions (or literals) are mutually exclusive (mutex) at some stage if no valid plan could contain both.
- Two actions are mutex if:
 - ① Inconsistent effects: an effect of one negates an effect of the other
 - Interference
 - Competing needs

Two actions are mutex. We will consider inconsistent effects, interference and competing needs. First we will look at what we mean by inconsistent effects. So two actions a_1 and a_2 are mutex they cannot occur together. If an effect of one negates an effect of the other. Suppose the effect of a_1 is P and the effect of a_2 is $\neg P$ these two actions cannot come at this level of any valid plan. This is graphically illustrated here.

(Refer Slide Time: 22:18)

Indian Institute of Technology, Kharagpur

Mutual Exclusion relations

Inconsistent Effects

We have a proposition level P_i and here we have the proposition level P_i plus 1 and these are the actions. We say that action a_1 and action a_2 are mutex which we denote by this line between them a_1 and a_2 are mutex. If the effect of a_1 is inconsistent with

the effect of a_2 this could happen. Suppose q is the effect of a_1 and $\text{NOT}q$ is the effect of a_2 and since q and q_2 are mutex they cannot occur together then a_1 and a_2 will be mutex. Therefore this condition is called inconsistent effect condition. Two actions will be mutex if their effects are inconsistent.

(Refer Slide Time: 23:25)



Indian Institute of Technology, Kharagpur

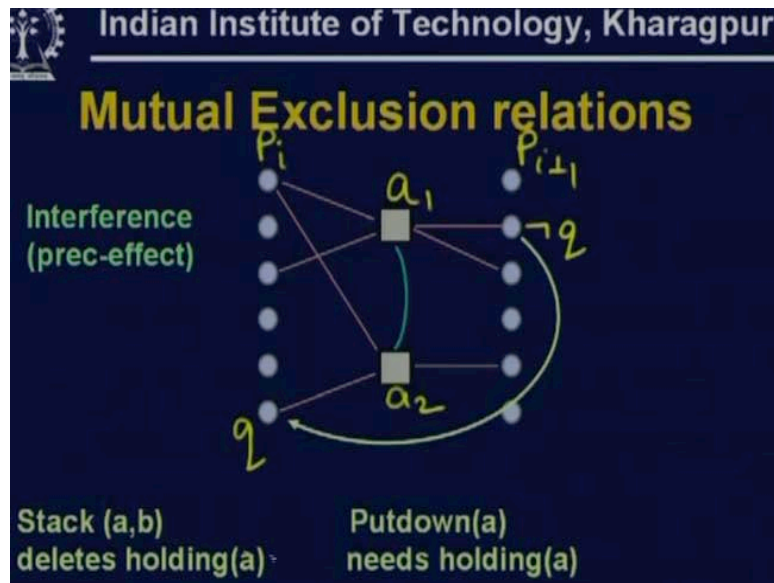
Mutual Exclusion relations

- Two actions are mutex if:
 - *Inconsistent effects*
 - **Interference**: one deletes a precondition of the other
 - *Competing needs*

Secondly, we will say that two actions are mutex under the interference condition. That is, one of the actions deletes a precondition of the other action. So this illustrates interference so we have these propositions at level P_i and this is P_i plus 1 and we say action a_1 and a_2 are mutex. If action a_1 has an effect $\text{NOT}q$ that is action a_1 deletes q where q is a precondition of a_2 so in this case we say a_1 and a_2 interfere. Therefore these two actions will be mutex. An example is the two actions Stack (a, b) and putdown (a). Now putdown (a) has holding (a) as precondition but Stack (a, b) is an action that deletes holding (a). So, Stack (a, b) deletes holding (a) and putdown (a) needs holding (a). Hence we cannot have these two actions together so they are mutex.

Thirdly, two actions are mutex under the condition of competing needs. That is, if these actions have mutually exclusive preconditions, if the preconditions cannot occur together then at the next stage we cannot have those two actions together. This is illustrated by the following diagram:

(Refer Slide Time: 24:44)



Suppose we have these two actions a_1 and a_2 and suppose this is the precondition of a_1 and this is the precondition of a_2 we say that if these two preconditions of a_1 and a_2 are mutex then these two actions must also be mutex. For example, consider the action stack (a, b) and unstack (a, b), stack (a, b) requires that b must be clear and unstack (a, b) requires that a must be clear. So if these two cannot be clear at the same time then these two actions cannot be executed in parallel. Therefore under the condition of competing needs also these two actions are mutex. So we have three conditions under which actions are mutex. Two actions are mutex if they have inconsistent effects, if there is interference or if there are competing needs.

Now let us see when two propositions are said to be mutex. These three conditions of action mutex can be found easily while constructing the planning graph. So we can label these inconsistencies easily. It is also important to note here that these are only some of the mutex conditions. But these mutex conditions are easy to find and they put some restrictions on the planning graph which goes towards reducing the amount of search that we need to carry out.

(Refer Slide Time: 27:44)

Indian Institute of Technology, Kharagpur

Mutual Exclusion relations

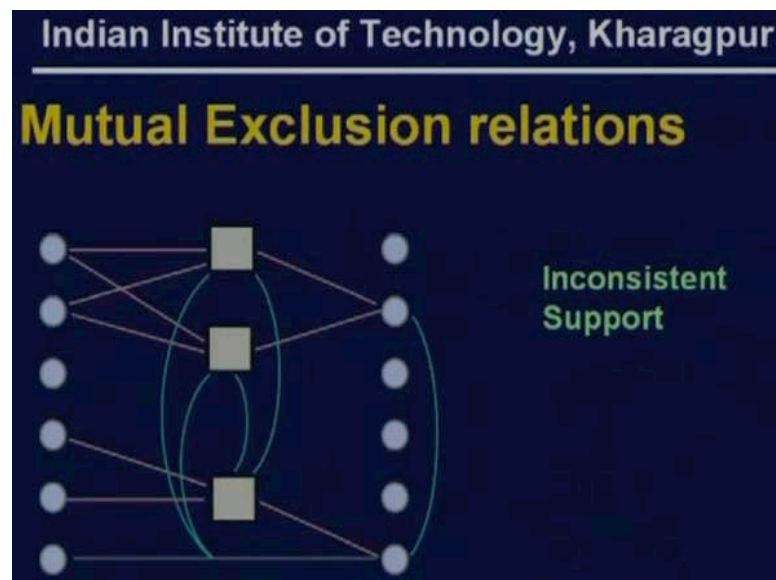
- Two propositions are mutex if:
 - All ways of achieving them are mutex

One is the negation of the other
OR
All ways of achieving them are pairwise mutex

Now two propositions are said to be mutex if all ways of achieving the propositions are mutex. Now when would we say this?

