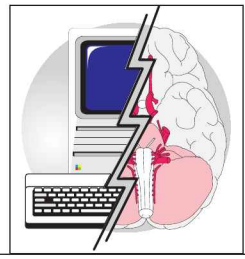
CHAPTER 3



Uninformed Search

Learning Objectives

- To understand searching and problem solving
- To study basics of search techniques and their types
- To understand the importance and applications of search
- To study different uniformed search techniques and their applications
- To understand the performance and quality of different search methodologies
- To know variations in basic uninformed search techniques to overcome practical limitations
- To understand the control strategy and its use
- To understand the various methods of search with reference to complexity of problems

INTRODUCTION

For every problem, we are in search of solution. Solution is typically a state, where all requirements are fulfilled. It can be referred to as the *goal state*. These problems vary from simple selection of place with the best view, selection of flat that can meet our requirements or even reaching a certain destination in time. The goal state, hence, exhibits attributes and fulfils the requirements of user. In case of a product building, the goal state could be a product that can perform necessary functionalities.

To solve problems in real life, we tend to perform certain actions to achieve some predefined goal. Consider a scenario where you want to go to a particular hotel X that serves all kinds of Indian delicacies. But unfortunately, you are not aware of the exact route leading to your goal. You try out the different routes available and finally reach the hotel.

(You are hungry enough!). What you have done is looking out for the solution, inspecting some sequences or approaches. So, search is inspecting the several sequences and choosing the one that achieves the goal. Search, hence, is nothing but the search of goal state. The search begins from the initial state and ends up at the goal state, if any (here, goal state is the state that meets certain criterion). It is equally necessary to consider the time and cost factors during the search process. Generally, everyone prefers to reach to the goal state at the earliest and in less complicated manner. There can be one or more goal states and that would make the problem even more interesting. For example, suppose there are various hotels and you are hungry and you would like to reach to the hotel that serves delicious vegetarian food.

In the above case, the problem is that you are hungry and the solution is food. There can be a number of food joints, but you are looking for good quality of food and also specific type of food you like. This restricts a number of goal states. Hence, out of all possible goal states, we are keen to reach to an optimal goal state, where healthy, tasty and vegetarian food is served, and the place is not far from your initial state. Here, initial state is hungry and a particular location. As we proceed in this chapter, we will discuss the search methods in more detail and define the goal state in more clear way.

It is obvious from the discussion so far that any problem can be treated as a search problem. Here, search is for the goal state. AI problem focuses on the use of intelligence to reach an optimal goal state. Search techniques are the core of AI problems. The problems that can be addressed by the search techniques fall under the categories of agent-based pathfinding problems, two-player games and constraint satisfaction problems. In football, the goal state is typically ball reaching to the goal post. With every pass and kick, there is a state transition. But there are many constraints, which can be rules of game or can be even due to positioning of the players of opposition team. The classic example of pathfinding problem is 3×3 (Eight digits puzzle), as shown in Fig. 3.1 and its variants. This puzzle comprises 8 tiles numbered from 1 to 8 and a blank tile space. Here, the objective is to arrange the numbers by sliding the tiles to get the desired goal. Here, it is not the actual distance, but the search is constrained by legal moves defined by the game rules.

5	4	
6	1	8
7	3	2

1	2	3
8		4
7	6	5

Figure 3.1 Initial state and the goal states of 8 number problem.

Even the travelling salesman problem falls under this category of problems. The example of two-player problems is chess, whereas the constraint satisfaction example is of eight queens. The 8-queen problem involves placing 8-queens on the 8×8 board such that no queen is attacked by the other in horizontal, vertical or diagonal direction when a new queen is placed. A solution is depicted below in Figure 3.2, where no queen attacks another queen on the board:

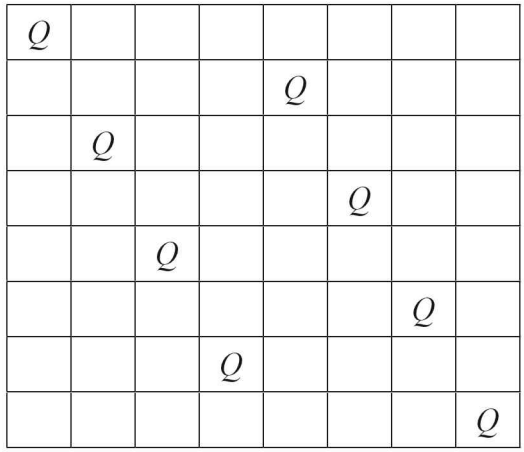


Figure 3.2 8-queens solution.

The search is about the possible legal moves allowed for the next transition and then, it helps in deciding and selecting from the various possible options, where the opponents move is also considered. Though the complexities involved are large, the addition of evaluation (often called *heuristic*) helps further in solving the problem or deciding the appropriate move. Hence, while searching, we should prefer to make optimal use of all the available information. This information builds some sort of guidelines for searching, and hence, reduces the complexity of search.

For the problem domains discussed, a search technique is the one that helps in reaching the solution or a desired goal. Even in real-life scenarios, we face various problems, where there are many possibilities and the optimal solution is generally not known. We need to inspect all the alternatives with reference to legality in this case. We can, thus, say that we are looking out or searching for the solutions. Hence, search is inspecting the various options or the sequences and choosing the one that is the most suitable and helps in achieving the goal. Thus, the expected outcome of search is typically an optimal route from the initial state to the goal state. So, to solve any problem, we need to get a solution or achieve a goal. Mapping the goal to be achieved by the search technique, leads to building a goal-based agent.

We start with the basic concepts along with the definition of the search. We can rightly define *search* as finding an appropriate option, which is provided with some directions. Search explores and evaluates the available options. It examines the states in search of goal. It generates or expands the different states while proceeding towards the solution. Sometimes, it has also cost associated with it. By adding some criteria, the search can also be restricted. So, search is an algorithm that discovers or locates a path to the solution. It makes use of typically called *control strategy* for the search process and posseses some knowledge to get the outcome.

In a broader sense, the search technique is considered to be informed and uninformed. This differentiation is on the basis of how the search actually takes place and what knowledge or information it uses during the search. Informed search, which is also called *heuristic search* is discussed in Chapter 4. In this chapter, we discuss about the uninformed

search strategies (many times referred to as *blind search* or *non-heuristic search*) in detail. Let us begin our discussion with the general search algorithm.

3.1 GENERAL SEARCH ALGORITHM

What do we intend to do while searching is finally to give or to find an appropriate option. The basic or the generalised method to do so is discussed here.

Before we start with the discussion on search algorithms and its techniques, let us discuss problem solving and searching for the solutions.

3.1.1 Searching for Solutions

Searching for solution is typically finding out a path or route to goal state. While doing so, we enter into the various states. It can be referred to as an *intermediate step* during the course of search. Let us start with the basic terminologies, including state space and state-space search.

In generalised terms, a *state space* is defined as a collection of all possible configurations of the system. In short, it is a set of all states that can be reached from the initial states with all possible legal actions. A state space is represented as [*S*, *A*, *I*, *G*].

Here, *S* is the set of nodes or the states for a given problem to reach the solution; *A* is the set of arcs; *I* is the set of initial states and *G* comprises goal states. Set *A* corresponds to the steps required for problem solving; it might have some cost associated with it. One or more nodes can be a part of *I* set and every node has a set of successor nodes. A solution path is a path through the graph from *I* to *G*.

A *state-space search* is the process of searching the state space for a solution to reach the goal. Remember that every node represents a partial path. Now, let us try to develop a generalised search algorithm.

For any problem *P*, we have a set of initial states, say *I*, and a set of goal states, say *G*. We start with the first state and proceed till either we get a solution or return to failure.

Let us assume that the list *L* comprises the nodes or the states from *I*. The algorithm is described as follows:

While list *L* is not empty or has a solution

{
    Remove the state *X* from *L*.
    Take the next decision for the successor.
    Generate a new state for each possible choice for the decision.
    Merge/re-order the new states with *L*.
}

If *L* not empty (has a solution), return success.
Else return failure.

Generation of new states or the expansion mode is dependent on the search strategy used. We will discuss about the various search strategies ahead. It should be noted that in order to simplify the search techniques, it is convenient to represent the problem space as a tree.

3.1.2 Problem-solving Agents

Agent is the most common heard term in AI. What is it? An *agent* can be termed as an entity that can perceive the environment and act on it. So, an agent has an ability to perceive environment and act according to the situation to produce the required results. Hence, when we mention about the goal states and problem formulation, the thing that comes to our mind is agent. An agent, as we are familiar, acts and tries to improve its performance based on the outcome. To achieve so, it adapts to a goal. We can always say that the problem-solving agents are special type of goal-based agents. (Refer to the initial discussion on the problem solving in Chapter 2, to understand how a problem-solving agent works with search).

An agent formulates the goal as well as the problem. These are the basic steps that are required for solving any problem. Then, the actual search method comes into picture. What does the agent do in search once this is done? In search, the agent determines the possible set or rather sequence of actions. This process is carried out so that it can help the agent itself in reaching the states that can ultimately lead to the goal state. So, a search algorithm takes the input of the problem and outputs a solution, which here is an action sequence, that the agent executes finally.

The following steps describe the process:

1. Formulate goal.
2. Formulate the problem which has the goal and the initial states → problem.
3. Now search with the given problem → sequence of actions.
4. Action on the sequence → Act according to the sequence of actions.

Let us proceed towards how control strategies help and play a role in the search techniques.

3.1.3 Control Strategies

Addressing the question, i.e., how to reach or what way to follow while searching for a solution is still remaining. Control strategies are the ones that define this way. So, the rules play an important role in the decision-making; where and which rule is to be applied is decided based on the search strategy. In short, search strategy deals with the overall rules and approach towards searching.

The most common strategies used are discussed below:

Forward search: The strategy here is to explore towards a solution. The main aim is to reach a solution starting from initial state. The methods employing this strategy are often referred to as *data-directed ones*. In this case, your search expedition begins from current position. For example, locating a city from current location in map.

Backward search: In this case, the strategy proceeds, from goal to a solvable initial state. It could be to a solvable sub-problem or a state. Here, the methods employing this strategy are referred to as *goal-directed ones*. Unlike in forward search, if your search begins from target city, it can be termed as *backward search*.

There can also be a combination of these strategies as well, where both the techniques are incorporated. These are as follows:

Systematic search: A systematic strategy is used when the search space is small and systematic, but blind. We call it blind because this is the case where the search has no information about its domain. The only thing it can do is just distinguishing between the goal and non-goal states. For example, you are to locate a particular book in a library, where the books are not indexed and arranged properly. Then, there would be no option but to go blindly and check every book rack. So, systematically, you try to cover all book racks.

Depth first search and *breadth first search* are the two most common methods falling in this category. They also belong to the category of uninformed search. We will discuss these methods in detail further in the chapter.

Heuristic search: *Heuristic* is typically a technique based on the previous experience and provides guidelines to solve the problems. It is a technique that improves the efficiency of a search process. It is exploratory in nature. Imagine a teacher assisting a student in developing some layout of a document. In this process, the teacher may give some important hints to help the student in doing it in the best possible way or may miss some of them. The function of the teacher can be said to have a heuristic approach. Remember that heuristic search may not find out the best solution, but it guarantees to find a good solution in a reasonable amount of time. It uses some measure of relative merits, which can guide the search. For example, if you are locating a city, air distance of the city from current location can guide you to select the direction. This approach is most commonly used where the systems have time or space constraints, and where the dependency is on the knowledge of the problem domain. So, they rely on the promising lines of development to proceed.

This is all about the control strategies that are used in search techniques. But with every search strategy that is used, there are some parameters that measure the effectiveness of the search strategy.

Parameters for Search Evaluation

The main parameters that are used in the evaluation of search are completeness, time complexity, space complexity and optimality. Let us brief on what they mean.

1. **Completeness:** By completeness, we mean that the algorithm finds an answer in some finite time. So, the algorithm is said to be complete if it is guaranteed to find a solution, if there is one.
2. **Space and time complexity:** With space and time complexity, we address the memory required and the time factor in terms of operations carried out.
3. **Optimality and admissibility:** Optimality actually tells us how good the solution is. So, an algorithm or a search process is said to be optimal, if it gives the best solution.

Let us now proceed with the methods and the different types of uninformed search methods.

3.2 UNINFORMED SEARCH METHODS

Uninformed search methods, as discussed earlier, belong to the category of methods that have no prior knowledge of the problem. They are generally exhaustive methods that are brute force, searching all possible paths. Since these methods work in the absence of any information, they are also referred to as *blind search methods.* These methods can be understood by a very simple example. If we would like to search a boy named Ram in a school, with the number of classes from 1 to 10 and each class having two divisions, then the search would involve going to each and every class till the student named Ram, we are looking for, is found. Since no information is available, we need to select certain strategy and expand the nodes (here, the classes) in some predefined order. *Breadth first search* and *depth first search* are the two basic approaches of expanding and visiting nodes in the absence of any heuristic. These are discussed along with some other techniques in the subsequent sections.

3.2.1 Breadth First Search (BFS)

The search technique in BFS follows the shallow node approach. Here, the basic idea is that across the level of the tree, all the nodes are searched. So, it is a search technique, where from the root node, all the successors are searched across the level and expanded. Queue data structure is used to carry out the search, where things happen on first in first out basis (FIFO).

The BFS approach is shown in Figure 3.3. Here, the dotted arrows indicate the search method. With reference to the diagram, the search proceeds with the processing of all the nodes at a particular level and then proceeds to the next one.

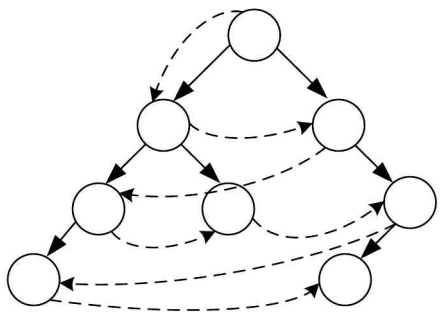


Figure 3.3 Breadth first search tree.

Let us take an example of BFS for better understanding.

Consider that A is the start state and D is the goal state (Figure 3.4). With the help of a queue, the example is demonstrated.

In BFS, the step-wise working is as follows:

Queue	*Check*
A	—
	Removed *A*, is it goal? No, add children. So, *B* and *E* are added.
BE	
E	Removed *B*, is it goal? No, add children.
	So, *C* is added at the end of queue.
EC	
C	Removed *E*, is it goal? No, add children.
	So, *F* and *G* are added.
CFG	
FG	Removed *C*, is it goal? No, add children.
	So, *D* is added.
FGD	
GD	Removed *F*, is it goal? No, add children.
	Cannot be added, leaf node, so remove the next node.
D	Removed *G*, is it goal? No, add children.
	Cannot be added, leaf node, so remove the next node.
Empty	Removed *D*, is it goal? YES!

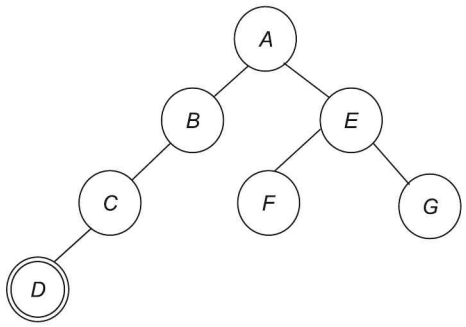


Figure 3.4 Sample tree input for BFS.

From the example discussed here, it is clear that each time you add a node, its children are to be added at the end of the queue as it proceeds level-wise. Since in the example, the goal state lies at the last level, each and every node or state is traversed and added to the queue.

The breadth first search algorithm is summarised as follows:

1. Create a node-list (queue) that initially contains the first node.
2. Do till a goal state is found
   Remove X from node-list. If node-list is empty, then exit.
   Check if this is goal state.
   If yes, return the state.
   If not, get the next level nodes, i.e., nodes reachable from the parent and add to node-list.

If for a given problem, there exists a solution, then BFS is guaranteed to find it out. Hence, we can say that it has the completeness property. It also possesses the property of optimality for the shortest path. Here, the cost of every edge is same, i.e., the cost to reach from one node to another is same. What about the space and time complexities? Let us assume that b is a branching factor or rather the number of successors at every level and d is the depth. Then, the time complexity will be $O(b^d)$—exponential in the depth of the solution. Let us map it mathematically. We start from the root and proceed towards the next level, then the nodes generated at each level are as follows:

$$b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b)$$

This equals to $O(b^{d+1})$, and hence, it comes to $O(b^d)$ and the space complexity is $O(b^d)$, as every node is saved in the memory. Hence, this search is not feasible when you have a large search space.

