CS324: Planning and Agent System. **Spring 2015**

Lecture 19: April 20, 2015

Lecturer: K.R. Chowdhary *: Professor of CS (VF)*

19.1 Introduction

A planning problem involves deciding what actions to do and when. The "when" part of the problem has been traditionally called the *scheduling problem*. This separation is motivated as much by computational considerations as by the fact that algorithm for automating scheduling have been around for a long time (e.g., in the operations research areas) than automating the complete planning problem.

19.1.0.1 Automated Planning

The planning is about how an agent achieves its goals, even the simplest ones an agent must reason about the future. Since the goal is not achievable in single step, the number of steps to be carried out needs to be broken up into subtasks, and steps for each needs to be the goal, what an agent will do in next step also depends on its past.

To complete each subtask, there are actions, which need to be carried out; each action has a subsequent state as well as a preceding state. To be simple at start of this subject, following assumptions are made:

1. the actions are deterministic, i.e., the agent can determine the consequent of the actions,
2. the world is fully observable, i.e., the agent can observe the correct state of the world, and
3. the closed world assumption, i.e., the fact snot described in the world are false.

19.2 The Basic Planning Problem

A basic planning problem usually comprises an initial world description, a description of the goal world, and a set of actions (sometimes also called *operators*) that map a world description to another. A solution is a sequence of actions leading from the initial world description to the goal world description, is referred to as a *plan*.

A *deterministic action* is a *partial function* from states to states. [1]. For example, a robot can move block x onto y, if x has top-clear, y has top-clear, and obviously, $x \neq y$. The that all the alternatives are not available. A *precondition* of an action decides about when the action can be carried out, and resulting state due to an action is the effect of the actions.

[1] Partial function: Every state "(state, action)" pair does not necessarily result to a state.

AI planning techniques are techniques to search for a plan: *forward planning* is a planning technique building a plan starting from the initial state; *backward planning* starts from the goal states, and *least-commitment planning* constructs plans by adding actions in a non-sequential order.

19.2.1 The Classical Planning Problem

The simplest case of the planning problem, where the environment is static and deterministic and the planner has complete information about the current state of the world, is called as the classical planning problem. Classical planners face two important issues: modeling actions and change, and organizing the search for plans (action sequences) capable of achieving the goals.

The classical planning problem can be defined as follows. Given the,

1. a description of the known part of the *initial state* of the world (in a formal language, usually propositional logic) denoted by **I**,
2. a description of the *goal* (i.e., a set of goal states), denoted by **G**, and
3. a description of the possible **atomic actions** (**R** (rule))that can be performed, modeled as state transformation functions,

determine a plan, i.e., a sequence of actions that transforms each of the states fitting the initial configuration of the world into one of the goal states.Thus, classical planning problem is a tuple $\langle I, G, R\rangle$. Consider the following example of planning the Transport a passenger by cab.

Example 19.1 *Transport by cab.*

Suppose that initially (i.e., in all states of the world that match the description **I**), there is a cab at a location A, represented by a binary state variable $cab(A)$, and a passenger at a location B, represented by $passgr(B)$. In each of the states described by **G** the passenger should be at a location C, denoted by $passgr(C)$. Furthermore, suppose that there are three actions that can transform (some part of) the state of the world.

Following are the steps for actions:

1. The cab can move from one location to another: $move(x, y)$ with $x, y \in \{A, B, C\}$. This action requires that a priori $cab(x)$ holds, and ensures that in the resulting state $\neg cab(x)$ and $cab(y)$ hold, that is cab is not at place x as well not at the place y.
2. The passenger can get into the cab: $load(passgr)$. This action requires a priori $cab(x)$ and $passgr(y)$ and $x = y$, and in the resulting state both $\neg passgr(y)$ and $passgr(cab)$ (i.e., passenger in cab) should hold.
3. The passenger can get out of the cab: $unload()$. This action requires that cab is at location x ($cab(x)$) and passenger in cab ($passgr(cab)$), and results in $\neg passgr(cab)$ and $passgr(x)$.