For example, suppose one proposition is the negation of the other so P and $\neg P$ are mutex because both of them cannot be simultaneously true at a state. Two propositions are also mutex if all ways of achieving them are pairwise mutex. This is illustrated by the following diagram:

(Refer Slide Time: 28:11)



We have these two propositions that we are considering, we say that they are mutex. These propositions can be achieved by action a_1 or action a_2 . This proposition can be achieved by action a_3 or by this maintenance action. Now a_1 is mutex with a_3 , a_2 is mutex with a_3 , a_1 is mutex with this maintenance action, a_2 is also mutex with this maintenance action so these two propositions can not be true simultaneously because you cannot achieve both of them simultaneously because all ways of achieving them

are mutually mutex. Hence there are two ways of achieving this proposition, there are two ways of achieving this proposition and they are pair wise mutex so each of these two ways is mutually exclusive with each of these two other ways so this is a case of inconsistent support. Therefore these two propositions will be mutex. Now let us look at an example to illustrate the construction of the planning graph and then subsequently we will use the same example to run the graph plan algorithm.

(Refer Slide Time: 30:00)

Indian Institute of Technology, Kharagpur

Dinner Date Example (by Dana Nau)

Suppose you want to prepare dinner as a surprise for your sweetheart (who is asleep)

$s_0 = \{\text{garbage, cleanHands, quiet}\}$
 $g = \{\text{dinner, present, -garbage}\}$

Now this problem is proposed by Dana Nau. Suppose you want to prepare dinner as a surprise for your sweetheart the initial state is as follows:

There is garbage, hands are clean and is quiet. So garbage, clean hand, quiet is the characteristic of the initial state and your goal is to prepare dinner have a present and there should be no garbage. So, dinner and present and no garbage is the goal condition.

(Refer Slide Time: 30:44)

Indian Institute of Technology, Kharagpur

Dinner Date example

- **Initial Conditions:** (and (garbage) (cleanHands) (quiet))
- **Goal:** (and (dinner) (present) (not (garbage)))
- **Actions:**
 - Cook :precondition (cleanHands)
:effect (dinner)
 - Wrap :precondition (quiet)
:effect (present)
 - Carry :precondition
:effect (and (not (garbage)) (not (cleanHands)))
 - Dolly :precondition
:effect (and (not (garbage)) (not (quiet)))

These are the actions which are available to you:

Cook, wrap, carry and dolly.

Cook is an action whose precondition is clean hands and effect is dinner.


Wrap is an action its precondition is quiet and effect is present.

Carry is an action which has no precondition and its effect is no garbage and not clean hands. So hands become dirty but there is no garbage.

Dolly is an action which has no precondition its effect is no garbage and not quiet.

We have these actions; cook, wrap, carry and dolly and with these actions we have to see how we can achieve this goal of dinner and present and no garbage. So this table has the same thing such as this action, precondition and the effect.

(Refer Slide Time: 31:55)



Indian Institute of Technology, Kharagpur

Example

Action	Preconditions	Effects
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	none	\neg garbage, \neg cleanHands
dolly()	none	\neg garbage, \neg quiet

Also have the maintenance actions: one for each literal

Cook has clean hands as precondition and dinner has effect, wrap has quiet as precondition and present as effect, carry has no precondition no garbage and no clean hands is the effect, dolly has no preconditions no garbage not quiet is the effect. In addition to these four actions we also have all the maintenance actions. It is one for each literal. Let us see what it signifies.

In the initial state we have garbage and then clean hands, quiet. And you also know whatever propositions are not mentioned in the initial state are assumed to be absent. So, in the initial state there is no dinner no present.

(Refer Slide Time: 32:25)

Indian Institute of Technology, Kharagpur

Example

- *carry* is mutex with the maintenance action for *garbage* (inconsistent effects)
- *dolly* is mutex with *wrap*
 - interference
- \neg quiet is mutex with *present*
 - inconsistent support
- each of *cook* and *wrap* is mutex with a maintenance operation

What are the actions that one can carry out in initial state?

One can do carry, one can do dolly, carry and dolly have no preconditions. One can do cook the precondition is clean hands, one can do wrap the precondition is quiet. And the effect of cook is dinner, effect of wrap is present, effect of carry is no garbage and no clean hands, effect of dolly is no garbage and not quiet. Now let us see what sort of mutex hold between the actions. In addition to these four actions we also have the maintenance conditions one for each literal.

For example, garbage to garbage there is a maintenance action. From clean hands to clean hands there is a maintenance action. Quiet to quiet similarly no dinner to no dinner, no present to no present are the maintenance actions. Now we notice that carry has no garbage as effect and the maintenance action corresponding to garbage has garbage as the effect. Since garbage and no garbage are mutex they cannot occur together so carry is mutex with this maintenance action. Therefore carry and this maintenance action are mutex.

Similarly, dolly has no garbage as the effect. This no op has garbage as the effect so dolly is also mutex with the maintenance action. So dolly has an effect of not quiet and quiet has the effect of quiet. Since not quiet and quiet are mutex dolly cannot occur along with the maintenance action quiet. Hence dolly and quiet are mutex. Dolly is mutex with wrap because wrap requires quiet and dolly has an effect not quiet, this is an example of interference. Dolly has effect not quiet wrap requires quiet so dolly and wrap cannot occur together.

Not quiet is inconsistent with present, cook is inconsistent with not present maintenance action, wrap is inconsistent with not present maintenance action. So we start with the initial state garbage, clean hands, quiet, no dinner, no present. And these are the four possible actions carry, dolly, cook, wrap and these are the maintenance actions. And then we find all the mutex relations between these actions according to the three rules that we have looked at. And then carry is mutex with this maintenance action for garbage, dolly is mutex with the maintenance action from garbage, dolly is

mutex with wrap, not quiet is mutex with present, cook is mutex with not dinner, wrap is mutex with not present. So at state level 0 we have all the propositions of the atoms that are mentioned in initial state.

(Refer Slide Time: 37:18)

Indian Institute of Technology, Kharagpur

Example (continued)

- state-level 0:
 $\{ \text{all atoms in } s_0 \} \cup \{ \text{negations of all atoms not in } s_0 \}$
- action-level 1:
 $\{ \text{all actions whose preconditions are satisfied in } s_0 \}$
- state-level 1:
 $\{ \text{all effects of all of the actions in action-level 1} \}$

Union {the negations of all atoms which are not in S_0 }

At action level 1 we have all actions whose preconditions are satisfied in S_0 .

In state level 1 we have all effects of all the actions in action level 1. This is how we can construct the planning graph. And by looking at the mutex conditions we can specify the mutual exclusive relations between them.

(Refer Slide Time: 37:44)

Indian Institute of Technology, Kharagpur

Action	Precond	Effects
cook()	cleanH	dinner
wrap()	quiet	present
carry()	none	\neg garb, \neg cleanH
dolly()	none	\neg garb, \neg quiet

Here I have not put the maintenance actions for not dinner and not present but the rest we have is this two level planning graph for this problem. So the actions as we have noted are carry, dolly, cook and wrap. These are the precondition links and these are

the effect links. These are the maintenance actions and then we have specified the mutex relations between the different actions.