3.2.2 Uniform Cost Search

Generally, in the search method, with every edge that we traverse, there is a cost associated with it. With BFS, we are assuming that every edge is having the same cost. Now speaking about the uniform cost search, if all the edges do not have the same cost, the breadth first search generalises to uniform cost search. So, the expansion takes place in the order of costs of the edges from the root rather than the order of the depth.

At every step, the next thing to be performed expands/selects a node X, whose cost $c(X)$ is lowest, where $c(X)$ is the sum of cost of edges from root to the node. It is the core step that is followed while performing the uniform cost search. This search technique can be easily implemented using a priority queue.

It is quiet obvious that the search is least concerned about the steps it needs to carry out to reach the goal step. The only concern is the cost. So, there could be a possibility that the search could get in a loop. The search is said to ensure completeness if the cost at every edge is greater or equal to some constant. It, therefore, satisfies the optimal property as well. The uniform search has $O(b^d)$ space and time complexities, as it mostly explores large trees.

Let us consider an example to show how it works.

Consider the following tree given in Figure 3.5, where the uniform cost search is to be applied from the start, i.e., node A to the goal node G. We will use a frontier and an expanded list while generating the solution. (An explored list can also be maintained to keep track of the explored nodes. This list is not considered in this example to avoid confusion.)

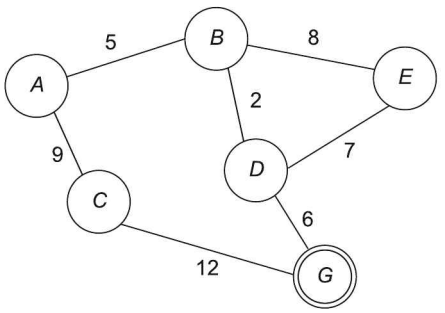


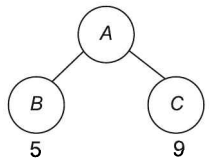
Figure 3.5 Input tree for uniform cost search.

Let us discuss how the uniform cost search proceeds step-wise.

Step 1: Start from *A* Expanded none

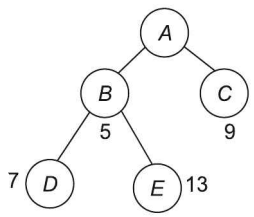
Step 2: Frontier Expanded none

Step 3: Frontier: *B* and *C* Expanded *A*



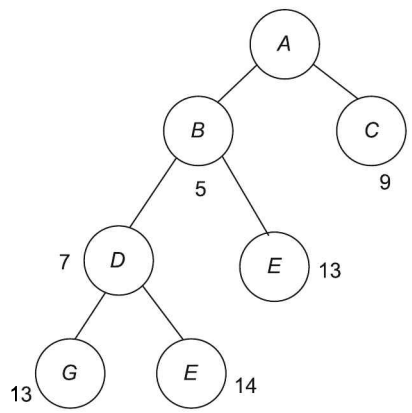
Select the lowest, i.e., *B*

Step 4: Frontier: *D*, *C*, *E* (priority queue) Expanded *A*, *B*



Note: Remember the costs are added from the root till the node in consideration. Select the lowest, i.e., *D*.

Step 5: Frontier (*C*, *E*, *E*, *G*) Expanded *A*, *B*, *D*



In step 5, we have considered *E*, as its parent is different. Since we have reached the goal state, from the tree, we can see that we have found a path. The path is *A*-*B*-*D*-*G*. With the frontiers and the tree generation process, we can easily have the solution to the problem in uninformed search. Dijkstra's approach is often considered to be variant of this uniform cost search.

3.2.3 Depth First Search (DFS)

Unlike BFS, in this method, a single branch of the tree is considered at a time. Taking the same example of finding a route to the hotel, you start with one route and continue it till you come to know that you are at the hotel or you have to change the path, as it will not lead you to the hotel or it is a dead end. While doing so you can think of going back at some point and taking up some other lane/road that can lead you to the place and this is very much possible. This is called *backtracking*. It is the stack data structure that is most commonly used in the implementation of DFS, where the states or the nodes are in operation on first in last out basis.

Speaking in terms of states, you always generate a new state as you proceed through the way.

In search, the same can be formulated as follows:

1. If initial state is the final state (You are just in front of the hotel) → success, exit
2. Do it till you get success or failure.
   Generate a successor, if no more successors can be generated → failure
   Go to step 1 with initial state to be the current generated successor.

The algorithm just discussed above does a DFS. Figure 3.6 depicts the DFS process. As described, the search starts with the initial node or state say root. It proceeds with its successor and follows the path ahead till either is knows that it has come to a solution or has to go back and seek new solution as shown in the figure.

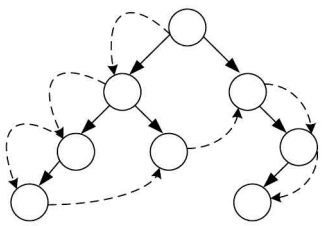


Figure 3.6 Depth first search tree.

Let us take the same example that we had for breadth first search.

Here, we will discuss two cases—where the goal state is D, and where the goal state is F (Figure 3.7).

With the start node of A and the goal state of D, we have the DFS approach carried out as follows:

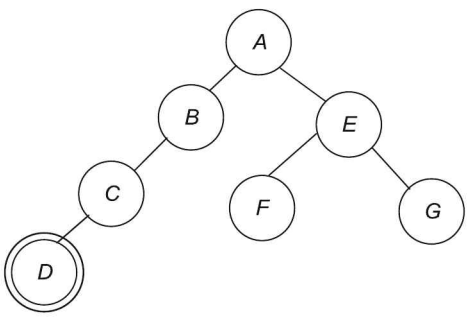


Figure 3.7 Sample tree input for DFS.

	Stack	
Step 1:	$A \leftarrow$ top $B \leftarrow$ top E	Pop A, is it goal? No, push its children on the stack. A's successor is pushed.
Step 2:	$B \leftarrow$ top E $C \leftarrow$ top E	Pop B, is it goal? No, push its children on the stack. B's successor is pushed.
Step 3:	$C \leftarrow$ top $D \leftarrow$ top E	Pop C, is it goal? No, push its children on the stack. C's successor is pushed.
Step 4:	$D \leftarrow$ top E	Pop D, is it goal?? YES!

Pop all the contents. Path found!

So, by traversing from one path in one direction of depth, we lead to the goal state. In this case, no backtracking is required. Consider the second case, where we want to reach to F.

The initial working is same as the way we had to reach to D. The example below shows that after having reached till D state, the next step needs backtracking and hence it starts with E. So, we will start from that point. Thus, the stack contents are as follows:

	Stack	
Step 1:	$E \leftarrow$ top	After having traversed the steps till D, stack will have E. So, pop E, is it goal? No, push its children.
Step 2:	$F \leftarrow$ top G	E's successor is added.
Step 3:	$F \leftarrow$ top G $G \leftarrow$ top	Pop F, is it goal?? YES!

Pop the contents. Path found! So, the path A-E-F is found.

With respect to the completeness of DFS, we cannot say that it will converge. The reason is we go through the selected path and keep on generating the successors and proceed. The search is not optimal. Always remember never use DFS if you suspect a big tree depth. In the worst case, DFS visits all the nodes before getting a solution. The time complexity of DFS, hence, results in $O(b^d)$. At any point of time, we have to keep the path from the root to the current node that is in consideration. This is clear from the example. Hence, considering the depth and the branching factor, the space complexity is $O(bd)$. This is the only good part of DFS. The advantages and disadvantages of DFS and BFS are depicted in Tables 3.1 and 3.2, respectively.

TABLE 3.1 Advantages of Depth First Search and Breadth First Search

Advantages of Depth First Search	*Advantages of Breadth First Search*
1. Simple to implement 2. Needs relatively small memory for storing the state-space	1. Guaranteed to find a solution (if one exists) 2. Depending on the problem, can be guaranteed to find an optimal solution

TABLE 3.2 Disadvantages of Depth First Search and Breadth First Search

Disadvantages of Depth First Search	*Disadvantages of Breadth First Search*
1. Cannot find solution in all cases, and hence, can sometimes fail to find a solution 2. Not guaranteed to find an optimal solution can take a lot of time to find a solution	1. Memory management along with allocation is the key factor, and hence, more complex to implement 2. Needs a lot of memory for storing the state space if the search space has a high branching factor

3.2.4 Depth Limited Search (DLS)

After dealing with the depth first search, we are aware that though it has desirable properties, the probabilities of it getting terminated with wrong expansion cannot be eliminated. Thus, the notion of depth limit comes. The basic idea is not allowing expansion after the certain depth. This process proves to be the most useful if one is aware of the maximum depth of the solution. Imagine that you are out for purchase of a mobile and in a particular area. For convenience, you set 3 streets as depth limit in that area. You begin with street 1. If your budget requirements and features required are available in the first shop itself, then you would stop and purchase there or else go to the next shop on that street. So, possibly at depth of that street, i.e., 1, you have explored all the shops. If the requirements are not satisfied, you would go to the next level, i.e., street 2 and then street 3. (Assumption is that you are not doing a survey!)

The following algorithm summarises the working of depth limited search:

1. Set the depth limit to the maxdepth to search.
2. Initial node = current node
   If initial node = goal, return
3. If depth(initial node) > depth limit
   return
   else
   Expand(initial node)
   Save(successors) using stack
   Go to step 2

Since depth limited search does not find solutions in all cases, it is not complete. The only reason behind it is that if the depth of the solution lies beyond the one, then it is selected. But if the depth limit is greater than that of solution's depth, then it is complete. The search method is not an optimal one. There is no guarantee that the search will give a solution that will be optimal, as it finds the one which is within its limits. With respect to the space and time complexities, the space complexity is same as that of DFS, i.e., $O(bl)$. But with respect to time complexity, it now comes till the order of l, i.e., $O(b^l)$, where l is the depth limit.

3.2.5 Iterative Deepening Search (IDS)

Iterative deepening is an enhanced version of the depth limited search. In some cases, depth limit restricts DLS from finding solution, since solution may exist beyond prescribed

depth limit. Iterative deepening helps in addressing such problems. It further continues the DLS for the next level by extending the depth limit as limit = limit + 1 and so on till we are guaranteed of a solution. We can say that iterative deepening combines the benefits of BFS and DFS. If we look from a broader perspective, breadth first search is a special case of iterative deepening search, with the deepening factor equal to 1 and initial depth equal to 1. Considering the same example of mobile shopping, one can think of some limit, say 3 (number of streets), and possibly, there are 7 streets with the mobile shops. The iterative deepening, as discussed earlier, continues ahead of 3, provided the solution is not achieved till the street number 3. Figure 3.8 depicts the basic concept of iterative deepening.

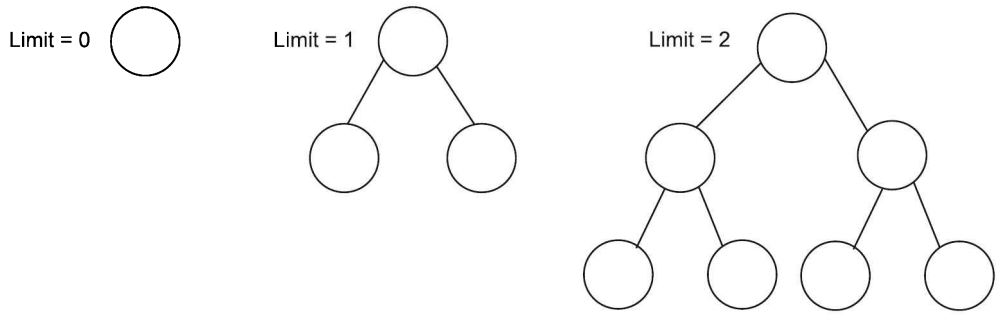


Figure 3.8 Iterative deepening.

The algorithm for IDS is as follows:

1. Set depth-limit ← 0
2. Do
   Solution = DLS(depth-limit, initial node)
   If(solution= goal state) then return
   else
   Depth-limit = depth-limit + 1
   continue

Though the algorithm is complete, the major factor of concern here is the regeneration of tree for every depth searched. That is, at every time we set up a new depth, the previous search results are discarded and we start from the scratch. So, there is a trade-off between the space and the time. With regard to completeness and optimality of this approach, the algorithm is complete and optimal. The time complexity comes to $O(b^d)$ due to the iterative calls. In the sense, higher the branching factor, lesser is the overhead of expanding the states. The space complexity of IDS comes to $O(bd)$. When the search space is large and there is unavailability of the depth factor or rather when the depth factor is not known, then in such cases, IDS method is preferred.

3.2.6 Bi-directional Search

Another variant of this type of search technique is bi-directional search. The search is carried out form both the ends and we are unsure of getting a solution. The search needs to be explicitly specified with the goal state. It comprises forward search from initial stage

and a backward one from the goal state. At some point, they possibly can intersect, and if they do, then we have a solution or else there is no solution. The search is done by expanding the tree with the branching factor b and the distance from start to goal, i.e., d. We can relate this again to BFS that is carried out from both the ends. It is not the case that the bi-directional search always uses BFS as a path to search; it could be with the best first or the other method too. Bi-directional search can also be a part of heuristic.

Figure 3.9 depicts a typical bi-directional search, where search begins from both initial state and goal state. Consider a problem with the depth as 4. Here, if we have the search paths working in both the directions, one at a time, then in the worst case at depth of 2, they would meet. Thus, the bi-directional search is complete and at the same time, gives an optimal solution as well. The time and space complexities come to $O(b^{d/2})$ with BFS working in both directions and the space comes to $O(b^{d/2})$.

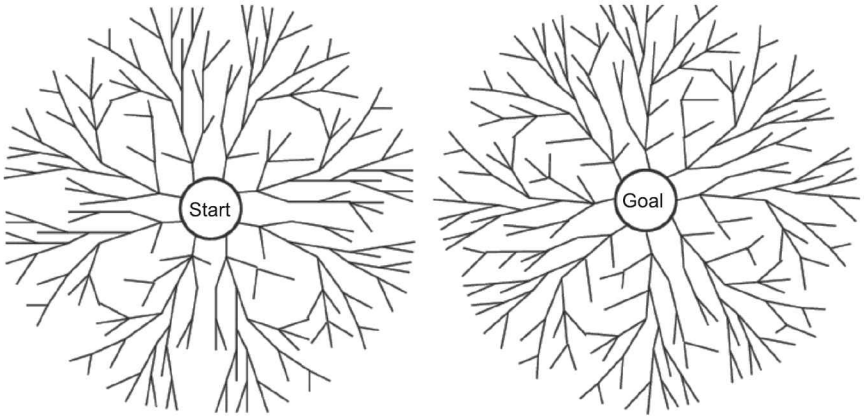


Figure 3.9 Bi-directional search.

Bi-directional search is not that easy to implement as one would think of. The implementation approach needs to be very efficient such that the intersection of the two approaches is possible.

3.2.7 Comparison of the Uninformed Techniques

Table 3.3 summarises the complexities, optimality and completeness about the search techniques.

TABLE 3.3 Comparison of Uninformed Search Techniques

Search techniques	*Complete*	*Optimal*	*Time complexity*	*Space complexity*
Breadth first search	Yes	Yes	$O(b^d)$	$O(b^d)$
Uniform cost	Yes	Yes	$O(b^d)$	$O(b^d)$
Depth first	No	No	$O(b^d)$	$O(bd)$
Depth limited	No	No	$O(b^l)$	$O(bl)$
Iterative deepening	Yes	Yes	$O(b^d)$	$O(bd)$
Bi-directional	Yes	Yes	$O(b^{d/2})$	$O(b^{d/2})$

SUMMARY

In this chapter, we have discussed about the basic search approach and the problem domains handled by the search techniques. The control strategies that help in defining the way to reach the solution have also been discussed.

In search techniques, an agent plays a critical role. It is the agent that has to formulate the goal and the problem prior to the start of the search process. An agent takes appropriate action with reference to the scenario to lead to the goal state. The different techniques of search are then applied to reach the solution. With the existence of different strategies, the search techniques are classified in a broader sense into informed and uninformed. Uninformed search are blind search techniques with restricted information availability. The effectiveness of the search is often measured in terms of the parameters of completeness, optimality, and time and space complexities. With every approach having its own flavour, there is always a trade-off between the selections of any search method and is totally dependent on the type of problem that is to be solved. In some cases, it is the problem space size that has to be considered, while in some cases, it is the efficiency that matters. Breadth first search and depth first search are the two major approaches in uninformed search. Further, there are variants like depth limited search, iterative deepening search and bi-directional search.