With **I** as initial set of states, and **G** as set of goal states, the sequence of state transitions can be indicated as follows:

I	; Initial state
$move(A, B)$	; cab moves from location A to B
$load(passgr)$	; passenger gets into cab
$move(B, C)$	; cab moves from location B to C
$unload(passgr)$	; passenger unloads from cab
G	; goal: cab at location C

□

19.2.2 Agent types

Agents can be classified according to the techniques they employ in their decision making:

1. *Reactive agents*: They base their next decision solely on their current sensory input.
2. *Planning agents*: They base their course of action considering the anticipated future situations, possibly as a result of their own actions.

Considering the *states set* as $s_1, s_2, \ldots$, and set of *actions* as $a_1, a_2, \ldots$, they are be represented using a tree shown in figure 19.1 or explicitly by a table 19.1.

Table 19.1: Table for mappings: State × Action → State.

State	Action	Resulting State
s_1	a_1	s_2
s_1	a_2	s_3
s_2	a_3	s_4
s_2	a_4	s_5
...	...	...

Example 19.2 *A delivery Robot's (Robo's) planing.*

A delivery robot shown in figure 19.2 is responsible for many jobs. It can pickup mail from mail room and deliver to Shyam's office, and also can pickup coffee from the coffee room and can deliver to Shyam's office. The robot, called *Robo* can also reach these places by moving clock-wise (mc) as well by counter clock-wise (mcc). Assume that mail it handles is postal mail (not the email). The domain of the world is represented by the terminology described as follows.

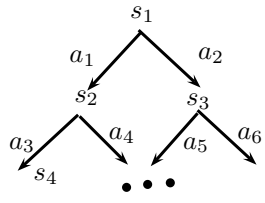

Figure 19.1: World States and Robot Actions.

Various *locations* are represented by following symbols. Each of them can be true/false.

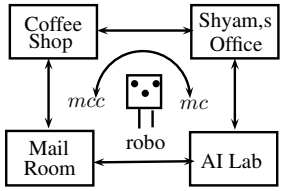

Figure 19.2: Delivery Robot with service locations.

cs: robo at Coffee shop
off: robo at Shyam's office
mr: robo at mail room
lab: robo at AI lab

Following are the various *states* of the world, which can also be true/false:

mw: Mail waiting in the mail room
rhc: Robo is holding coffee
swc: Shyam wants coffee
rhm: Robo is holding mail

Various *actions* performed by the Robo are:

mc: Robo moves clockwise
mcc: Robo moves counter clockwise
puc: Pickup coffee
dc: Deliver coffee
pum: Pickup Mail
dm: Deliver Mail

Presence of an action symbol indicates that action is true.

A state may comprise many parameters or preconditions for the action to take place at that state. For example, the state,

$$\langle lab, \overline{rhc}, swc, \overline{mw}, rhm \rangle \tag{19.1}$$

indicate that Robo is in AI lab, Robo has no coffee in hand, Shyam wants coffee, mail is not waiting, and Robot holds Mail. Another state,

$$\langle lab, rhc, swc, mw, \overline{rhm} \rangle \tag{19.2}$$

indicates that Robo is in lab, Robo is holding coffee, Shyam is waiting for coffee, mail is waiting in mail room, and Robo is not holding mail. The table 19.2 table shows the transitions for certain states.

Table 19.2: Some mapping: State × Action → State, for fig. 19.2.

State	Action	Resulting state
$\langle lab, \overline{rhc}, swc, \overline{mw}, rhm \rangle$	mc	$\langle mr, \overline{rhc}, swc, \overline{mw}, rhm \rangle$
$\langle lab, \overline{rhc}, swc, \overline{mw}, rhm \rangle$	mcc	$\langle off, \overline{rhc}, swc, \overline{mw}, rhm \rangle$
$\langle off, \overline{rhc}, swc, \overline{mw}, rhm \rangle$	dm	$\langle off, \overline{rhc}, swc, \overline{mw}, \overline{rhm} \rangle$
...	...	...