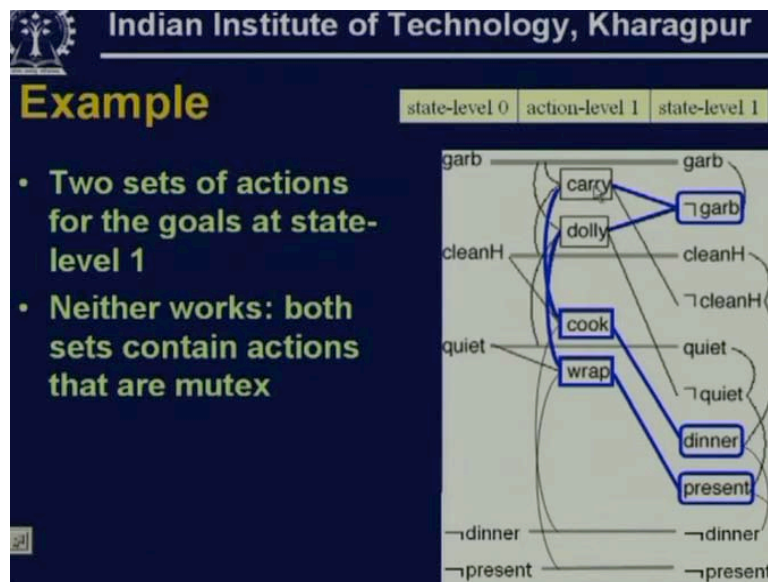
We already saw how to have the mutex relation between the different actions. And then we can also have mutex between the different propositions. Therefore this is easy garbage mutex with no garb, clean hands is mutex with no clean hands, quiet is mutex with not quiet then dinner is mutex with not clean hands because dinner requires clean hands and dinner is mutex with not clean hands. Present is mutex with not quiet because present requires quiet. So these are the mutex relations between the different propositions.

Now, once we have this planning graph where we have state level 0, action level 1 and state level 1 now we need to check to see whether there is a possible plan of length 1 through this planning graph. So the goal is not garbage, dinner and present. All these three propositions are present in the goal and we want to see if a plan exists. Now, not garbage, dinner, present all are present in state level one and they are not mutex with each other so this satisfies a necessary condition that a plan can exist. Now we have to do plan extraction to find a solution.

How do we find a solution?

Not garb requires either carry or dolly, dinner requires cook, present requires wrap. So, for present we have to take wrap, for dinner we have to take cook and for not garbage we have to take either carry or dolly. Carry is mutex with cook so we cannot take carry then we have to consider dolly. Dolly is mutex with wrap so we cannot take the action dolly. So in none of these ways we can satisfy the not garb action.

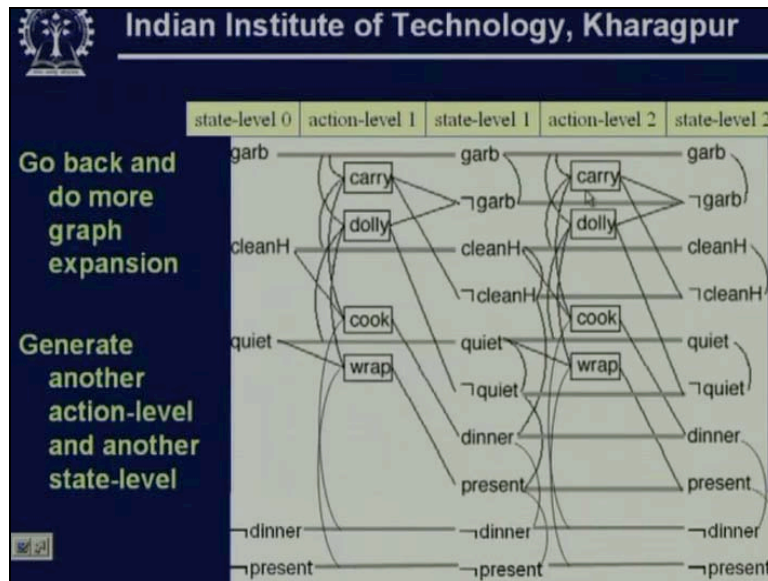
(Refer Slide Time: 41:18)



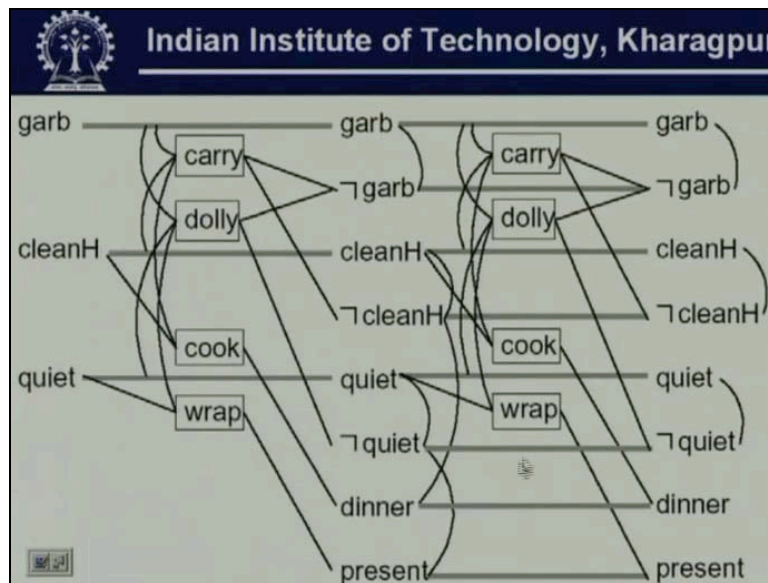
Therefore this plan cannot be extracted starting from this planning graph because not grab can be achieved in two ways carry and dolly and carry is mutex with cook. Cook is required for dinner, present can only be achieved by wrap and wrap is mutex with dolly. So neither carry nor dolly can be executed which is consistent with these two.

So neither of this work and so we have to abandon this plan so this plan does not exist.