If any additional information other than the costs is available for the search, then it would be more effective. This is what a heuristic search does. So, by providing the guidelines, a heuristic search, which is also called informed search, helps in making it more effective.

KEYWORDS

1. **Search technique:** It is a technique that provides with some appropriate option in reaching the solution or a desired goal.
2. **State space:** A state space is defined as a collection of all possible configurations of the system.
3. **State-space search:** A state-space search is the process of searching the state space for a solution to reach the goal.
4. **Control strategy:** It is a strategy that defines a way to follow while searching for the solution.
5. **Forward search:** It is a strategy that proceeds from the initial state towards the solution.
6. **Backward search:** It is strategy that proceeds from the goal state to the solvable initial state.
7. **Systemic search:** It is a blind search technique that is systemic in nature and is applied for small problem space.
8. **Uninformed search:** It is a search technique that is blind and does not make use of the information about the problem. It simply distinguishes between the goal and non-goal states. Also, it is systemic in nature.

9. **Heuristic search:** It is a technique that improves the efficiency of search technique by providing guidelines.
10. **Optimality:** A search is optimal when the best solution is generated by the approach.
11. **Completeness:** Completeness defines that the algorithm/approach will give the solution if there is one.
12. **Breadth first search:** This refers to a shallow approach where nodes are searched all across the levels.
13. **Uniform cost search:** It is a breadth first search, where the search takes place with respect to the cost of the edges unlike the normal BFS.
14. **Depth first search:** It is a search technique where path from one node to its children on the same line is continued. So, it is a search which goes first deeper and then across branches.
15. **Depth limited search:** This refers to a version of depth first search, where the expansion is limited to a specified depth limit.
16. **Iterative deepening search:** It is an extension of depth limited search, where if the solution is not found at the specified depth limit, it carries the search to further levels of depth.
17. **Bi-directional search:** It is a search technique, where the search is carried in both the directions—forward and backward.

MULTIPLE CHOICE QUESTIONS

1. Your friend is in a building that has 9 floors and you want to locate him. Which search technique would you use?
   (a) Depth first search (b) Breadth first search
   (c) Depth limited search (d) Iterative deepening.
2. In uninformed search
   (a) Heuristic plays a very critical role in the decision of next successor
   (b) There is no information about the nodes
   (c) The best path is selected
   (d) None of the above
3. A parent wants to put his child in a good school. There are many schools in the city, but the parent's preference is to the one that is nearby his house. Which of the following search technique(s) would the parent use in the selection of the school?
   (a) Depth first search (b) Uniform cost search
   (c) Depth limited search (d) All of the above
4. If any search algorithm is able to generate a solution, then the search is called
   (a) Efficient (b) Optimal (c) Complete (d) Informed

5. Which of the following search technique uses a priority queue?
   (a) Breadth first search (b) Depth limited search
   (c) Iterative deepening (d) Uniform cost search
6. The time complexity of iterative deepening comes to
   (a) $O(bd)$ (b) $O(d^b)$ (c) $O(b^d)$ (d) $O(b^l)$
7. Which search technique can be considered as the special case of iterative deepening?
   (a) DFS (b) BFS (c) Bi-directional (d) None of these
8. Which search is effective in the easy detection of cycles?
   (a) DFS (b) BFS (c) Both (a) and (b) (d) None can detect
9. In searching techniques, a node/state is marked to be visited when
   (a) It is pruned (b) Its successors are traversed
   (c) It is expanded (d) Both (b) and (a)
10. The branching factor determines
    (a) The cost for the search
    (b) The number of operators that can be applied
    (c) How close the path is to the solution
    (d) All of the above

CONCEPT REVIEW QUESTIONS

1. What is a problem-solving agent?
2. Distinguish between depth first search and breadth first search.
3. Discuss about the time and space complexities of the uninformed search techniques.
4. In which search techniques it is necessary to maintain the previous states?
5. What is the difference between completeness and optimality?
6. Show with an example that the time complexity of DFS is $O(b^d)$.

CRITICAL THINKING EXERCISE

1. For $b = 10$ and $d = 5$, compute the overhead between the depth limited search and the iterative deepening search.
2. Can you map a greedy approach to any of the search methods discussed? Discuss with an example.
3. Is it possible to relate any of the searching techniques in a case when any Bluetooth-enabled device is looking for the other Bluetooth-enabled devices?
4. Can you think of different possible combinations of DFS and BFS?

PROJECT WORK

1. Develop a program to solve the eight queens problem. Which strategy would you use for the same? Discuss about its complexity parameters.
2. Consider the graph given in Figure 3.10.

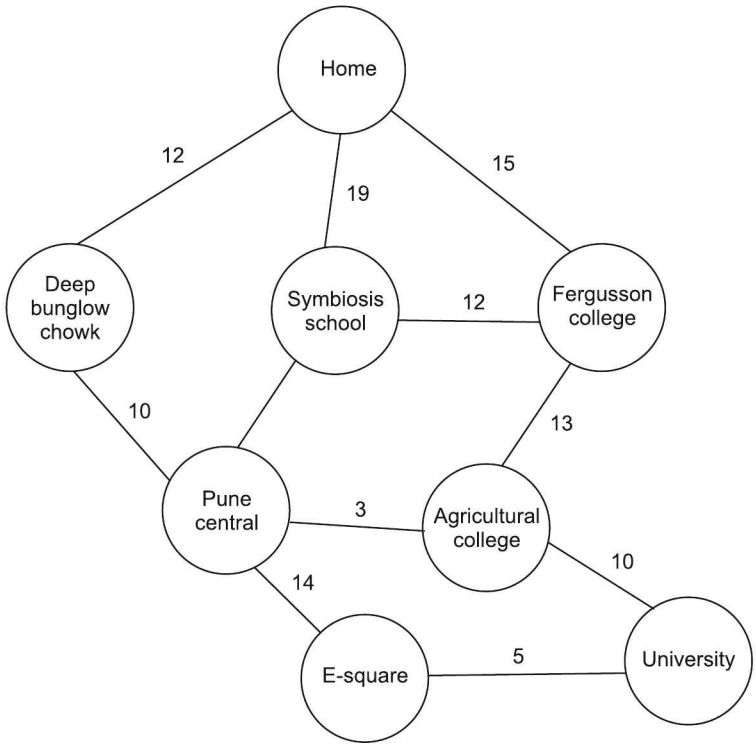
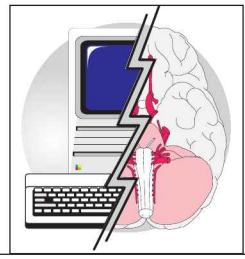


Figure 3.10

Apply the following methods to the above graph:

(i) Implement uniform cost search, where start node is Home and goal is E-square.
(ii) In the absence of the distance parameter, implement DFS and show step-wise how the goal state is reached.
(iii) In the absence of the distance parameter, implement BFS and show step-wise how the goal state is reached.
(iv) In the absence of the distance parameter, implement DLS and show step-wise how the goal state is reached, where the limit is 3.
(v) In the absence of the distance parameter, implement IDS and show step-wise how the goal state is reached, where the limit is 3.

CHAPTER 4



Informed Search

Learning Objectives

- To appreciate the concept of solving complex problems through search
- To understand the various search methods evolved over the time
- To identify the various search applications and algorithms
- To know about the details of the various search algorithms
- To understand the relationships among different search techniques
- To understand the different heuristic or informed search methods, and their variants
- To develop understanding to select an appropriate search method in context of a problem
- To appreciate the role of search in AI
- To analyse decision complexity of the algorithm and process of problem solving through search

INTRODUCTION

As we have discussed in Chapter 3, search is one of the most important topics in AI. Any problem can be presented as a search problem. In all the cases, we are in search of a solution. This solution could be a particular destination, particular medicine, or even can be a particular person or a document. The search complexity increases with the complexity of the problem. A very simple search method may work for very simple problems, but may fail to handle a few tricky situations and may end up by getting trapped some times in the local minima, and sometimes, it returns a solution which is not optimal, and in most of the cases, it takes a lot of time to arrive at a solution.

These applications (where search methods can be employed) may vary from simple selection of an object from a stack of objects to the complex problems like route selection,

forecasting scenarios, and even to the search of the best strategic position. This chapter provides reader more insight into the search techniques and equips them to solve different class and complexity problems with search paradigm while elaborating the heuristic search methods in a greater detail.

Uninformed searches (that we have studied in Chapter 3) lead to higher time and memory complexity. Optimising a search problem and optimisation of the whole process are challenging tasks. Right heuristic and right application of heuristic are some of the important aspects of efficient search techniques.

Thus, there is a need for informed or heuristic search. Let us begin the discussion with informed search techniques. Informed search techniques are also called *heuristic search techniques/ strategies*. These strategies use the information about the domain, or knowledge about the problem and scenario to move towards the goal nodes. Heuristic methods do not always find the best solution, but they guarantee to find a good solution in reasonable amount of time. For this, they sacrifice the completeness, but then try to increase the efficiency. We are aware that many search problems are NP-complete, so in the worst case, they can have exponential time complexity, but a good heuristic can find a solution for a problem efficiently. Hence, these methods are useful in solving tough problems, i.e., the problems that take longer time to compute solution. A very classic example of heuristic method is travelling salesman problem.

Heuristic Function

A *heuristic function* is the one that guides the decision of selection of a path. A heuristic function $h(n)$ provides an estimate of the cost of the path from the given node to reach to the closest goal node. Let us discuss the various methods in heuristic.

4.1 GENERATE AND TEST

The most easiest and obvious way to solve any problem is to generate a solution and check if this is the real one. The algorithm for the method is discussed below:

1. Generate a possible solution, which can be a node in the problem space or it can be a path from the initial state.
2. Test if the possible solution is the real one, i.e., compare with the goal states.
3. Check solution, if true, return the solution, else go to step 1.

Now you would be feeling as if you are studying depth first search, but here complete solutions are generated before testing. It is often called the *British Museum method*. Now, from where does heuristic come into picture? Heuristic is needed to sharpen the search. While generating the solutions, we can have an intermediate option between the generation of complete solutions and random solutions. This approach proceeds systematically and does not consider the paths that are unlikely to lead to the solution. Now, this evaluation of which path should be considered is done by the heuristic function.

4.2 BEST FIRST SEARCH

Best first search is a combination of depth first search and breadth first search. Each of the methods—depth first and breadth first have their own positive aspects that are exploited

in the best first search. The positive aspect of DFS is that the goal can be reached without any need to compute all the states, whereas with BFS, it does not get halted or trapped in dead paths. The best first search allows us to switch between the paths, and thus, gets the benefit of both. To do so, the selection of the most promising node is done. So, it analyses and checks if the selected node is better than the previous one. If not found so, it reverts back to the previous path and proceeds. So, backtracking occurs. In order to achieve this, even though a move is selected, other options are kept so that they can be revisited.

Before we proceed to the algorithm and example of best first search, we will brief on OR graphs.

OR Graph

The notion of OR graphs is required in order to avoid the revisiting of paths and for propagating back to the successor, in case it is required. In this case, for the algorithm to proceed, a node contains the following information:

1. Description of the state it represents.
2. Indication how promising it is.
3. Parent link that points to the best node it has reached from.
4. List of nodes that are generated from it.

A graph employing these descriptions is called *OR graph.*

To have this type of graph search, we describe further two lists of nodes, viz., open and closed.

1. *Open list:* It consists of list of nodes that have been generated and on whom the heuristic function has already been applied, but yet not examined. We can map them to a priority queue. This queue consists of the nodes or the elements with highest priority. This is dependent on the heuristic function that associates the cost with the node, and hence, prioritises them.

2. *Closed list:* It contains the nodes that have already been examined. When a node is generated, then this is required to check that whether it has already been generated.

So, what is heuristic and the function evaluation procedure, $f(n)$? $f(n)$ is the function that can be defined as the sum of two elements, i.e., the cost of reaching from the initial node to the current node, $g(n)$ and the additional cost of getting from the current node to the goal state, $h(n)$. So, $g(n) + h(n)$ defines the function.

Now, let us discuss the *best first search algorithm.*

1. Open list ← initial state/node.
2. Do (till goal is found) or (no nodes in the open list).
   (i) Select the best node (since priority queue is used, this is possible) from open list. Add to closed list.
   (ii) Generate the successors.
   (iii) For every successor, do
      (a) If the successor is already generated, change the parent if this new path is better than the earlier one. Accordingly, update the costs to this node as well as to the previous successors, if any.

(b) If the successor is not generated previously, then evaluate it on the basis of heuristic, add it to the open list and make a note of its parent as well.

Consider the following graph given in Figure 4.1:

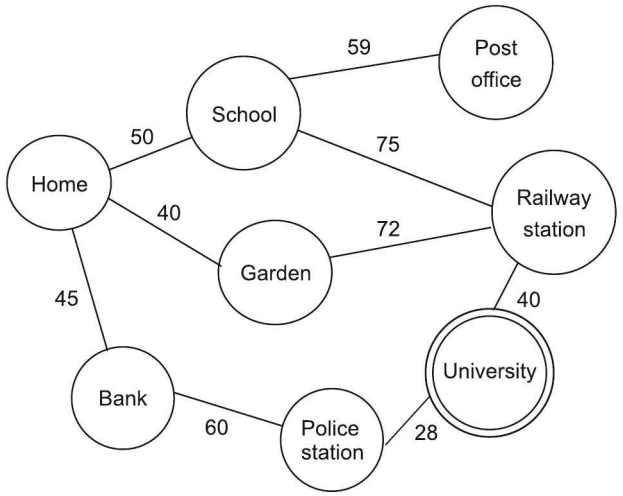


Figure 4.1 Graph sample example with sample graph.

The above graph is an example that is referred in the chapter to show the working of the methods. The values on the edges represent the distances or the costs. For the above graph, it is shown further that how a best first search finds a solution from Home (as the initial state/node) to University (as the goal state). As discussed earlier, the best first search is about using heuristic, which is used to calculate the cost. Consider the following hypothetical h values, which are straight line distances that are available from the states to reach to University (goal state).

So, here $f(n) = h(n)$. This example explains the states to proceed towards the goal of University. The values in the example represent the heuristic (you cannot actually compute the values; they are assumed to be computed on the basis of some previous experience), and hence, as the name suggests, the search proceeds in a greedy fashion.

Home 120
Bank 80
Garden 100
School 70
Railway Station 20
Post office 110
Police station 26

The approach proceeds by selecting the node that is closest to the goal state. It starts with Home, selecting the lowest heuristic, it goes to School (see Figure 4.2). From School, it proceeds towards Railway station and then to University. In this course of search, the node or the state of Police station is not considered as the part of evaluation process.

Let us come to the discussion of the properties of this algorithm. The best first search is not complete. The only reason is that heuristic puts every option or path to be the

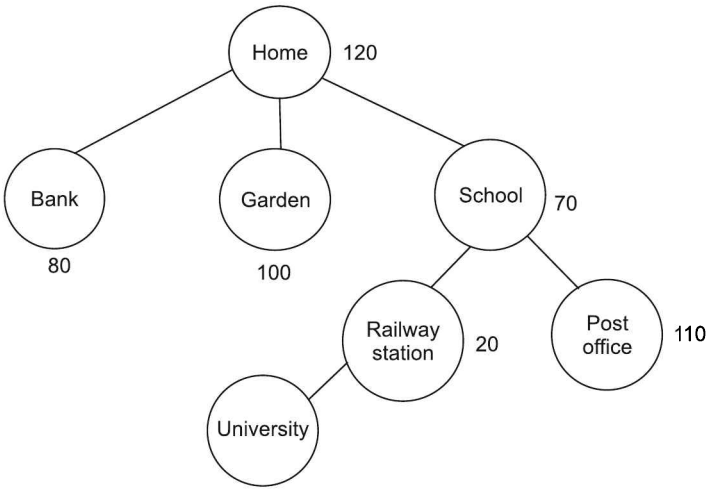


Figure 4.2 Best first search.

best one. But broadly speaking, heuristic that works reasonably will lead to a reasonable solution, and hence, will not get trapped into this problem. About optimality, it is neither optimal. The reason behind this is the selection of minimum value h. The search could also lead to dead ends. Time and space complexities of this algorithm are $O(b^m)$, where m represents the maximum depth. Since we require maintaining a queue for the successors that are not generated, the space complexity is $O(b^m)$.

4.3 GREEDY SEARCH

There is a long debate as to whether the best first search and the greedy search are same. So, we need to first understand the relationship among A*, best first and greedy. The best first is said to be the simplified version of A*. Now speaking about the greedy and best first, it is the best first search that is named as greedy best first search only. Also, a point that is worth a mention is that there is a family of best first search algorithms, which have different evaluation function. In some cases, it could be mapped as $f(n) = h(n)$ as well.