19.3 Forward Planning

It is one of the simplest planning is to treat the planning as a path planning problem in the state-space graph. The nodes here are states, transitions are actions, and results of actions are also states. A forward planner searches the state-space graph from start state for goal state. The figure 19.3 shows the state-space graph for forward planning with start state as $\langle cs, \overline{rhc}, swc, mw, \overline{rhm}\rangle$, with three transitions from start state, corresponding to the actions: pickup coffee (puc), Robo moves clock-wise (mc), and Robo moves counter-clockwise (mcc). If we choose the action puc, the next state is $\langle cs, rhc, swc, mw, \overline{rhm}\rangle$. The new state indicates that robo still remains facing coffee shop; since it pickup coffee - the robot holding coffee is no more false, mail waiting remains true (unchanged).

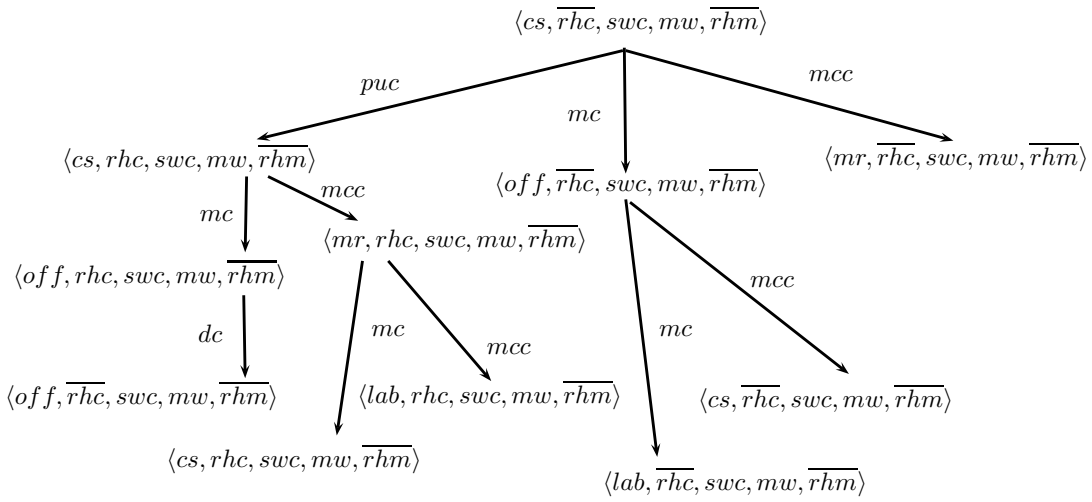

Figure 19.3: State-Space for Forward Planning.

The branching factor in figure 19.3 is 3, and the search can be done in DFS or BFS. Theoretically, since, the Robo can be at any of the four locations, and other four parameters in the state can be true / false, there are $4 \times 2 \times 2 \times 2 \times 2 = 64$ total possible states in the world. Obviously, all of these states are not possible to reach in graph search.

The representation above is simple and clear, but it is not suitable due to following reasons:

- there are too many states to acquire, reason, and represent,
- small change in the requirements will need a major change in the model, for example, if we need to have information about robot battery level to be added as one of the parameter, the entire structure gets modified.

19.4 Partial Order Planning

A partial-order plan consists of the following:

1. A set of steps, each step maps to an operator, except for the start step and the last step. The start step has no preconditions and has the initial state as its postcondition. The final step has the goals as its preconditions and has no postconditions.
2. A set of orderings, each ordering specifies a pair of steps, the first step before the second step. The start step is always ordered before all other steps. The finish step is always ordered after all other steps.
3. A set of causal links. Each causal link specifies a pair of steps and a proposition, where the proposition is a postcondition of the first step and a precondition of the second step. The first step is ordered before the second step.