(Refer Slide Time: 41:38)



(Refer Slide Time: 42:33)

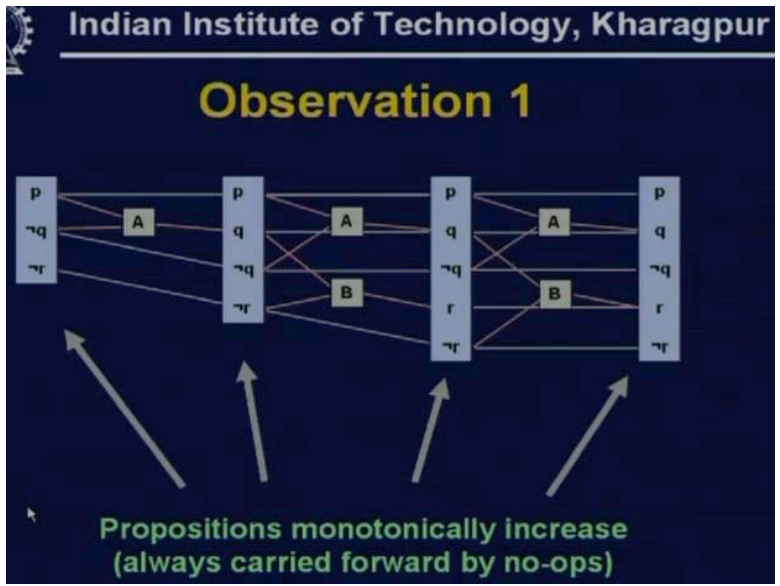


Now what you do if this plan does not exist?

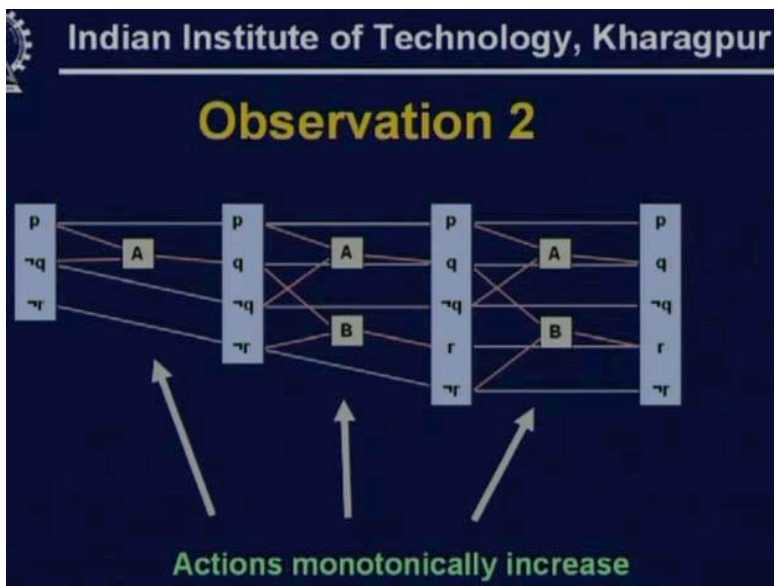
Now you have to consider expanding your planning graph further. So you have to go back and expand the graph further to one more level. So we generate one more action level. We had already generated up to this, we generate one more action level action level two and then state level two. And then we again try to see if state level two satisfies the necessary conditions for plan existence and then we try to extract the plan from state level two. This is the planning graph expanded up to state level two. Here we see that the carry, dolly, cook and wrap actions are there and we have found all the mutex relations between the four actions and we have found the mutex relations

between the propositions. Before we proceed further let us look at certain observations.

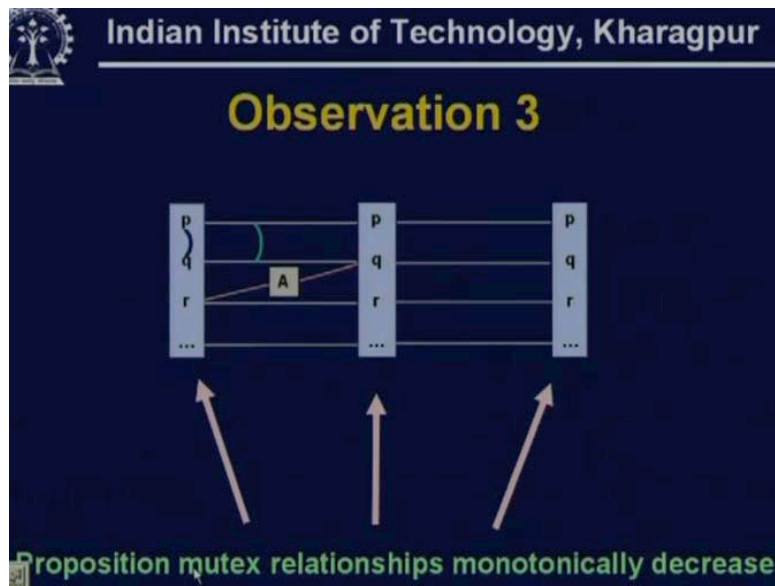
(Refer Slide Time: 42:54)



(Refer Slide Time: 43:00)



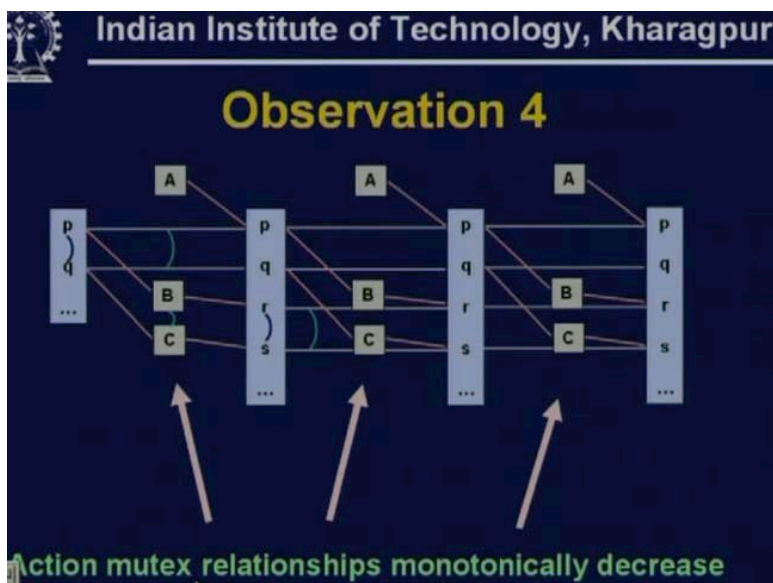
(Refer Slide Time: 43:18)



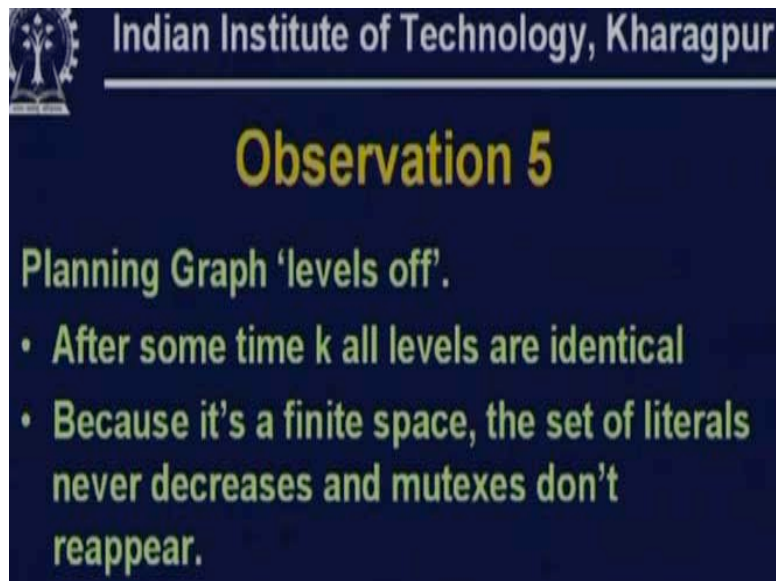
So the first observation is that a set of propositions in state level zero, these are the set of propositions in P_1 , this is P_2 , P_3 and we observe that the set of propositions are always monotonically increasing, the size of the set is increasing. We also note that the proposition mutex relationships are monotonically decreasing. Whatever mutex relationships we had initially as we proceed further these relations can be dropped and no new mutex relations will arise.