4.4 A* SEARCH

The best first search that we have learnt is a simplification of the A* approach. This algorithm was presented by Hart. A* is the most widely used approach for pathfinding. It is the A* approach that now makes use of the function $g(n)$ along with h. Evaluation function $f(n)$ represents the total cost. Here $f(n) = g(n) + h(n)$, where $g(n)$ is the cost so far to reach n, while $h(n)$ is the estimated cost from n to the goal state. The best first search is a special case of A*, where $f(n) = h(n)$. Let us discuss the algorithm in detail.

1. We begin with the open list containing only the initial state.
   (a) Let $g(n) = 0$ and $h(n) =$ value whatever it is.
   (b) So, $f(n) = h(n)$ as $g(n) = 0$.

2. Repeat: Do till we get the goal.
   If no nodes are existing in the open list, return failure to reach goal.
   Else
   1. Select and remove node from the open list, with the lowest f value.
   2. Let us call this node the current node or C-node.
   3. Put C-node on the closed list.
   4. Check if C-node = goal state, return solution.
   5. If not, then generate the successor for C-node and for every successor,
      (i) Save the path from the successor to C-node. (Required to retrieve the solution path)
      (ii) Calculate g(successor) ← g(C-node) + Cost of reaching to successor from C-node.

Now, three cases are to be handled.

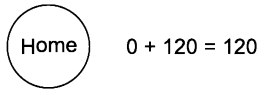
1. Check if the successor is on open list.
   This means that the node has been generated previously, but yet not considered in the path.
   (a) If so, throw this node (let us call this as Pnode) and add it to C-node's successor list.
   (b) Now, a decision is to be made whether the connect path between the Pnode's parent is set to C-node. This is dependent on the cost. If this cost is cheaper, then it should be. (This is true as successor and the P-node are same). This is done by comparing the g values. If g(successor) is less, reset P-node's link and update f(P-node) considering the value of g(P-node).
2. Check if the successor is on only closed list.
   If the node is present on the closed list, let us call this node as P-node.
   Add this node to the C-node's successors. Again check the costs as we have done in the previous step. Accordingly, update the g and f values.
   If this path is better, this needs to be rectified and informed to the P-node's successors. But how to inform the successors, as every successor is divided into further successors and so on. But remember, there is a stage when you get a node that is still on the open list and is a part of this successor family or does not have successor at all. To accomplish this, we have DFS that changes the g values, and in turn, f values as well. This is done for each branch till it terminates or a node to which a better path or the same value path has already been found.
   Let us make things simple. We are aware that every node's parent link points awards the best parent. As we propagate, we need to keep a track of whether the nodes' parents also point to this best parent; this is the same node from where we propagate. If it is, let the propagation continue, else g value specifies the better path and then the propagation may stop there. But it is quite possible that this new propagation might lead us to a path that is far better than the current. In that case, we need to compare and decide further. If the current path is better, stop propagation, else reset the parent and continue further.
3. Check if the successor is neither on the open list nor on the closed list.
   The most simple case, simply add it to the open list and the list of C-node's successors. Compute f(successor) = g(successor) + h(successor) and proceed.

Let us consider the same example as discussed for the best first search. Now, when it comes to A* approach, the function costs are updated. Consider that we need to find a solution from Home to University. It is the same case which we have discussed in the best first search. The steps of A* approach are shown below:

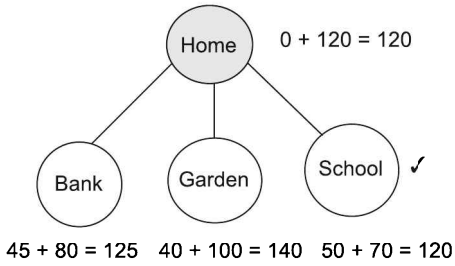
Initial state: Home
Goal: University

The detailed path for the search tree formation is represented here. At every point of time, the heuristic and the distance are taken into consideration.

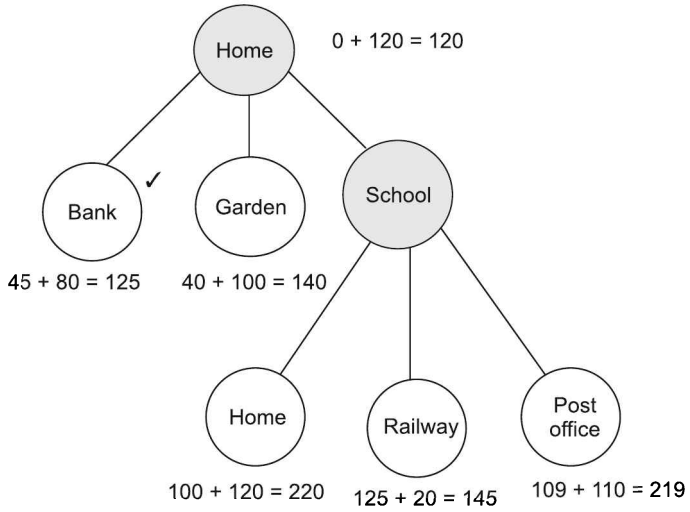
Step 1:



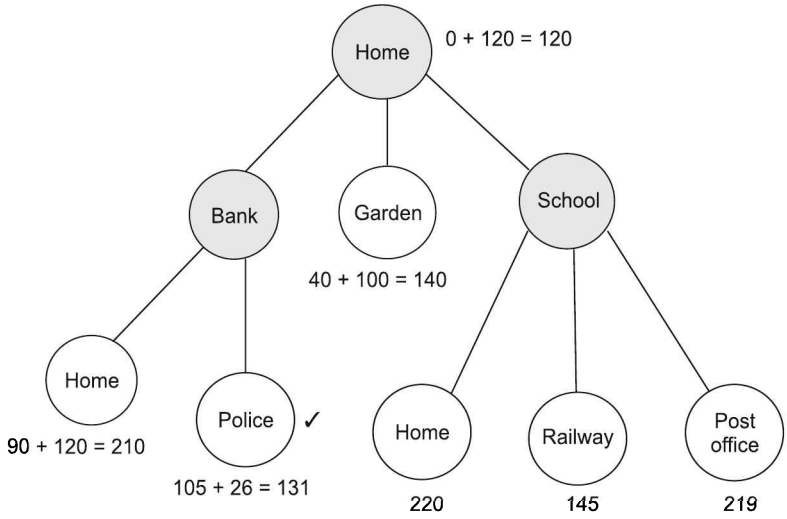
Step 2:



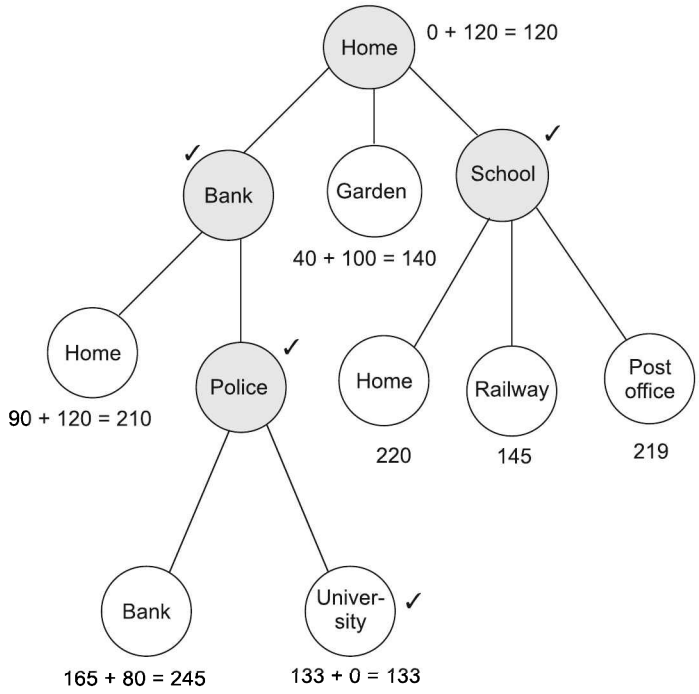
Step 3:



Step 4:



Step 5:



Step 5 shows the final tree generation from the Home node to the University. The ticked nodes refer to the expansion process being carried out.

4.4.1 Admissible Heuristic

Any search algorithm is admissible if it always produces an optimal solution. Here, it means that it would find the optimal solution as the first solution. So, it is required that the cost is optimal. That means for any node, say n, the heuristic, $h'(n)$ would be the cost of an optimal path from node n to the goal state. Thus, heuristic $h(n)$ is admissible if for all nodes n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the actual cost to attain goal state from n.

To understand it better, let us consider Figure 4.3 showing an example related to it.

Underestimating h

Referring to Figure 4.3, it is discussed here how h' underestimates h. Let us assume that A* is applied and we are at the intermediate stage that is mentioned. Here, *Init* has been removed from the open list and the stages (where we are having the nodes/successors expanded) are represented. Now, we can see that here Q node's evaluation function $f(Q) = h(Q) + g(Q)$ is equal to 6. Since this is the lowest cost, this node is selected. Next, S is the only successor of Q. $f(S)$ is same as $f(P)$. Here we continue with our expansion, so we reach to T. Now, $f(T) = 9$, whereas $f(P) = 7$; so we go back and expand P. Thus, due to underestimation of $h'(P)$, efforts and memory are wasted.

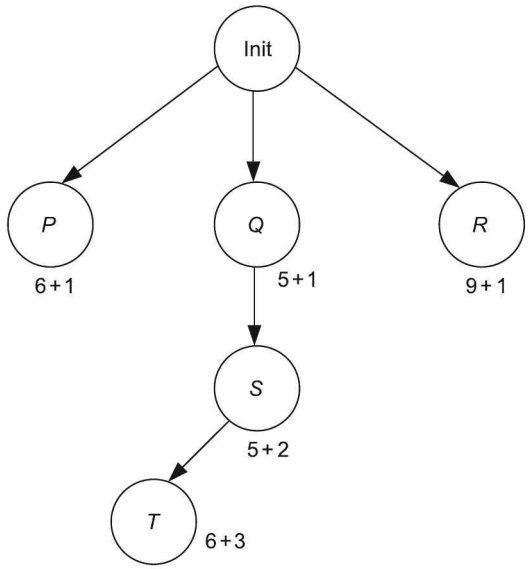


Figure 4.3 Underestimating h.

Overestimating h

Referring to Figure 4.4, now consider a scenario, where the heuristic function is overestimated. In the process of reaching to the goal state, we can see that from node Q, we move to S. $f(S) = 7$, and hence, being the lowest, it is selected. Ahead in the process, T is selected as $f(T) = 8$ and then we reach to U(say our goal state). Suppose there is direct path from P to goal state giving path length less than 3, we will not be able to find it. So, we can always say that if h' overestimates h, then, we have no assurance of getting the cheapest path.

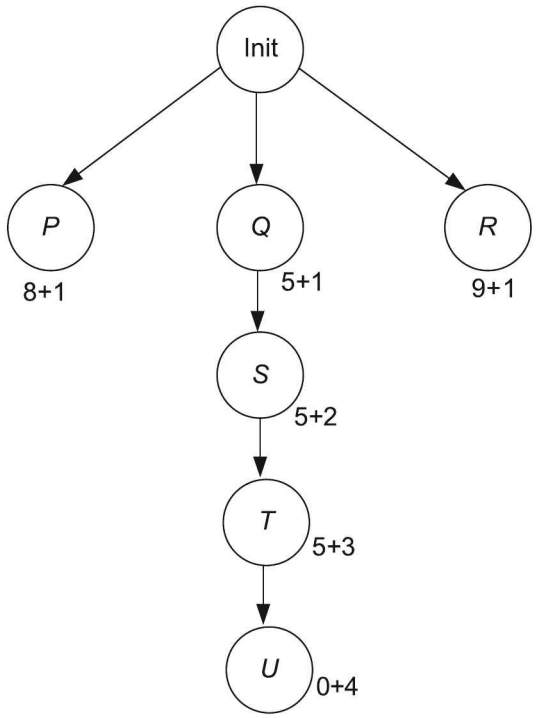


Figure 4.4 Overestimating *h*.

Then, the question is raised that if this overestimation does not occur, is the algorithm of A* admissible? We cannot guarantee whether *h'* will underestimate or overestimate A*. The only way is to set it to zero, but then this will lead to BFS.

Here, heuristic $h(n)$ is admissible, if for every node the heuristic cost $h(n)$ is always less than or equal to the actual cost required to reach the goal state from n, i.e., $h(n) \leq h^*(n)$, where $h^*(n)$ is actual cost to reach goal state. The heuristic that never overestimates the cost is an admissible heuristic.

4.4.2 Consistent Heuristic

Let us define what is meant by consistent heuristic. A heuristic $h(n)$ is said to be consistent, if for every node, say n and every successor s that is generated by an action a for n, the estimated cost to reach the goal state/node is not greater than the step-wise cost to reach s added to the cost of reaching the goal from s.

Mathematically we can say that for $h(n)$ to be consistent,

$$h(n) <= \text{cost}\ (n, a, s) + h(s)$$

Mapping this to the triangle inequality, we can say that each side of the triangle cannot be greater that the total of the remaining two sides.

4.4.3 Optimality of A*

Let us consider in which situation A* gives an optimal solution. Considering the algorithms, where the search is carried out from the root node, A* is said to be optimally efficient for any heuristic. By this, we mean that there exists no other algorithm or approach, which uses the same heuristic and can generate a fewer or lesser nodes than the ones generated by A*.

Time and Space Complexity

With respect to the time complexity, we always say that the time complexity depends totally on the heuristic function. In the worst case scenario, just imagine the number of nodes that would be expanded! It is exponential to the length of the solution. Mapping the sub-exponential growth mathematically, we get $\log[h^*(n)]$.

Here, $h^*(n)$ is the true or the exact cost to reach the goal state from node n. So, the error of h now grows faster than the logarithm and it is least proportional to the cost of the path. Hence, we cannot say that it definitely results into an optimal solution. In such cases, we can use variants of A* or some other better heuristic functions.

Time complexity of A* is not the main hindrance. The main issue of concern here is the space complexity. Since all the nodes that are generated are kept in the memory, A* algorithm runs out of space. Hence, it is recommend not to use A* for large-scale problems.

4.5 MEMORY BOUNDED HEURISTIC SEARCH

Why is it necessary to have memory-bound heuristic search? Generally, we tend to run out of the memory before time! This occurs when the entire search paths are stored and the only solution to overcome this is to remember partial solutions. Thus, in order to overcome the space requirements, memory-bounded heuristic search comes into picture. Iterative deepening A* (IDA*), Recursive best first search (RBFS) and memory-bounded A*(MA*) are the algorithms that fall under this search. Let us begin our study with IDA*.

4.5.1 IDA*

We are already aware that in simple iterative deepening, the search is restricted by the depth limit, running it recursively. So, it tries to resolve the problem of breadth first search. On similar grounds, IDA* resolves the problem of A*, where the memory problem is overcome, and at the same time, optimality is also maintained.

In IDA*, at each iteration, depth first search (DFS) is applied. A track is maintained of the costs, i.e., $f(n) = g(n) + h(n)$ of each and every node that is generated. Whenever a node is generated whose cost is more or exceeding the threshold for that iteration, the path is discarded. Then, comes the backtracking method. The initial cut-off is set to be equal to the heuristic of the root node. So stepwise, the approach can be summarised as follows:

Step 1: At first, set the limit = h(root). We can call this as f-limit.

Step 2: The next step is pruning if any node does not satisfy the limit condition, i.e., prune if f(node) > f-limit.

Step 3: Set f-limit to be equal to the minimum cost of any node that is pruned.

The search terminates when the goal state is reached, whose cost does not exceed the current threshold.

From the approach, we can see that IDA* actually performs a series of depth first searches. Thus, the space constraint or the requirement is linear with respect to depth. Further things are dependent on the heuristic function as well. If an admissible heuristic exists, an optimal solution is guaranteed. Thus, we can always conclude that IDA* is optimal in terms of time and space. Most importantly, it reduces the overhead of managing the open list and the closed list. Although in IDA*, we cannot deny the problem of excessive node generation.

4.5.2 Recursive Best First Search (RBFS)

It is the simplest recursive algorithm that performs the best first search with a linear space. The basic idea here is to remember the best path or the best alternative and backtrack if the best first gets very expensive, in the sense when the cost exceeds that of previously expanded node.