19.5 Planning Languages

The STRIPS (*STanford Research Institute Problem Solver*) formulation for planning problems employs propositional logic as a language for describing states of the world - a simple but general format for specifying operators, and has a clear semantics. In the STRIPS formulation, there is a set of conditions, for example, a set of Boolean variables to describe states. A complete assignment maps the set of conditions to possible values, for example, truth values, and a partial assignment maps a subset of conditions to values. A state is a *complete assignment*. An operator consists of two *partial assignments* - first is *preconditions* or *results*, which determines the states in which the operator can be applied. The second is *postconditions*, that determines the state resulting from applying the operator in a particular starting state. Implicit frame axioms of STRIPS specify that the value of any condition not mentioned in an operator's postconditions is unchanged by the application of the operator.

The STRIPS is action centric representation, which is based on the idea that most things are not effected by single action. It specifies: *action*, *precondition*, and *effect*. For example, for the goal: "Robo to pickup coffee", we write,

$$\begin{aligned} &precondition : cs \wedge \overline{rhc} \\ &effect : rhc. \end{aligned}$$

The other features are unaffected in above. The action of delivering coffee (dc) is:

$$\begin{aligned} &precondition : off \wedge rhc \\ &effect : \overline{rhc} \wedge \overline{swc} \end{aligned}$$

19.6 Planning with Propositional Logic

Consider the graph/tree shown in figure 19.4, with s_0 as start state, and $a_0 \dots a_{n-1}$ as actions representing a path leading to the goal state s_n. If s_0, $a_0 \dots a_{n-1}$, s_n are considered as propositional expressions, then the equation,

$$p = s_0 \wedge (a_0 \wedge a_1 \wedge \cdots \wedge a_{n-1}) \wedge s_n \tag{19.3}$$

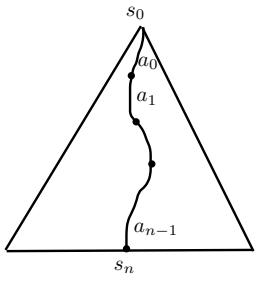

Figure 19.4: Propositional Planning.

is a propositional expression representing the path to goal state from start state. If there is only one goal, then p is the only path to goal, and all other propositions p' are false. The equation 19.3 can always be transformed into a *CNF* (conjuctive normal form) or *SAT* expression. Thus, planning through propositional logic is to find a satisfiability expression, comprising *start state*, *path* and *goal state*, i.e. logical sentence equal to,

$$initial\ state \wedge all\ proposition\ action\ descriptions \wedge goal. \tag{19.4}$$

19.7 Planning Graphs

The planning graphs give better heuristics and consists of sequence of levels, for *time steps* in plan. Each level has literals (constant values) which have become true because of previous action, and each level has preconditions for the next action. The planning graphs represent the *actions* as well as *in-actions*. The following example demonstrates the application of planning graphs.

Example 19.3 *Problem of having a 'Biryani'and eating 'Biryani'.*

$$\begin{aligned}
&init(have(biryani))\\
&Goal(have(biryani) \wedge eaten(biryani))\\
&Action(eat(biryani)\\
&\quad Precond : have(biryani)\\
&\quad Effect : \neg have(biryani) \wedge eaten(biryani))\\
&Action(cook(biryani)\\
&\quad Precond : \neg have(biryani)\\
&\quad Effect : have(biryani))
\end{aligned}$$

The planning graph for these actions is shown in figure 19.5. The box action in the figure indicate the *mutual exclusions* of actions. In the planning graph all the actions A_i at level i contain all actions that are applicable at state S_i, along with constraints saying which part of actions cannot be executed. Every state at level S_i contains all literals that could result from any choice of actions at A_{i-1}, along with constraint saying which part of actions cannot be executed.