Observation two is that since between two levels the number of propositions are increasing the number of actions which are applicable will also increase. Therefore more actions will be applicable here than in this level. Therefore the number of state variables is increasing, the number of propositions is increasing, the number of actions is increasing and then the mutex relationships between the actions are decreasing as we go.

(Refer Slide Time: 44:00)



(Refer Slide Time: 44:11)



Indian Institute of Technology, Kharagpur


Observation 5

Planning Graph 'levels off'.

- After some time k all levels are identical
- Because it's a finite space, the set of literals never decreases and mutexes don't reappear.

From this we can conclude that as we construct the different levels of planning graph after some time the planning graph will level off because we are always increasing the propositions and after sometime we cannot increase any more propositions so the propositions that we had at level i will be the same as the propositions we will have at level i plus 1. Therefore after sometime we will find that all the levels are identical. So, because we have a finite space the set of literals will never decrease and no mutexes will appear. Hence this means, after sometime our planning graph will reach a saturation point so the planning graph cannot go on for ever.

(Refer Slide Time: 45:00)



Indian Institute of Technology, Kharagpur

Valid plan

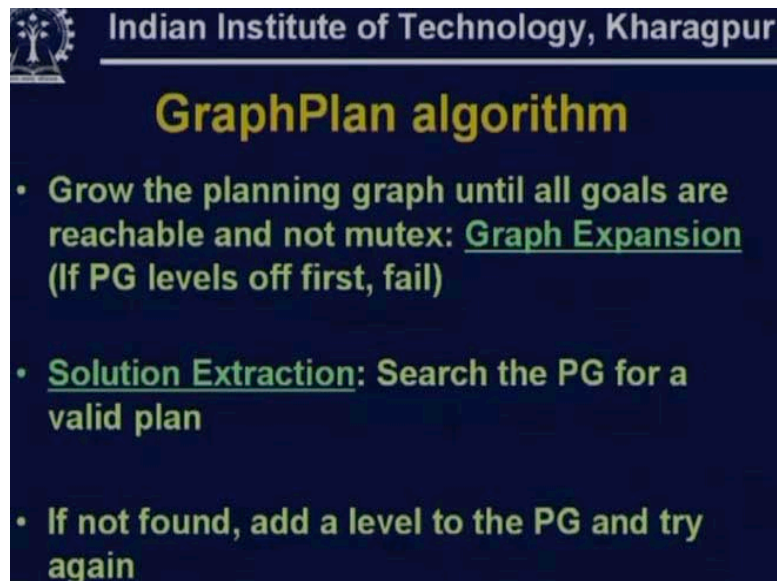
A valid plan is a planning graph where:

- Actions at the same level don't interfere
- Each action's preconditions are made true by the plan
- Goals are satisfied

Now let us see what a valid plan is. A valid plan is a planning graph where actions at the same level do not interfere. Each action's preconditions are made true by the plan and the goals are satisfied. As we have noted a valid plan is a sub-graph of the planning graph we have. And it is that sub-graph where the actions at the same level

do not interfere that is they cannot be mutex. Each action's preconditions are made true by the plan and the goals are satisfied.

(Refer Slide Time: 45:45)



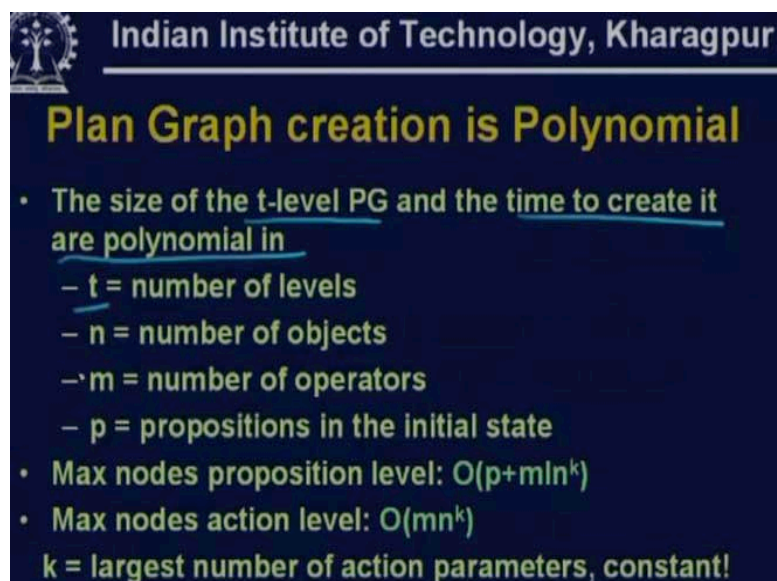
Indian Institute of Technology, Kharagpur

GraphPlan algorithm

- Grow the planning graph until all goals are reachable and not mutex: Graph Expansion (If PG levels off first, fail)
- Solution Extraction: Search the PG for a valid plan
- If not found, add a level to the PG and try again

In the graph plan algorithm as we have noted earlier there are two phases. In the first phase grow the planning graph this is called the graph expansion phase and then we have the solution expansion phase where we search the planning graph for a valid plan. Now if I do not find the valid plan we will add one new level to the planning graph and try again. Before we proceed we note that the creation of the planning graph is a polynomial time algorithm.

(Refer Slide Time: 46:25)



Indian Institute of Technology, Kharagpur

Plan Graph creation is Polynomial

- The size of the t-level PG and the time to create it are polynomial in
 - t = number of levels
 - n = number of objects
 - m = number of operators
 - p = propositions in the initial state
- Max nodes proposition level: $O(p+mln^k)$
- Max nodes action level: $O(mn^k)$

k = largest number of action parameters, constant!