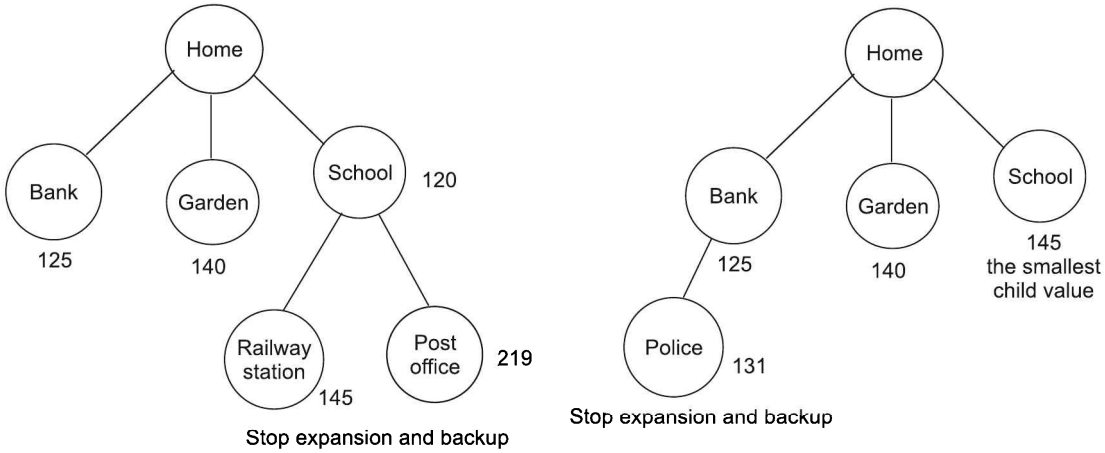
But then, tracking of the costs is required simultaneously with the backtracking. This means the *f*-value needs to be maintained. So, keeping the track of the *f*-cost or the *f*-value of the alternative paths that are available, the search process backtracks if the current path becomes expensive. As the decision to backtrack is taken, the *f*-cost of the current node (i.e., expanded) is replaced with the best *f*-cost of its children. Recording the *f*-cost enables the algorithm further to take decisions about whether there is a need to expand the tree further or not.

The algorithm for RBFS is explained below:

1. If curr_state = goal state
   Return curr_state (success!)
2. Expand(curr_node) → children
3. If empty(children) return failure
4. Do
   for each *c* (child of the curr_node)
   $f[c]$ = maximum($g(c) + h(c)$, f[curr_node])
   best cost=lowest of *f*-cost, i.e., $f[c]$
   if(best cost >prev *f*-cost found) return failure
   consider the alternative *f*-cost
   goto step 1 with (best-cost node, min (*f*-cost, alternative *f*-cost))

The following example (Figure 4.5) shows a snapshot of how RBFS works for the same problem of Home to University.

Commenting on the optimality of RBFS, it is also optimal if the heuristic is admissible. Evaluation in terms of time complexity is very difficult, as it is totally dependent on the switching between nodes and the heuristic. The space complexity comes to $O(bd)$, where b is branching factor and d is depth. The major drawback of RBFS and IDA* is the generation of same states again and again. The memory utilisation for the methods is just the *f*-cost that is required. So, if we are having more memory, why not to use it? Keeping this in mind, memory-bound variants MA* and SMA* (or Simplified Memory Bounded A*) approaches are used.



Example 4.5 RBFS snapshot.

4.6 HEURISTIC FUNCTION

We have already taken into account, in brief, the basic concept of heuristic and its function. Let us proceed further to explain the heuristic accuracy, its effect on the performance and invent admissible heuristic functions.

The discussion here is made with reference to the eight-puzzle problem. Following diagram shows initial and final states for eight-puzzle problem (Figure 4.6).

5	4	
6	1	8
7	3	2

1	2	3
8		4
7	6	5

Example 4.6 Eight-puzzle problem.

The puzzle is one of the popular heuristic search problems. The decision of heuristic is dependent on the moves taken. In the puzzle, 22 steps is the average cost required to reach goal state. The b value (branching factor) is set to at the most 3, but it comes to 4 only when the middle tile is empty. It also comes to 2 or 3 depending on the tile position. While solving this problem, it would require 3^{22} states generated. The state generation factor is restricted by keeping track of the repeated states. But when it comes to the increase in size of the puzzle, what can be done? So, there is a need for good heuristic function. Let us say that that we are using A*. There is need to have heuristic that will not have overestimation of h. With the previous efforts taken into account, the two h-value estimations are as follows:

Case 1: Here, the value of h can be the number are in expected position of misplaced tiles. At the initial state, it is equal to 8, where none of the tiles are in expected positions.

Since no tile is in expected position, every tile needs to be moved at least once. Hence, minimum moves required to reach goal state are 8. Thus, h is admissible.

Case 2: Here, h is the sum of moves required for the tile to reach the goal positions. So, the horizontal and vertical moves are counted. This is even referred to as *Manhattan distance*. In this case too, h is admissible, as any move of the tile brings it near to the goal. In this case, it comes to 18, which is less than the solution cost of 26.

4.6.1 Effect of Heuristic Accuracy

How can we say that the heuristic decided/estimated is the best? This characterisation of deciding the quality is the effectiveness of the branching factor. Let b^* be the effective branching factor, N be the number of nodes generated by A* and the depth of the solution be d. Using Manhattan distance, we can write it as follows:

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots (b^*)^d$$

This b^* can give a guideline in terms of quality of h. A good heuristic has b^* value near to one, and hence, can allow large problems to be solved. When we look at the two cases of h, then we come to know that h in case 2 is better. Why? Referring back to the point of admissible heuristic in A*, h in case 2 results in the expansion of all nodes that are expanded by h in case 1. But case 1 possibly expands other nodes too. So, taking up h with larger value is always better, but it is required to check the overestimation factor too! So, the value of h affects the performance.

4.6.2 Inventing Heuristic

The concept of heuristic has been crystal clear now. Things become easy when you try and relate to solve the problems on your own, but what about the heuristic that has been invented by machine?

There is a concept of relaxed problem. We have come across the word *relaxed rules*, where the criteria are somewhat less restricted or stringent. Similarly, in problem solving, if the rules are bit simplified with less restrictions on the actions, the problem is said to be a relaxed problem. Some of the constraints in the original problem are removed to convert into relaxed problem. Now, the *admissible heuristic* for the original problem is the cost of an optimal solution that is applied to a relaxed problem. This is admissible because the optimal solution of the original problem is also a solution of the relaxed problem. A simple example of relaxed problem in eight-puzzle problem is less restriction on the movement of the tiles. From these relaxed criteria, we have the mapping for the cases 1 and 2 stated earlier for h. But if the relaxed problem is also difficult to solve, then inventing the heuristic becomes more difficult.

ABSOLVER, a heuristic generator problem is designed by Prieditis that can generate heuristics automatically using the relaxed method approach and other methods as well.

Further, one can also think of the issues like if multiple heuristic values can be estimated, then which is to be selected? As discussed earlier, it is better to have the maximum value. So, the maximum of h is selected for the search.

Heuristic Building from Experience

Relaxed problem is a method for building heuristic. But there can be other possibilities also. This can be on the basis of some previous experiences. While solving the problem, you would get multiple optimal solutions. Each of these solutions can be experienced and can be used as examples to learn and build h. Many techniques can be used for the same. *Inductive learning* is a method that can help in building h function.

4.7 AO* SEARCH

A* algorithm uses open and closed lists to maintain the node status. AO* instead maintains the entire graph that has been generated till the current state. So, every node or state stores its h-value rather than storing the g-value. The algorithm is explained below:

Algorithm considers threshold value such that if h(start) > threshold, solution becomes impractical.

1. Initially, graph contains only start node/state. Compute h(start).
2. Till start is labelled as solved or h(start) > threshold, repeat
   (i) Generate/trace the children from start.
   (ii) Select promising node that is not yet expanded, let this be C-node.
   (iii) Generate C-nodes successors, if none, then C-node is not solvable. Hence set h (C-node) = threshold else for each successor, do
   Add to graph,
   if a terminal node-then label it solved and assign to it $h = 0$
   else compute its h-value.
   (iv) Propagate this information back. This is done as follows:
   If X is the set of nodes whose values are changed or is labelled as solvable, then
   (a) Select a node from X, whose descendents appear in the graph. If none, select any other node from X. Let this be select_node and remove from X.
   (b) Compute the cost of reaching (arcs) other nodes from select_node. This is equal to the sum of h values to reach there. Assign select_node the minimum of these values as its h.
   (c) Set the best path out of select_node that has the minimum cost computed earlier.
   (d) Label select_node as solved if all nodes connected to it are already labelled solved.
   (e) If select_node is labelled as solved, then its status must be propagated. So, add its ancestors to X.

It is necessary to mention that the propagation of the cost is through a long path, which impacts adversly on the efficiency of the algorithm. Along with this, we cannot forget the point that the graph may have cycles. This further increases the complexity of the algorithm.

4.8 LOCAL SEARCH ALGORITHMS AND OPTIMISATION PROBLEMS

Local search begins with one of the initial solutions and iterates, exploring the search space. Each step here is a search of solution, which is achieved through the moves

associated with reference to the neighbourhood definition. Here, each step is a typical transition to one of its neighbourhood from present state. The move is based on the value of cost function. In local search algorithms, a single current state is selected and algorithm tries to improve it.

Two important parameters to be considered in local search technique are—selected initial solution and the stop criterion. Stop criterion typically defines the conditions when the search phase is over and the best solution found is returned.

Local search is based on the local view. Let us take an example of a car driver driving a car on a road with a number of curves up the hill, with possibility of multiple junctions and routes. Let us assume that he/she is driving on that road first time. He/She can see only the road close to her but cannot see the top or even beyond the next curve. His/Her decision about the route and speed depends on the local information and slope. Hence, there is only one way to check, based on the local information. The choice of the route and further selection of action among the possible actions for improving the objective function can only be made based on the local solution. It must be made by observing near or local solutions only. Though this mimics most of the real situations, the issue with the local search procedure is that no one can assure that the best solution is found. In short, it is not known whether the solution found is locally optimal or globally the best solution. There can be different ways or search strategies to deal with such situations. Local search algorithms do not guarantee an optimal solution. Generally, local search methods search non-optimally with reference to certain stop criterion. The effectiveness and widespread applicability make these techniques more appealing. There are many optimisation paradigms like evolutionary algorithms that sit on the top of simple local search techniques. In local search, larger the elements in the neighbourhood, more is the difficulty in exploring and better is the quality of local optimum.

Let us define three main entities for local search problem:

1. Search space
2. Neighbourhood relations
3. Cost function

State Space

It is a typical representation in the form of a graph, where nodes represent partial or complete solution of the problem and each of the arcs corresponds to the problem-solving process.

Neighbourhood Relations

In general, a *neighbourhood* of a point is a set or region containing the point, where one can attain that point without leaving the region. For a given state—$s(s \in S)$, a set $N(s)$ represents neighbouring solutions that can be attained in the given region. This $N(s)$ is called *neighbourhood of s*.

Cost Function

The selection of the move to be performed at each step of the search is based on the cost function. It is the cost function f that is associated with each element $s \in S$, and is represented as $f(s)$ that assesses the quality of solution.

4.8.1 Hill Climbing Search

In search methods, we continuously update the search space with reference to the goal state. In hill climbing, the search begins with a random point in search space. The basic idea of hill climbing is to always head towards a state which is better than the current state. Always move to the neighbour with a better score. Neighbour is typically a state that is reachable from the current state with any possible legal action. The neighbourhood needs to be small enough to achieve a better efficiency. The best neighbour among neighbourhood is picked. In case, there is no neighbour better than the current state, then the program stops. Since it is always moving to the node, which has a better value, it is very greedy and can easily stuck.

In steepest ascent hill climbing, the next state is always made the best successor of the current state, and it only makes a move if that successor is better than the current state. Local maxima and local optima are the major issues of this algorithm. There can be many local optima. Even there can be ridges. Figure 4.7 depicts the same.

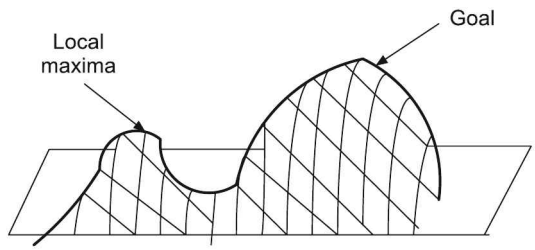


Figure 4.7 Local maxima.

Hill climbing search strategies expand the current state to evaluate children. Obviously, the best child is selected for further expansion—neither its sibling nor its parents are retained. Since it keeps no history, it cannot recover from failures of its strategy. In small settings of specific environment, though this strategy works very well, it may not be appropriate for many real-life scenarios because of the shape of the entire space. Typically, heuristic helps in deciding the direction of search.

Simple Hill Climbing

Typical steps in simple hill climbing are listed below:

1. Let *IS* be the initial state and *GS* be the goal state. At first, the initial state *IS* is evaluated.
   Check if *IS* = *GS*? If yes, then quit.
   Otherwise continue with the initial state as the current state *CS*.
2. Continue until the solution is found.
   (a) Apply an operator so far not applied to the current state and apply it to produce a new state *NS*.
   (b) Check for the goal state if *NS* = *GS*? If yes, then quit.
   Otherwise if it is better than the current state, then make *CS* = *NS*.
   (c) If *NS* is not better than the current state, then continue the loop.

Here, to determine whether the current state is better or not, an evaluation function is used. This calculation of evaluation function is based on heuristic. Generally heuristic is based on certain task-specific knowledge. Hence, this method is heuristic-based search method. This knowledge helps in building heuristic and solves some intractable problems. In this method to determine whether the state is better or not, precise evaluation is required. Hence, evaluation function-based state selection is required to make this search method work efficiently and accurately. Generally, a better state means a higher value of evaluation function.

There are a few possible issues with this algorithm. One of the most important issues is completeness. It can be a typical extension to generate and test using the knowledge that directs the search. This knowledge is based on heuristic function that detects the closeness of current state to the goal state.

EXAMPLE: Let us assume that we are solving travelling salesman problem with hill climbing.

1. Begin with the initial state (that is a city salesman has visited).
2. Move in the direction so that the city is not repeated, but additional city is visited (better state).
3. Heuristic can be the number of cities visited.
4. Keep moving in the state that improves the number of cities with legal action.
5. When there is no way to improve heuristic function, stop.

Hill climbing algorithm makes incremental changes in the optimal direction based on heuristic.

There are many variants of hill climbing algorithms. Let us discuss them.

Steepest Ascent Hill Climbing

It is a simple variant of hill climbing that considers all possible moves from the current state and selects the best current state. This method is also called *steepest ascent hill climbing* or *gradient search,* since it tries to select the state that gives the maximum gain. The algorithm works as follows:

1. Let *IS* be the initial state and *GS* be the goal state. At first, the initial state *IS* is evaluated
   Check if *IS* = *GS*? If yes, then quit.
   Otherwise continue with the initial state as the current state *CS*, i.e., *CS* = *IS*.
2. Continue until the solution is found or there is no change in the current state.
   (a) Let SS_1, SS_2, ..., SS_N be a set of successor states of *CS*. Let *SS* be any possible successor state. Apply an operator so far not applied to the current state and apply it to produce a new state *NS*.
   (b) Check for the goal state if *NS* = *GS*? If yes, then quit.
   Otherwise if it is better than all successor states, then set this state to *SS*.
   (c) If *SS* is better than current state then set *CS* to *SS*.

To apply the steepest ascent hill climbing to the travelling salesman problem, we need to consider all successor states (possible moves) to choose the best one. This algorithm has higher complexity than the basic hill climbing, since it requires selecting move from a number of possible moves.

Both basic and steepest ascent hill climbings may not able to reach the goal state, and hence, may fail to find the solution. Generally, it terminates in finding a state that does not have a better state in the vicinity or from where it is not possible to get to a better state in a single move.

Problems with Hill Climbing

We can see from the example that the process may reach a solution where no other better solution is available in percept. In short, no transition improves the situation. Typically, this happens when it reaches a local maximum, plateau or ridge.

1. **Local maximum:** This is a state better than the local region or neighbouring states, but not a global maximum. This occurs since a better solution exists which is not in the vicinity of the present state.
2. **Plateau:** This refers to a flat area or space, where neighbourhood states have the same value as the present state and hence, fails to determine the best direction to move on. This situation needs big jump in some direction or in selecting a new part or section of the search space.
3. **Ridge:** It is the search space at a higher altitude than the surroundings. It cannot be traversed by a single move. It is a special kind of local maxima. Ridges create a challenging problem, since hill climbing adjusts one element in vector at a time and keeps moving in axis-aligned direction. It may require to move in multiple directions at once to find the solution. Ridges are illustrated in Figure 4.8 and the problems are depicted in Figure 4.9.