19.8 Hierarchical Task Network Planning

Consider an example of "building a house", where the task of house building can be decomposed to: acquiring land, preparation of design map, obtaining the NOC (no objection certificate) from municipal corporation, arranging house

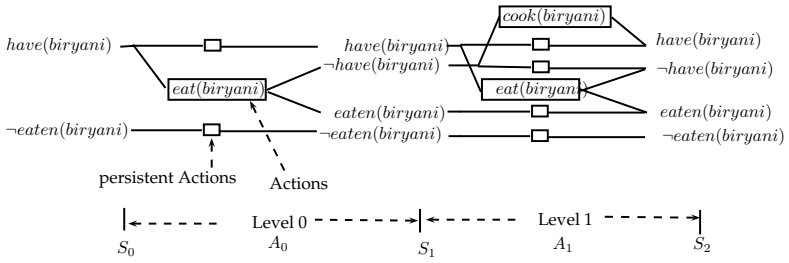

Figure 19.5: A Planning Graph for "Eating a dish, called Biryani."

loan, hiring a builder, paying to builder, and so on, as shown in figure 19.6.

We understand that some of these activities can be done in parallel, but not all. Thus, there is a partial order relation between those actions. The decomposition can be expressed as $decompose(a, d)$, where action a is decomposed into a partial-order plan d. Various activities for building a house as per the plan in figure 19.6 can be formally described as follows.

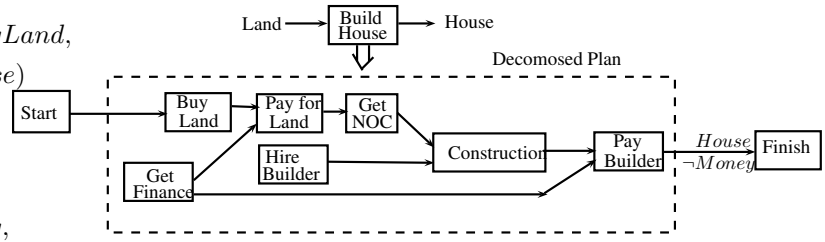

Figure 19.6: Decomposition of task of House Building.

$$Action(Buildhouse,\ Precond : BuyLand,\ Effect : House) \quad (19.5)$$

$$Action(Buyland,\ Precond : Money,\ Effect : Land \wedge \neg Money) \quad (19.6)$$

$$Action(Getfiance,\ Precond : Goodcredit,\ Effect : Money \wedge Mortgage) \quad (19.7)$$

$$Action(Hirebuilder,\ Precondition : Nil,\ Effect : contract) \quad (19.8)$$

$$Action(Construction,\ Precond : NOC \wedge Builderhired,\ Effect : Housebuilt \wedge \neg NOC) \quad (19.9)$$

$$Action(Paybuilder,\ Precond : Money \wedge Housebuilt,\ Effect : \neg Money \wedge house \wedge \neg Contract) \quad (19.10)$$

$$Decompose(steps : \{A_1 : Get\ NOC,\ A_2 : Hirebuilder,\ A_3 : construction, A_4 : Paybuilder\} \quad (19.11)$$

$$Orderings : \{start \prec A_1 \prec A_3 \prec A_4 \prec Finish; \\ Start \prec A_2 \prec A_3\} \tag{19.12}$$

Here $A_i \prec A_j$ indicates that activity A_i precedes the activity A_j.

The linking of states and activities through activities/resources can be expressed as follows:

$$Links : \{start \xrightarrow{Land} A_1, Start \xrightarrow{Money} A_4, A_1 \xrightarrow{NOC} A_3, \\ A_2 \xrightarrow{Contract} A_3,\ A_3 \xrightarrow{Housebuilt} A_4, A_4 \xrightarrow{House} Finish, \\ A_4 \xrightarrow{\neg Money} Finish.\} \tag{19.13}$$