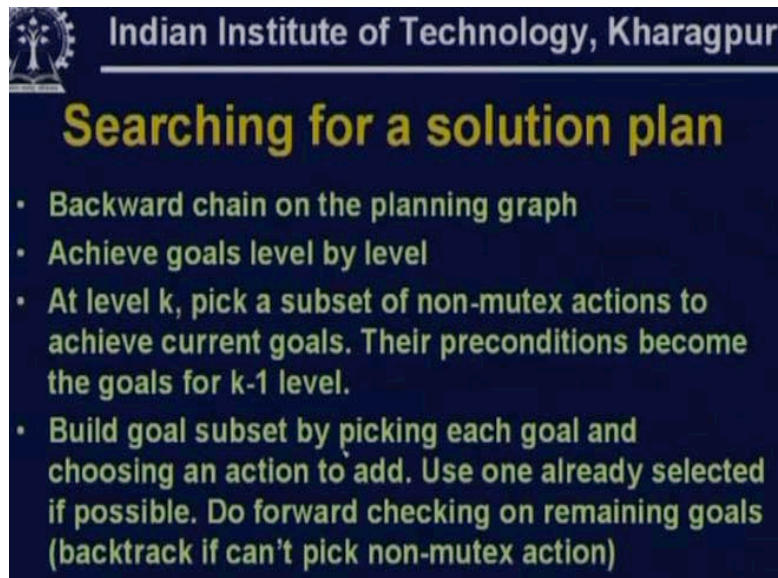
The size of a t level planning graph and the time to create the planning graph are polynomial in the following:

- t is the number of levels

- n is the number of objects
- m is the number of actions and
- p is the number of propositions in the initial state

The size and the time of this planning graph is proportional to p plus m into l into n power k . So, for k constant this is a polynomial. The maximum action nodes are m into n power k where k is the largest number of action parameters we can have and k is usually constant, k is the number of actions in the possible actions planning problem that we are considering. Once we have got the graph let us look at how we do solution extraction. For solution extraction we do a search for a solution plan by backward chaining on the planning graph. We first start at the last level of the planning graph where the goals are present and we try to achieve goals level by level. At level k we pick a subset of non-mutex actions that can achieve the current goals. Their preconditions of these actions become the goals for the k minus 1 at level.

(Refer Slide Time: 47:44)



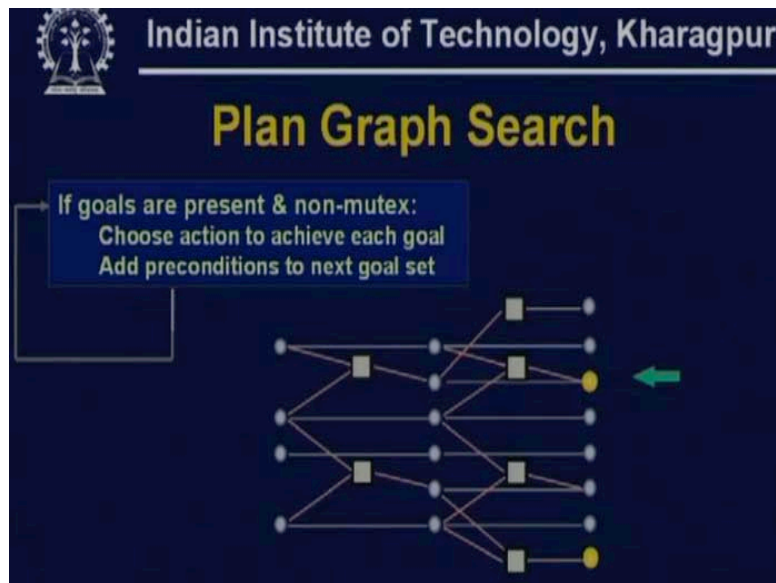
Indian Institute of Technology, Kharagpur

Searching for a solution plan

- Backward chain on the planning graph
- Achieve goals level by level
- At level k , pick a subset of non-mutex actions to achieve current goals. Their preconditions become the goals for $k-1$ level.
- Build goal subset by picking each goal and choosing an action to add. Use one already selected if possible. Do forward checking on remaining goals (backtrack if can't pick non-mutex action)

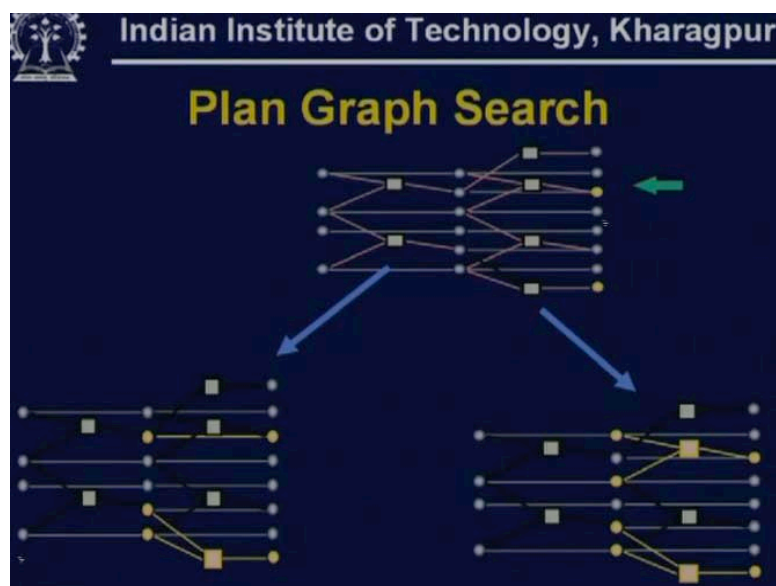
We build the goal subset by picking each goal and choosing one of the actions to add. It could be that one action achieves two different goals. So, if we had a goal g_1 for which we chose action a_1 and if g_3 is also achieved by a_1 then we can achieve g_3 by a_1 . Then we continue on the remaining goals until we find a solution. If we do not find a solution we backtrack and we check the alternate ways of achieving the unachieved goals.

(Refer Slide Time: 48:44)



In the plan graph search we are basically going in a loop. If goals are present at the current level and they are non-mutex with each other we choose action to achieve each goal and add preconditions to the next goal set at the previous level and then we continue recursively until we find a solution. So this is what we execute in the different phases. Therefore this is an example of a planning graph where suppose this and this are the goals these two propositions are the goals that we must satisfy. And as you note, this proposition can be satisfied either by executing this action or by using this maintenance action. This proposition can only be satisfied in one way. So we must take this action it must be selected and out of this action and the no op we have to select one of them. So this is the planning graph we start with and corresponding to these two possibilities we get two different planning graphs.