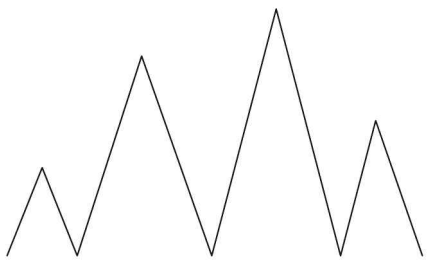


Figure 4.8 Ridges.

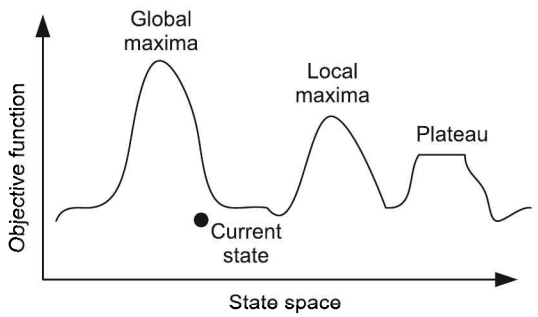


Figure 4.9 Hill climbing: Problems.

A few approaches to deal with these problems are:

1. In case, the process gets stuck at some local minima, then backtrack to some earlier node and try to move in some other direction. This is a better and acceptable option, in case, there is some other path that exists at an earlier node and looks promising. To implement this strategy, it is required to maintain earlier nodes and corresponding paths in the memory.
2. Another option is to take a big jump in case the process gets stuck. This may allow going to a different section of the search space. This is a very good option to deal with the plateau.

Variations of Hill Climbing

1. *Stochastic hill climbing* chooses randomly among the neighbours going uphill.
2. *First choice hill climbing* generates random successors until one is better. It is good for the states with high numbers of neighbours.
3. In *random restart hill climbing,* the sideway moves restart from a random state.
4. *Evolutionary hill climbing* represents potential solutions as strings and performs random mutations. It keeps the mutations that are better states. It is a particular case of first choice and the ancestor of the genetic algorithms.

Stochastic Hill Climbing

It is a variant of basic hill climbing. While the basic and steepest ascent hill climbing algorithms select the steepest uphill move, stochastic hill climbing selects at random among all possible uphill moves. This can help in addressing some issues in simple hill climbing like ridges. Even in case of travelling salesman problem, it can help in reaching an optimal solution.

First Choice Hill Climbing

It is a variant of stochastic hill climbing. In this algorithm, the successor is generated randomly until the one generated is better than the current state. When there are a large number of successor states then this is a better option.

Random Restart Hill Climbing

It tries to overcome the other problems with hill climbing. It conducts a series of hill climbing searches from randomly generated initial states. Each run from the series of runs is like an independent hill climbing algorithm that progresses until it reaches to a position from where no progress is possible. It saves the best results among all results. This can help in getting an optimal solution. This search approach is complete and its probability is close to 1. If each hill climbing search has probability p_1 to succeed, then obviously $1/p_1$ restarts are required.

Hill climbing algorithms are very sensitive to the shape of search space. A small number of local maxima can be handled by random restart hill climbing very efficiently. In such cases, it can find optimal solution very quickly. Generally, NP-hard problems have very high, rather exponential, local maxima and may get stuck. To great extent, the

problem of local maxima in simple hill climbing is addressed efficiently by the random restart hill climbing.

Evolutionary Hill Climbing Search

It represents potential solutions as strings and performs random mutations. It keeps the mutations that are better states. It is a particular case of first choice and the ancestor of the genetic algorithms. This is genetic algorithm-based search.

4.8.2 Simulated Annealing

Hill climbing always moves uphill and there are no downhill moves. As a result, it can get stuck to the local minima, hence it is incomplete. A purely random walk does not care whether it is uphill or downhill and randomly selects a successor. Hence it is very expensive, though complete. One algorithm is greedy and efficient but incomplete, while the other one is very inefficient but complete. Hence, it makes sense to combine the best of both the algorithms. Simulated annealing tries to do that. It is based on metallurgy concept of annealing. This is about cooling of material in heat bath. Typically in annealing, metal or glass is heated to a very high temperature and then it is cooled gradually. This helps the molecules to settle in such a way that the material condenses to low energy crystalline state. This increases the strength of material due to proper low energy condensation. At high temperature, there is a very high energy that allows random moves. The random and bad moves that are allowed in the beginning are controlled as the temperature goes down. Figure 4.10 depicts the random jumps that can occur.

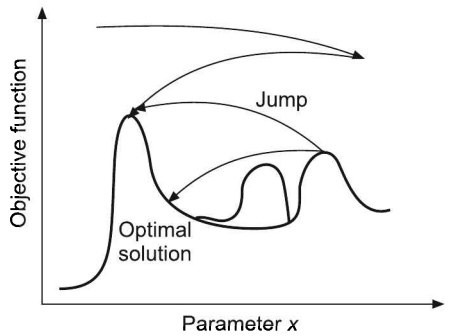


Figure 4.10 Simulated annealing.

In gradient descent model, the energy of search is increased to such an extent that it gets out of the local minima. In this process, it crosses most of the local maxima and settles at minima close to the optimal solution maxima, if the energy is reduced slowly to make it settle down. In gradient descent model, the high energy search, i.e., the random jumps are reduced slowly to get an optimal solution. Initially, the energy of a ball is increased to such a level that it is not obstructed by any of the maximas. Figure 4.10 depicts same. If the energy is then reduced slowly, it will settle near the valley of the global maxima. Next part of simulated annealing will follow algorithm very similar to hill climbing. It is not

greedy as instead of best move it picks random move. In case, the move leads to a better state, it is always accepted. In case, it could not, it is accepted with lower probability p. The assessment of the state in terms of betterment or inferiority decides the probability. Figure 4.11 shows the basic steps.

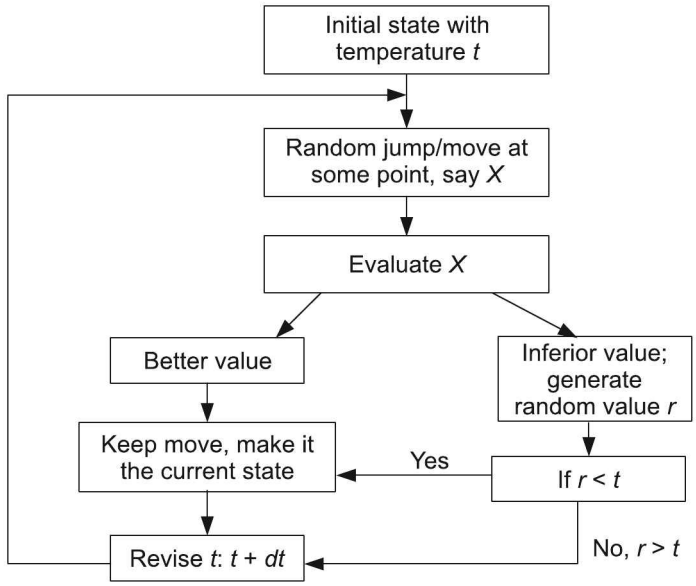


Figure 4.11 Simulated annealing: Basic steps.

The algorithm for simulated annealing is as follows:

1. Let *IS* be the initial state and *GS* be the goal state.
2. Check if *IS* = *GS*. If true, return success.
   Else
   (i) Initialise t for *IS*, where t is temperature or energy level to allow maximum movement
   (ii) Do while a stop condition is not satisfied.
      (a) Randomly pick a neighbour, say x.
      (b) Evaluate x.
      (c) ΔE = Energy_val(x) – Energy_val(*IS*)
      (d) If x = *GS*, return success.
      (e) If $\Delta E < 0$, make x to be *IS*. (This means new state/solution is a better one)
      (f) Else generate random number: r[0, 1].
         If $r < e^{-\Delta E/t}$, make x to be *IS*.
      (g) Revise t values.

Simulated annealing can be used in layout problems, arrangements, scheduling and multi-solution optimisation problems.

Let us study the example of eight-tile puzzle for better understanding. The problem solution is the one where all the tiles are in required order. The tiles need to be moved one at a time to the empty slot. From the initial condition given, it is possible to have different

states. So, from every new state that is generated, different states are possible. Simulated annealing works on the principle of computing the energy. This energy is computed based on the distance measure. For example, consider it to be an absolute difference (Position where the number is – Position where the number should be). The temperature is set to some value, which is decremented or revised at every iteration.

Let us assume that the following initial and goal states are given (Figure 4.12):

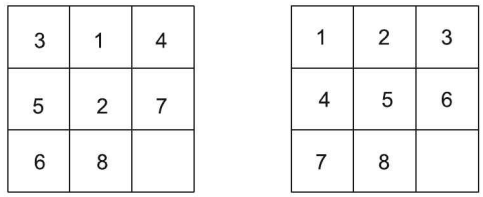


Figure 4.12 (a) Initial state, (b) Goal state.

Energy for the initial states is given as:

$(2 - 1) + (5 - 2) + (3 - 1) + (4 - 3) + (5 - 4) + (7 - 6) + (7 - 6) + (8 - 8) = 1 + 3 + 2 + 1 + 1 + 1 + 1 + 0 = 10$

We are aware that there are two different cases possible after this state. So, we have them, as shown in Figure 4.13 below.

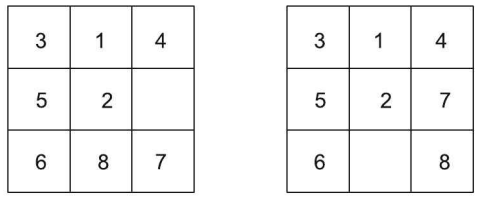


Figure 4.13 (a) Energy 12, (b) Energy 11.

Since any random neighbour is selected and in either case, the difference in the energy for the current and initial states is not less than zero, we need to consider the probabilities and select the next state. Let us assume the temperature to be 50. The probability assumes that the first state selected will be

$$e^{-(12-10)/50} = 0.96$$

So, depending on the value of r, the state can be considered to be the initial state. Do remember that since the temperature is reduced at every iteration, chances are less that it selects a worst state.

4.8.3 Local Beam Search

In hill climbing, we keep just a single node in memory. This cripples the algorithm to a great extent. *Beam search* is a heuristic search algorithm. This algorithm explores the states

or graph by expanding the most promising nodes in a limited set. At any level, it expands identified best node. There can be more than one node identified at each level, say *K*. It is based on breadth first search to build search tree. The algorithm steps are as follows:

1. Maintain *K* states and not just a single state.
2. The search begins with *K* randomly generated states.
3. At each iteration, all possible successors of *K* randomly generated states are identified.
4. If the goal state is found, then halt, else select *K* best of the successors.

K number of best nodes are expanded at each level. Hence, *K* is the width of the beam. If *B* is the branching factor, then *B***K* number of nodes will be evaluated at each depth. Out of that, only *K* nodes will be selected. Hill climbing is the special case of local beam search, where $K = 1$. In case of no restriction on *K* it becomes breadth first search.

In local beam search since each of the *K* threads runs parallel and useful information is passed/shared among all *K* threads, good successors are selected. The algorithm quickly stops unfruitful searches and focuses on *K* most leading searches. It may suffer from lack of diversity among *K* states and may abandon sometimes searches which could have led to an optimal outcome. Even many times, it becomes concentrated because of typical patterns in regions.

EXAMPLE: Let us assume that we want to select the best engineering students from the country, the steps are as follows:

1. First select *K* states from the country with the best engineering results.
2. Select *K* cities with the best results.
3. Select *K* colleges with the best results.
4. Select *K* courses with the best results.
5. Select *K* divisions with the best results.
6. Select *K* students who scored maximum marks.
7. Select a student among them with the maximum marks.

A typical local beam search is depicted in Figure 4.14. The beam width is of 2 nodes and the filter width is of 3. Nodes with dashed line indicate that they are pruned. Black coloured nodes are the beam nodes and the normal ones are those that are selected for evaluation.

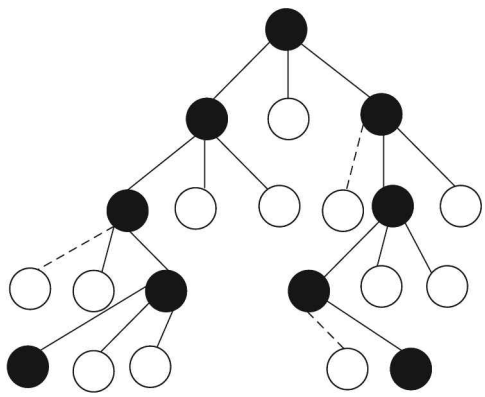


Figure 4.14 Local beam search.

Stochastic Beam Search

It is a variant of local beam search and is very similar to stochastic hill climbing. Instead of selecting K successors which are the best ones, stochastic beam search selects K successors at random. This selection is done in such a way that the probability of choosing a given successor is an increasing function of its value. This process has some similarity to genetic algorithm or natural offspring selection.

4.8.4 Tabu Search

Tabu search was created by Glover in 1986. Local searches, we have discussed so far, are focused on neighbours and have tendency to get stuck in suboptimal maxima. These methods are based on short-term memory to prevent reversal of recent moves. The idea of tabu search is very similar to the other variants, where when the searches get stuck in suboptimal region, the non-improving moves are tried to search the solution. Tracing and backtracking to previous nodes in case of local minima are prevented in tabu search with the use of list call tabulist, which is nothing but a list of recent history of search. Steepest ascent is very similar to it. Hill climbing when combined with short-term memory results into simple tabu search. Search space and neighbourhood structure are the two basic elements of tabu search heuristic.

With tabu search, one point to mention is the memory. Tabu list with long-term memory maintains history all through the exploration process, whereas short-term memory keeps the most recent ones. Tabu search flow is depicted in Figure 4.15.

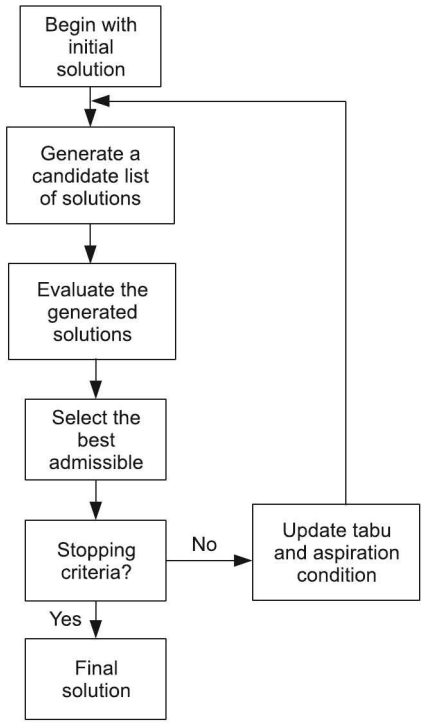


Figure 4.15 Tabu search: Flow.

The algorithm steps are as follows:

1. At any instance, the algorithm begins with some initial random solution.
2. At any iteration, find a new solution by making local movements over the current solution.
3. The next solution selected is the best among the neighbourhood.
4. During exploration, to avoid visited solutions, tabu list is maintained and updated. So, a tabu list maintains solution points that must be avoided.
5. Since the neighbourhood is changing during the course of exploration, it is dynamic unlike the previous searches.
6. A movement of tabu status is avoided for revisiting, unless it satisfies certain aspirational conditions. This overcomes the local optima issue.
7. The algorithm stops when the algorithm as a whole terminates or has a stopping criterion.

A tabu search can have stopping criteria like a number of solutions to be explored or the neighbourhood is empty. Here are the pros and cons of tabu search.

Pros

1. It allows to exit from sub-optimal regions by making non-improving solution to be accepted.
2. The use of tabu list improves efficiency.
3. It can be applied to both discrete and continuous solution spaces.
4. It can address difficult problems (scheduling, quadratic assignment and vehicle routing). Tabu search obtains solutions that often surpass the best solutions previously found by the other approaches.

Cons

1. It has higher dimensions and parameters.
2. Number of iterations could be very large.
3. It cannot find global optimum in some cases.

4.8.5 Genetic Algorithms (GAs)

Genetic algorithms are the computational models, which can be used for searching and problem solving. These models are inspired by the concept of evolution in living organisms. GA builds successive searches or generation of solutions based on the evolution. It typically begins with initial state or hypothesis. The way in which living things evolve and the fittest survives, on similar lines, in GA, the combination and changes in existing states are build to achieve the fitter solutions.

The changes occur in chromosomes during reproduction. The chromosomes from mother and father get exchange randomly by the process of crossover. Hence, the offspring has traits from both father and mother. Even some traits are changed by mutation. There can be a few accidental and random changes also. These accidents, in some cases can produce more beautiful or fitter offspring (not always). Obviously, offspring with better

traits survive for longer duration, while the poor one gets extinct. So, after some period of time, offspring have genes from superior individuals. This process of fittest survival is also called *natural selection*. Figure 4.16 depicts the basic approach.

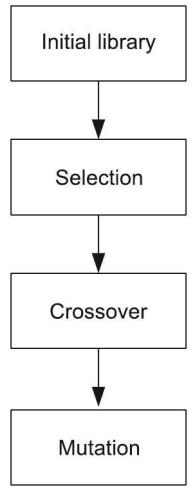


Figure 4.16 Genetic approach.

Simulated Evolution

In simulated evolution, for algorithms to be used, we need to define this process of reproduction more clear with reference to the objects or data we have. Like chromosomes, the structure here can be defined by fixed alphabets or numbers in the form of a string.

Genetic algorithms are very similar to search algorithms. In this case, search is based on natural selection that takes place in natural genetics. The natural evolution process that operates on chromosomes inspired genetic algorithms and hence, GA is developed to simulate it. Figure 4.17 shows the recombination process in genetic.

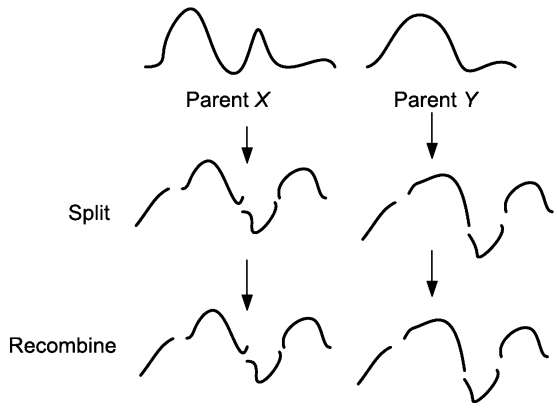


Figure 4.17 Recombination in genetic approach.

General Operations in Genetic Algorithms

1. **Reproduction:** It is the act of reproducing or making an exact copy of a potential solution
2. **Crossover:** It is the act of swapping bit/gene values between two potential solutions, simulating the mating of the two solutions.
3. **Mutation:** It is the act of randomly altering the value of a bit/gene in a potential solution.

Steps of Genetic Algorithm

1. Creation of population takes place.
2. Each member of population is evaluated with reference to fitness function.
3. The best pairs or best candidates are selected.
4. Genetic manipulation and operations are performed to produce new offsprings, and thus a new population too.

The steps are depicted in Figure 4.18.

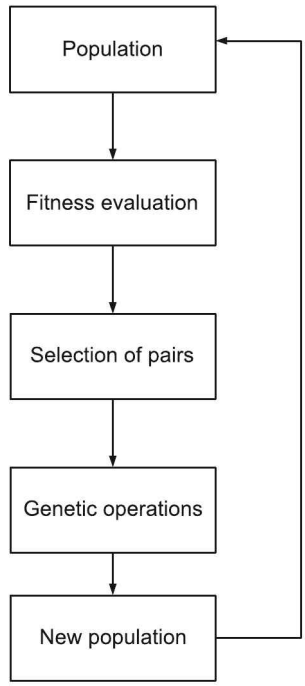


Figure 4.18 Genetic algorithm: Steps.

GA can be thought of heuristic search. Here, heuristic defines the fitness of the state. Rather, it is an adaptive heuristic search. Here the fittest state survives to get an optimal solution. It works on randomised combinations and selection, but guided by heuristic and even exploits historical information. A fitness function drives heuristic and it should return a higher value for a better state.

The selection probabilities of pairs for reproduction are directly proportional to their fitness scores.

While selecting pairs for reproduction

1. Either two pairs are selected for reproduction randomly.
2. The crossover operation recombines bits of selected candidates. There is possibility of single-point crossover as well as double-point crossover.

To solve any problem using genetic approach, a way to encode the solution is required. Generally, it is represented as bit string-chromosome.

To understand if we take simple example of generating a number, say 29, given 0–9 digits and a set of operations + and ×.

1. Represent each given data in gene. For example, let 0–0000, 1–0001 and so on till 9 and the operators be +: 1101, ×: 1010.
2. Now, let us build the chromosome. So, one solution is

	4	×	5	+	9
Represented:	0100	1010	0101	1101	1001

3. New population is generated with crossover and mutation based on fitness function. This fitness function selection can be as

$$1/(x - x')$$

where x is the desired number and x' is the one generated by chromosome.

The approach halts when it is divided by zero error.

4. Decision on selection of fitness value is dependent on the problem to be solved.

Outline of a Genetic Algorithm

1. Generate random population of n candidates.
2. Evaluate fitness of each of the candidate with fitness function $f(x)$.
3. New population is created using the following genetic operations:
   (a) Select two candidates from the population based on the fitness and other criteria.
   (b) With crossover probability from the parents, new offspring are created.
   (c) New offspring is placed in the population.
4. New population is used in the system.
5. If the goal state is reached, stop.
6. Else, go to step 2 and repeat.

As shown in Figure 4.19, the basic outline of the genetic approach is mentioned here. Let us take an example to understand how this crossover and mutation occur.

EXAMPLE: Crossover:
Parent 1: 101 | 11011
Parent 2: 100 | 10010
Child 1: 101 | 10010
Child 2: 100 | 11011

Mutation:
For the child: 10011011 → 10001011 (bit changed)

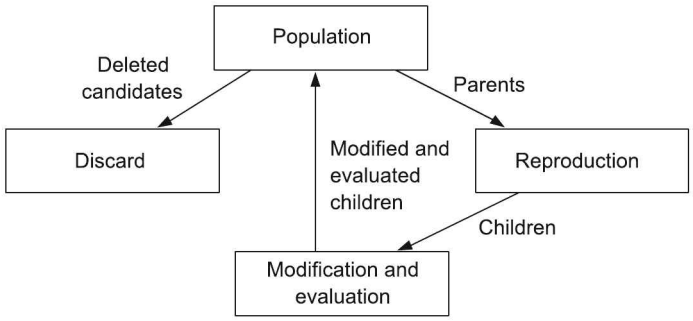


Figure 4.19 Outline: Genetic algorithm.

4.9 GRADIENT METHODS

The rationale behind the gradient methods is an assumption that if one moves in the direction, where the function is increasing most rapidly, and if the step sizes are controlled or rather kept small enough, then it is assured that the function is always being increased. Finally, it will arrive at a point, where it is at maxima and cannot be increased further.

Gradient descent is also known as *steepest descent*. The method states selection of direction with steepest slope from the present state. Let us assume that we have continuous function represented as $f(x_1, x_2, ..., x_n)$ which are to be minimised over X_1, X_2 to X_n.

The process for finding this minimum over $f(x)$ is

1. Calculate the gradients

$$\partial f(x_1, x_2, ..., x_n)/\partial x_i$$

2. Repeat until convergence for all x

$$x_i \leftarrow x_i - \eta \partial f(x_1, x_2, ..., x_n)/\partial x_i$$

(η is the learning rate that has small value, say < 0.1).

Unconstrained Optimisation and Newton's Method

As discussed earlier, the steepest descent uses only first derivative and does not yield the best result always. This is because the first derivative though detects slope, it may be advisable to consider changes in the slope. Hence, Newton's method uses the first as well as the second derivative. It performs very well, if the initial point is close to minima.

Newton–Rapson method can help in some of the problems in the first derivative usage. It helps in solving the linear equations. To find maximum or minimum of f, there is need to find x, where gradiant is zero.

$$p_k = -[\nabla^2_{xx} f(x_k)]^{-1} \nabla_x f(x_k) \tag{4.1}$$

Hessian matrix of second derivative can be used. Obviously substituting the identity matrix uses no real information from the Hessian matrix. The projection could then be derived as follows:

$$p_k = -H_k^{-1} \nabla_x f(x_k) \tag{4.2}$$

4.10 FUZZY ADAPTIVE SEARCH

Genetic algorithms have been applied to many types of optimisation problems by encoding the design variables. Genetic approaches are not always optimal considering their parameters. They also suffer from the iterations and the problem of having premature convergence. Lot of work has been undertaken to overcome these issues, where some have proposed use of dynamic control of genetic algorithms using fuzzy logic. An *adaptive fuzzy search* is the one, where the search is made efficient by the use of fuzzy rules to tune in the genetic parameters. Fuzzy adaptive techniques are most often used in parallel genetic algorithms. Average fitness and the difference between the maximum and average fitness are used while designing the fuzzy rules. Due to this, it has a quick ability to obtain a better result compared to genetic algorithms. The key aspect of this type of search is that the parameter tuning is done not only on the crossover or the mutation rate but also on the migration rate.

4.11 ADVERSARIAL SEARCH METHODS (GAME THEORY)

The searches required for game playing are bit different. Here, in this section, the typical searches required for game playing between two players, are discussed. The state space in this case can be represented as a tree. Games may give either perfect or imperfect information based on the type of game. The games generally have multi-agent environment. Further, there are different types of unpredictability, which introduce other contingencies. The environment is co-operative and competitive. Algorithms like minimax (MinMax or MM) algorithm and alpha-beta pruning are discussed under this category.

4.11.1 Minimax (MinMax or MM) Algorithm

Minimax algorithm while trying to cover the entire search space considers exhaustive possibility of state transition from the initial state. The minimax algorithm can be used in case of two-player games such as tic-tac-toe, chess, go, etc. The algorithm thus is effective for the games, which have few logical possible state transitions from the current state. Typically, these are the logic games, hence can be described by a set of rules and premises. So, it is important to know the possible moves from the present state.

A pseudocode for minimax is described below:

Minimum (node n, depth d, player p)

1. If depth = 0 then
   return value (node)
2. If player = 'MAX' //for a maximizing player
   set $\alpha = -\infty$
   for every child of node
   value = minimax (child, depth–1, 'MIN')
   α = max(α, value)
   return (α)

```
else    //for minimizing player
set α = + ∞
for every child of node
  value = minimax (child, depth-1, 'MAX')
  α = min (α, value)
return (α)
A maximising player would call it as minimax (start, depth, MAX).
```

As described above, two players are involved in this case, MAX and MIN. First starting with the current game position, a search tree is generated. That means the entire search space is expanded. Depth first can be used up to the end game position. There are two views—MIN view and MAX view. The end position is evaluated from the MAX view. The values are assigned based on the evaluation. The nodes that belong to the MAX are given the maximum value of its children. The nodes for the MIN are given the minimum value of its children. In short, if we consider Min and Max as players, MAX tries to move to a state of maximum value, while MIN tries to reach to a state of minimum value.

Let us take an example of carom game. We have situation analyser that converts the judgement about the overall positions of coins on the board into a single representative number. Let us say positive number indicates the favour to one player, while negative number indicates the favour to another player. *Static evaluation* is the process that reflects the board quality, while *static evaluator* is the procedure to determine this number. The number is evaluation score. Here, the maximiser tries to force player to move that leads to large positive score, while the minimiser tries to force to a move that leads to strong negative score. The minimax levels are depicted in Figure 4.20.

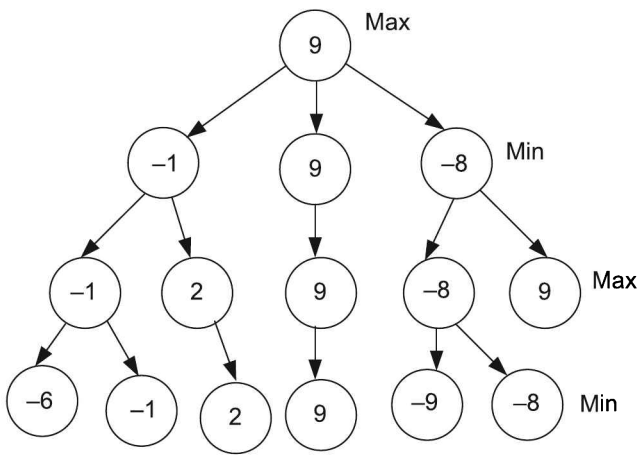


Figure 4.20 A search tree: Minimax.

There are alternate minimising and maximising levels. Eventually, the exploration limit is reached and the state evaluator helps in selecting a candidate among alternatives.

4.11.2 Optimal Decision States for Games

Generating trees for simple and not so complex games is not so time-consuming, but it is not the case when the complexity of the game increases. Hence, handling of such games demands optimisation.

One approach for optimisation could be limiting the depth of the tree. So, instead of having the entire tree generation, it is restricted by the depth. The dependency of this optimisation is on the branching factor. This results in reducing tree expansion, and hence, contributes in time optimisation. In another approach, a function for the evaluation of the game positions is used that can make use of heuristic.

4.11.3 Alpha-Beta Pruning

The minimax is an exhaustive search algorithm. It requires two-pass analysis of the search space. There can be many levels. But a question arises here—how many levels we should consider?

Minimax pursues all the branches in search space, including many branches which can be ignored or guaranteed not to lead to results. There is need for a procedure that can reduce the number of tree branches and the number of evaluations that must be done. Alpha-beta pruning uses different ideas to reduce the number of nodes we search and the overall search space. *Pruning* refers to elimination of nodes found to be unnecessary while searching and evaluation. In case, we have reached a scenario, where minimising or maximising has guaranteed certain values, and hence, pursuing certain node definitely would not take to a optimal result, then it is better to eliminate that node or path.

Note: Do not follow the idea that is surely bad.

Alpha cut-off [Refer diagram 4.21(a)],

For Max node:

1. Let us assume that we have explored branch P. The value obtained is 9. This is best value for A.
2. While exploring branch Q, at branch Q_1 we get val as 7. Since $7 < 9$, other branches, i.e., Q_2, Q_3 and Q_4 are not evaluated.
   Hence an alpha-cut takes place.
3. Thus, the best possible value at branch Q must be ≤ 9.
4. So, finally the best value obtained from branch P is returned.

Beta cut-off [Refer diagram 4.21(b)],

For Min node:

1. Let us assume that we have explored branch P_1 and obtained value 9 from it for P (i.e., ≤ 9).
2. While exploring P_2, we obtain a value of 20, where $20 > 9$.
3. Thus, other branches are not explored and a beta, cut-off takes place.
4. As the best possible value at P_2 is 20 which is greater than the best value of 9 obtained earlier. So, the value obtained from P_1 is returned.

This process for MIN and MAX is depicted in Figure 4.21 below:

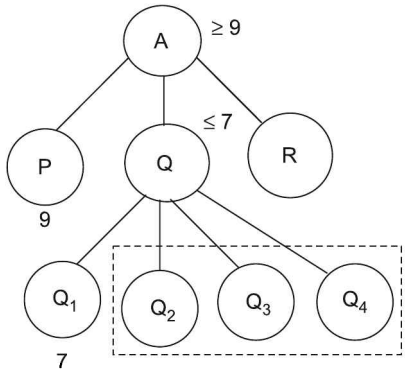


Figure 4.21(a) Alpha-cut.

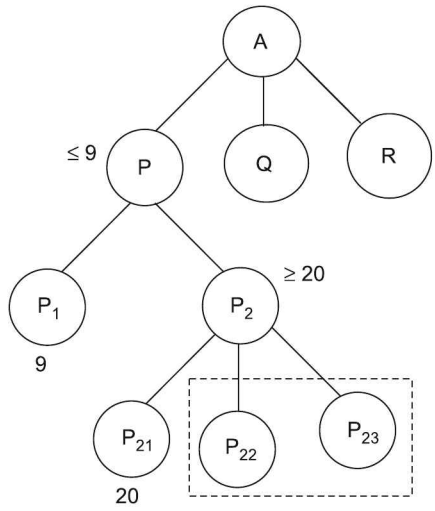


Figure 4.21(b) Beta-cut.

α is the value of the maximum/best (i.e., the highest value) choice found so far at any choice point along the path for MAX. If any value say v is worse than α, MAX will avoid it. β is the value of the minimum/worst (i.e., the lowest value) choice found so far at any choice point along the path for MIN. If any value say v is not worse than β, MIN will avoid it.