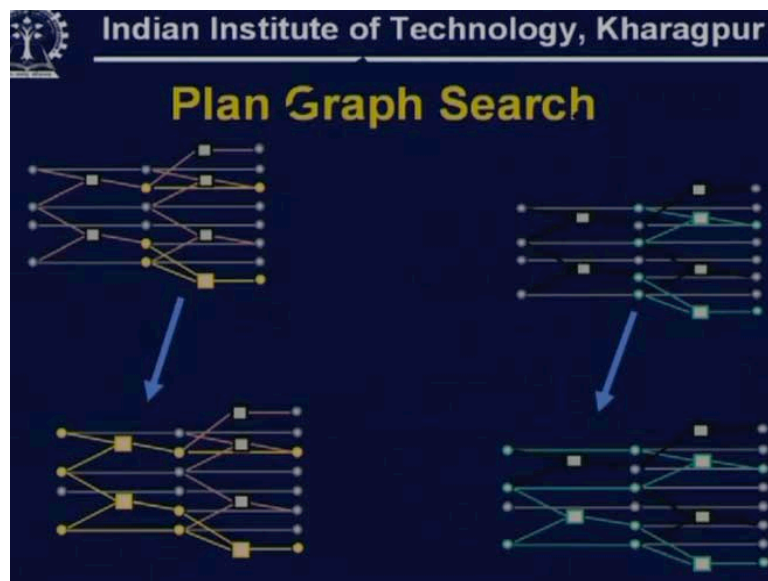
(Refer Slide Time: 50:02)



In this planning graph we have chosen this maintenance action to satisfy this goal. In both the planning graph we choose this action. In this planning graph we choose this action to satisfy. In this side of the search tree we have taken these two actions but here we have taken only this action. Now, in this case at the previous level we will require these two conditions to be satisfied and these two conditions are the preconditions of this action. In this case where we have this maintenance action for this proposition this proposition must have been true at the previous level. So, if we choose this maintenance action then my goals are at the previous level.

On the other hand, if I choose this action to achieve here I have these two actions to satisfy. This action has these two as the preconditions, this action has these two as the preconditions so here I have to satisfy these four preconditions. Therefore this is my goal at the next level. Now, for each of these cases we will again consider these new set of goals and try to see how we can achieve them. By taking the previous example where we had these three preconditions and for this precondition we have only this maintenance action, for this proposition this is the only action that achieves this proposition, for this proposition this is the only action that achieves this proposition.

(Refer Slide Time: 52:40)



So we take these two actions and we find that in order to do this at the previous level these three propositions are required. Therefore we get the goal at the next level and thus we proceed until we come to the zeroth level. If the propositions that we get at the zeroth level are a subset of the initial state description of the system then we have achieved a plan. If we do not achieve a plan we backtrack and we explore other portions of the search tree. Similarly, from the other side we will have some other goal and thus we will proceed. So graph plan proceeds like this.

How long will graph plan proceed?

After sometime graph plan will find that in the graph expansion phase at level k the propositions we have got is the same as what we get at level k plus 1 and that is saturation and we cannot expand the graph any further.

(Refer Slide Time: 53:08)

Indian Institute of Technology, Kharagpur

Termination for unsolvable problems

- Graphplan records sets of unsolvable goals:
 - $U(i,t)$ = unsolvable goals at level i after stage t .
- More efficient: early backtracking
- Also provides necessary and sufficient conditions for termination:
 - Assume plan graph levels off at level n , stage $t > n$
 - If $U(n, t-1) = U(n, t)$ then we know we're in a loop and can terminate safely.

(Refer Slide Time: 53:18)

Indian Institute of Technology, Kharagpur

The set of goals we are trying to achieve

The level of the state s_j

```
procedure Solution-extraction ( $g, j$ )
  if  $j=0$  then return the solution
  for each literal  $l$  in  $g$ 
    nondeterministically choose an action
      to use in state  $s_{j-1}$  to achieve  $l$ 
  if any pair of chosen actions are mutex
    then backtrack
   $g' := \{\text{the preconditions of the chosen actions}\}$ 
  Solution-extraction( $g', j-1$ )
end
```

A real action or a maintenance action

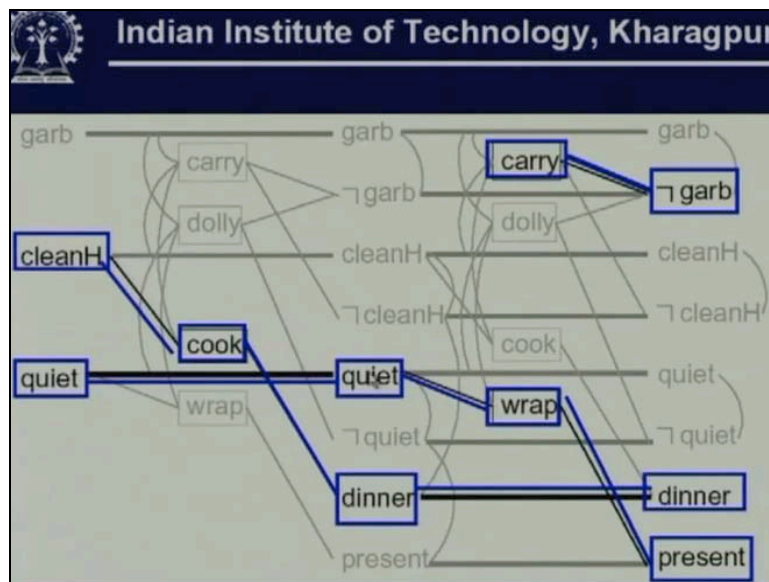
If we have not found the solution then we stop. Also, the graph can record the sets of unsolvable goals which can make it more efficient. For solution extraction, in graph plan g is the set of goals we need to achieve and j is the level of this current state S_j . If j is 0 that is we have reached the initial state we return the solution. Otherwise for each literal l in g for each proposition which occurs in the current goal we nondeterministically choose an action to use in state the previous state S_j minus 1 that achieves l . Once we have chosen actions for each of the literals and the goal if any pair of chosen actions are mutex then we backtrack. Otherwise we get g' which is the set of preconditions of the chosen action and then we recursively call solution extraction on g' and j minus 1. Therefore this is the solution extraction algorithm and this is the search algorithm but this search algorithm is constraint because the planning graph incorporates constraints on the problem.

Now let us come back to the planning graph we obtained. In this planning graph at level two we find that the actions not garb, dinner, present are all present so we try to find the solution. And then in order to find the solution we see that there are three ways to achieve not garbage such as carry, dolly and the maintenance action for not garbage. For dinner there are two ways; the maintenance action and cook, for present there are two ways; the maintenance action presents a wrap.

Now we choose a way of achieving not garb by carry, we choose dinner to be achieved by the maintenance action dinner and we choose present to be achieved by the action wrap. This is one possibility by which we can achieve these goals. In order to do this for wrap we have the precondition quiet, for dinner we need the precondition dinner, for not garb there is no precondition.

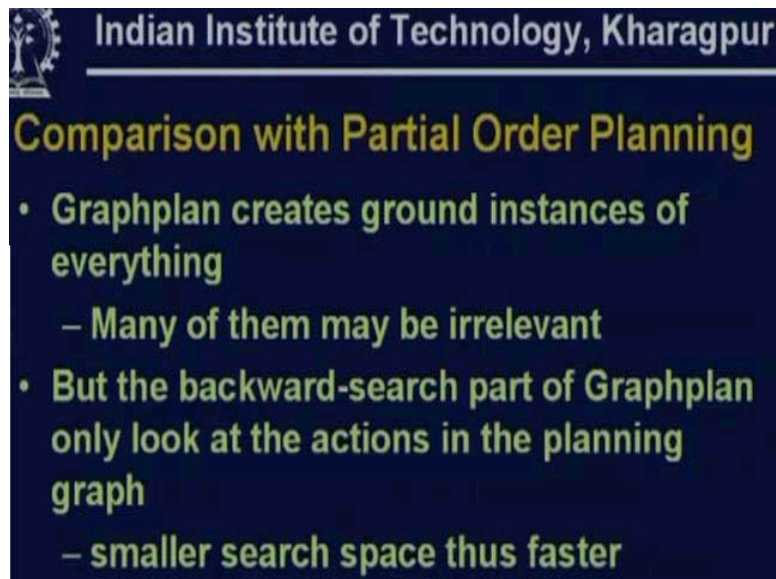
So at the level we have quiet and dinner as our goals.

(Refer Slide Time: 55:25)



To achieve quiet and dinner; quiet is achieved by the maintenance condition quiet and dinner is achieved by the action cook. Cook requires clean hand as a prerequisite and quiet requires quiet as a prerequisite so clean hands and quiet are required in the initial state. Now, clean hands and quiet are satisfied by the initial state description of the problem. Therefore we have found a plan. So this plan which is given in blue is a subset of the planning graph. This is a partial order plan which is a subset of the planning graph.

(Refer Slide Time: 56:44)



Indian Institute of Technology, Kharagpur

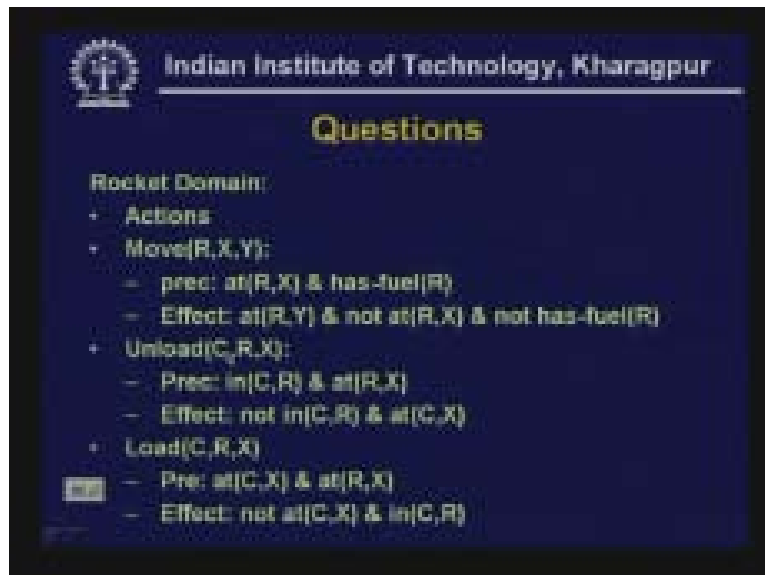
Comparison with Partial Order Planning

- Graphplan creates ground instances of everything
 - Many of them may be irrelevant
- But the backward-search part of Graphplan only look at the actions in the planning graph
 - smaller search space thus faster

Now, how does graph plan compare with partial order planning?

So because of the constraints of the graph plan imposes on the search space it is faster than partial order plan. Here is an exercise to try out. So the domain we will be considering is a rocket domain where we have the following actions. Move R, X, Y. Move rocket R from X to Y. Precondition is at R, X and has fuel R. Effect is at R, Y and not at R, X and not has fuel R. The second action is unload C, R, X which is unload cargo from rocket at X.

(Refer Slide Time: 57:18)



Indian Institute of Technology, Kharagpur

Questions

Rocket Domain:

- + Actions
- + Move(R,X,Y):
 - pred: at(R,X) & has-fuel(R)
 - Effect: at(R,Y) & not at(R,X) & not has-fuel(R)
- + Unload(C,R,X):
 - Pred: in(C,R) & at(R,X)
 - Effect: not in(C,R) & at(C,X)
- + Load(C,R,X)
 - Pre: at(C,X) & at(R,X)
 - Effect: not at(C,X) & in(C,R)

Precondition in C, R and R, X, rocket is at X and cargo is in R.

Effect: not in C, R and at C, X.

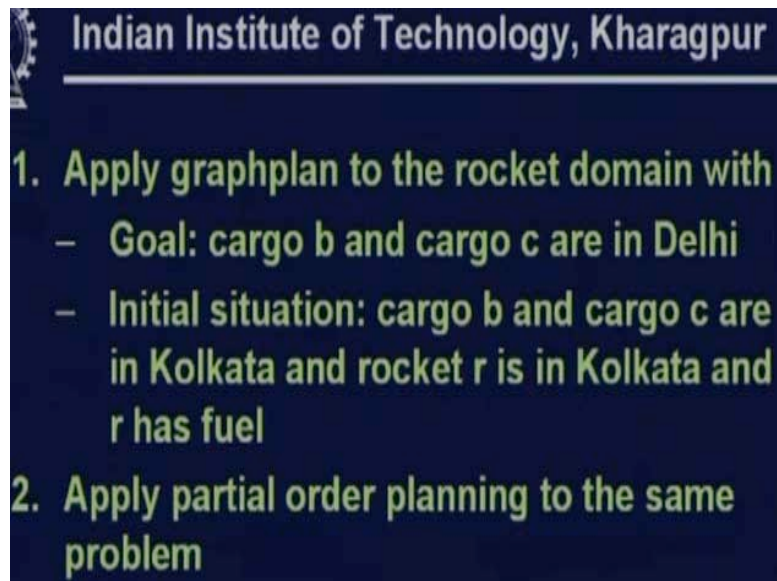
Third action is load C, R, X.

Precondition is at C, X and at R, X.

Effect is not at C, X and in C, R.

So we have three actions; move rocket from X to Y, unload cargo from rocket at X and load cargo from rocket R at X.

(Refer Slide Time: 58:11)



Indian Institute of Technology, Kharagpur

1. Apply graphplan to the rocket domain with
 - Goal: cargo b and cargo c are in Delhi
 - Initial situation: cargo b and cargo c are in Kolkata and rocket r is in Kolkata and r has fuel
2. Apply partial order planning to the same problem

Given this problem you need to apply graph plan to this domain with the cargo b and cargo c are in Delhi this is your goal. Initial situation: cargo b and cargo c are in Kolkata and rocket r is in Kolkata and r has fuel. Therefore for this you have to execute graph plan and on the same problem you execute partial order planning.