The algorithm is similar to that of the minimax, the difference being alpha and beta values are added here. The algorithm is discussed below:

Function alpha-beta (node n, depth d, α, β, player p)

1. If depth = 0
   return value (node)
2. If player = 'MAX'
   val = $-\infty$
   for every child of the node
   val = max (val, alpha-beta (child, depth-1, α, β, MIN))
   α = max (α, val)
   if $\beta \leq \alpha$ then break //β cut-off
   return (val)

   else
   val = ∞
   for every child of the node
   val = min (val, alpha-beta (child, depth-1, α, β, MAX))
   β = min (β, val)
   if $\beta \leq \alpha$ then break //α cut-off
   return (val)

Alpha-beta cut-offs speed up the process. Using them with a good evaluation functions, makes the search much faster.

4.11.4 Refinements and Variations

In recent years, there have been many contributions towards the refinements in the alpha-beta algorithm. The main constraints were speed and efficiency. They are based on some principles such as, minimal window search, forward pruning, move ordering and so on.

Typically, you must have noticed that narrower the search window, more is the possibility that the cut-off occurs. A search window with, say $\alpha = \beta - 1$ is called a *minimal window*. As it is narrow, it is believed that using this would yield more efficiency.

Similarly, the efficiency also depends on the move of search order. For better results, it is necessary that the best move is examined at prior. Here, many ordering techniques like iterative deepening, heuristic history are used.

In case of forward pruning, the unpromising branches are discarded. This reduces the tree size. The major drawback is there is a possibility that the best move might get pruned.

4.12 ONLINE SEARCH ALGORITHMS

The offline search algorithms discussed may not able to handle the real-world scenario. In online searches and online agents, environment is observed after every action and next action is taken. This can help in handling problems in dynamic and semi-dynamic environment, and even in the case of stochastic domains and environment. Online search are typically based on exploration. In case of exploration problems, states and actions are unknown to the agent. Hence, own actions and the outcome are used to determine the next action. The typical examples where online search is required are driving a car by a robot, auto pizza delivery, robot walking on uneven surface, automatic basketball trainer. An agent knows just actions, step cost function and the goal test.

Online search problem can be solved by the agents by considering following points:

1. After each action, the agent senses environment parameter. As a result, it receives a percept telling it what state has reached.
2. An agent can build/update its map from the environment.
3. Expansion of the nodes is different for online and offline agents(search).
4. An online agent can only expand the node it is physically in.

Typically, the objective of an agent is to reach the goal state in the minimum possible cost. It can be the total path cost that is selected by the agent and travelled to reach the goal state. If the agent knows the search space in advance, then the actual shortest path can be compared with the actual cost incurred. This is also referred to as *competitive ratio*. Online DFS agent is depicted below:

```
Online_Dfs(S)
{
   If(goal(S)) then return
   If(new state) then
      Unexplored[S] = Actions(S) // set of actions
   If(!empty(S)) then
      Determine next state with respect to action
   If state is earlier explored, take similar action
   Return action
}
```

4.13 SEARCHING ALGORITHM: EXAMPLE AND APPLICATIONS

We have studied that search techniques are required for every walk of life. The problems can be of simple type, typically belonging to route finding or even game solving. It is very essential to understand the application and determine the search technique to be used. Pattern-seeking problems have important application in production planning. These can be NP-hard. The use of heuristic is proved to be very appropriate in these cases. Neighbourhood evaluation search approaches have shown substantial progress in the same.

Heuristic finds a place in decision-making approaches as well as in classification techniques. It finds place in game playing as well as in web crawlers. Text classification techniques too employ search techniques.

Let us take an example of agriculture sector, where you are to design a tool to enable a farmer to select the best crop suitable for his land. With the various parameters with regard to the land and environmental conditions, there can be different heuristic methods applied to the selection of the crop, which give final decision. Based on the earlier experiences of the crops, heuristic can be appropriately estimated and such a search method can be very much useful in assisting the farmer.

On similar grounds, let us take another example of interior designing of a room. In a hall with various objects to be placed so that it looks beautiful, an algorithm like A* would possibly give good results, considering the space occupancy of the objects to be used. It can be applied with the number of objects to be placed, considering their size, shape, dimension, weight and so on. Even an approach for appropriate colour selection from the list of available furniture can make architects and interior decorators happy by making their work easy.

The list of applications of search techniques is endless. Right from searching solution for any problem in examination to search of any destination, there are numerous problems in day-to-day life where search techniques can be used.

SUMMARY

In the chapter, we have discussed in detail the various informed search methods. These methods are used, where there is a prior domain knowledge about the problems and guidelines are provided. A heuristic function is used to guide the entire process. Estimating heuristic is the most critical aspect in these search methods. It can be based on the past experience too. We have discussed heuristic-based methods such as best first search, A* and its variants.

While selecting a search technique, one needs to keep in mind the search space to be traversed. Local search techniques that are based on local view have also been discussed in the chapter. Hill climbing, simulated annealing and other approaches that are the parts of local search methods have given a broader idea of the different search methods. Approaches to game playing are somewhat different. The minimax algorithm is discussed in this context. For better optimisation, eliminating unwanted paths that are carried out in alpha-beta pruning have also been studied.

An introduction to online algorithms and examples in terms of applications have been discussed. Finally, one has to look at the various aspects with a reasoning of 'why', one particular search is the best for the problem at hand.

KEYWORDS

1. **Heuristic search:** It is a technique that improves the efficiency of search technique by providing guidelines.
2. **Heuristic function:** A heuristic function is the one that guides the decision of selection of a path.
3. **Best first search:** It is a search method that exploits DFS and BFS, switching between both to get the benefits of both using heuristic.
4. **OR graph:** It is used to avoid revisiting of paths and also for propagation to the successor. It maintains open and closed lists.
5. **Admissible:** Any search algorithm is admissible if it always produces an optimal solution.
6. **Consistent heuristic:** A heuristic $h(n)$ is said to be consistent if for every node, say n and every successor s that is generated by an action a for n, the estimated cost to reach the goal state/node is not greater that the step-wise cost to reach s added to the cost of reaching the goal from s.
7. **A*:** It is a heuristic approach that performs search to compute optimal solutions.
8. **Memory-bounded heuristic:** It is used to overcome the memory requirements of simple heuristic search. IDA*, RBFS and MA* are under this category.
9. **IDA*:** It is a memory-bounded heuristic method that employs DFS at each iteration. It maintains the *f*-value and applies threshold.
10. **RBFS:** It performs best first search by keeping track of the best path and backtracks when the cost exceeds.
11. **Relaxed problem:** A problem for which rules/criteria are simplified or less restricted is a relaxed problem.
12. **Admissible heuristic:** A heuristic is said to be admissible if it does not overestimate the costs.
13. **AO*:** It is a heuristic approach that maintains a graph rather than open and closed lists and performs search.
14. **Local search:** It is a search that begins with initial solutions and iterates ahead exploring the search space.
15. **Hill climbing:** It uses the notion of heading towards a state, which is better than the current state and moves to the neighbour with a better score.
16. **Local maximum:** This is a state better than the local region or neighbouring states, but not a global maximum, since a better solution exists, which is not in the vicinity of the present state.
17. **Plateau:** This refers to a flat area or space where neighbourhood states have the same value as the present state, and hence, fails to determine the best direction to move on.

18. **Ridge:** It is the search space at higher altitude than the surrounding that cannot be traversed by a single move. It is a special kind of local maxima.
19. **Stochastic hill climbing:** It is a variant of hill climbing that selects at random among all possible uphill moves.
20. **First choice hill climbing:** It is a variant of stochastic hill climbing where the successor is generated randomly until the one generated is better than the current state.
21. **Random restart hill climbing:** It conducts a series of hill climbing searches from randomly generated initial states. Each run is like an independent hill climbing algorithm.
22. **Evolutionary hill climbing:** It is a genetic algorithm-based search that performs random mutations and keeps the ones that are better.
23. **Simulated annealing:** It is an approach used for generating optimal solutions avoiding local optimas.
24. **Beam search:** It explores states or graph by expanding the most promising nodes in a limited set. At any level, it expands identified best node.
25. **Stochastic beam search:** It is a variant of local beam search and is very similar to stochastic hill climbing. Instead of selecting K successor which are the best ones, stochastic beam search selects K successors at random.
26. **Tabu search:** It is a search that tries non-improving move when gets stuck in suboptimal region. It maintains a tabu list.
27. **Genetic algorithms:** Genetic algorithms are the computational models, based on biological evolution that are used to solve optimisation problems.
28. **Reproduction:** It is the act of reproducing or making an exact copy of a potential solution that is done in genetic approach.
29. **Crossover:** It is the act of swapping bit/gene values between two potential solutions, simulating the mating of the two solutions that is used in genetic approach.
30. **Mutation:** It is the act of randomly altering the value of a bit/gene in a potential solution that is used in genetic approach.
31. **Adversarial search methods:** These are the methods used in game playing, where the environments are competitive.
32. **Minimax algorithm:** A strategy used in game playing to minimize the loss in worst cases.
33. **Alpha-beta pruning:** An approach that reduces the number of nodes evaluated by the minimax approach.
34. **Online search:** It is an algorithm, where the environment is observed and the next action is taken in real-world scenario.

MULTIPLE CHOICE QUESTIONS

1. IDA* employs
   (a) BFS at each iteration (b) Combination of BFS and DFS
   (c) Only DFS (d) None of these
2. A search algorithm is said to be admissible, if
   (a) It is informed search (b) It is optimal
   (c) It is uninformed search (d) It is efficient
3. A space where neighbourhood states have the same value as the present state and causes a problem in the hill climbing is
   (a) Plateau (b) Global maxima (c) Ridges (d) Local maxima
4. Random alteration of the values in genetic algorithm is done in
   (a) Crossover (b) Mutation (c) Fitness function (d) Both (b) and (c)
5. The search methods that work on one state, with an aim to improve it step-wise belong to the category of
   (a) Best first search (b) Depth first search
   (c) AO* (d) Local search methods
6. If heuristic is admissible, then A* guarantees that it will be
   (a) Complete (b) Optimal
   (c) Both (a) and (b) (d) None of the above
7. The algorithm that tries to resolve the issue of local minima with random moves is
   (a) A* (b) Hill climbing
   (c) Simulated annealing (d) Beam search
8. A search that tries a non-improving move when it gets stuck maintaining a list is
   (a) A* (b) Hill climbing (c) Tabu (d) Greedy search
9. In minimax approach, the values for α, β are
   (a) $\alpha = \beta - 1$ (b) α = max, β = min
   (c) α = min, β = max (d) $\beta = \alpha - 1$
10. A case where $f(n) = h(n)$ will be found is of
    (a) Simulated annealing (b) Hill climbing
    (c) Tabu (d) Greedy best first

CONCEPT REVIEW QUESTIONS

1. What is a heuristic function?
2. Distinguish between informed and uninformed search.
3. Explain the A* algorithm and conditions of optimality.
4. Explain the approach of alpha-beta pruning.
5. How does RBFS perform memory-bound search? Discuss.

CRITICAL THINKING EXERCISE

1. During the selection of the temperature values, which parameters should be taken into account while performing simulated annealing?
2. Can alpha-beta pruning approach fail to overcome minimax limitations? Discuss.
3. For a given problem, which parameters would you take into consideration and select a search strategy?

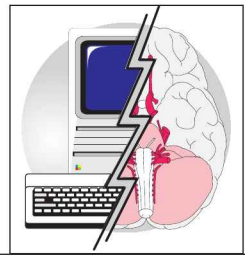
PROJECT WORK

1. Implement a system that performs arrangement of some set of objects in a room. Assume that you have only 5 rectangular, 4 square-shaped objects. Use A* approach for the placement of the objects in room for efficient space utilisation. Assume suitable heuristic, and dimensions of objects and rooms.
2. Implement hill climbing search to solve the following Sudoku puzzle given in Figure 4.25. Assume heuristic to be the sum of the conflicts that are identified.

1			4
	3	2	

Figure 4.25

CHAPTER 7



Knowledge and Reasoning

Learning Objectives

- To understand the importance of knowledge and its approaches
- To study the various representations of knowledge
- To study the role of logic in representation
- To understand the use of predicate and first order logic
- To study the inference and reasoning process
- To appreciate the relationship between knowledge and reasoning

INTRODUCTION

Search techniques and intelligent agents acquire information from environment and further build the knowledge with reference to problem at hand. It is this knowledge that is exploited further with the actions and decisions. Thus, appropriate and precise representation of the knowledge becomes a critical factor in the process. With the previous study on the search techniques and the applications, the agent environment, and the problem-solving issues, we now turn towards the knowledge representation. For all the methods that have been discussed previously, we need to look at the most important aspect of how the knowledge can be represented so that it can be used effectively and applied to the process. In turn, we can say that there is a reasoning process that actually is making the use of the knowledge.

But then, a question arises what is 'knowledge' basically? It is some set of patterns and associations derived from data or information that helps in making decisions and resolves problems that may be related to any day-to-day life or some complex problems. Consider a simple example, where a teacher has to judge the performance of a student for some exam. The teacher judges on the basis of percentile the student would obtain.

The judgement could be based on the previous performances of that student or on some information given by some other teachers about that student. This can be considered as the available knowledge. So, one can arrive at some decisions based on this knowledge.

A systematic reasoning process is required when we try to relate the events to the outcomes or to arrive at judgements. Hence, *reasoning* is the way we conclude on different aspects of problems based on the available knowledge representation. Logic plays an important role in the reasoning process. So, *logic* is the one that makes the knowledge representative. In the course of representing the knowledge and utilising it, the chapter focuses on the various aspects that are essential from appropriate representations to exploit the knowledge base (KB). Though there are restrictions with the knowledge base handled here, it will be a stepping stone for us to start with the representations and understanding its use, where the knowledge is certain.

Let us begin our discussion with the knowledge representation, various issues and other aspects related to knowledge representation along with the agent environment.

7.1 KNOWLEDGE REPRESENTATION

In introduction, we have mentioned that knowledge is an important aspect of the reasoning process. Our outcomes govern the way we have the knowledge and the way we update it. Mapping it in technical terms, depending on the domain one is working on, it is very crucial to identify and create representation of the knowledge. So, can we say that knowledge representation is about representation of the facts at hand? The answer truly lies in the fact, viz. which facts at hand can be represented or to be specific, which can be manipulated. This is required from the viewpoint of specific knowledge representation that gives a broader view for problem solving. Let us proceed with the approaches and issues of knowledge representation.

7.1.1 Approaches and Issues of Knowledge Representation

Consider a case where fruits are to be arranged in a basket. This would be a simple task at present. One can easily have a knowledge representation and act accordingly. There could be multiple options that will give the desired outcome. But with the addition of more facts like the size of the fruits, the quantity, the basket size and so on, it would further narrow down the arrangements. Further, it is quiet obvious that some information gets available over a period of time. The KB representation should be able to handle this sort of environment, where limited or partial information is available in the beginning and later on, more information is evolved. We cannot say that for a particular problem, there is only a specific way of knowledge representation. But the representation definitely counts while coming to reasoning. Figure 7.1 depicts the process of deriving facts. From the diagram, it is clear that raw data, domain knowledge and, percept allow to refine internal representation iteratively to arrive at the final facts. Here, reasoning is required to establish relationships among the available data and the final facts.

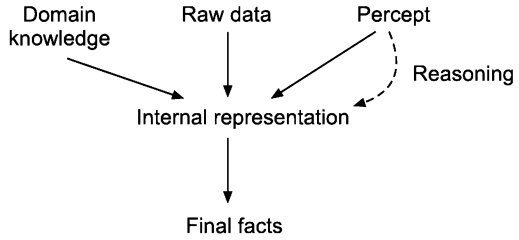


Figure 7.1 Knowledge building and representation.

Approaches

Before we start our discussion on the approaches, let us have a quick look on the properties that are required for knowledge representation in the system design. While designing a system for knowledge representation, we would always go for a system that allows representation of the entire knowledge, which is required with respect to the application or domain we work on. This is precisely the property that speaks about adequacy in terms of representation. The next property talks about adequacy in terms of inferring. Here, it is necessary that the knowledge should be represented in such a way so that there is a way to manipulate the representative data in order to derive at new ones. At the same time, it needs to be efficient in terms of inferring, where the additional data should be used in the direction of better inferring. Further, we would like to have the representation to learn; it should be adaptive and able to accommodate new additional information that would be available at any point of time. This is the property of efficiency in term of acquisition. Here, we mean to say that it should posses the property of being incremental. This could also be a simple knowledge update by the user himself.

During the course of representation, it is worth to mention that even though we expect it to possess the properties, it is not feasible that just one representation is able to fulfil them. Generally, different approaches are in use when it comes to representation of knowledge, even though it is for a specific domain.

Let us start with the basic and the simplest ways for representation. Now, when we are discussing about the representation, the obvious thing that comes to our mind is the database or files. Are they a part of knowledge representation? Are there some other methods too? Definitely, they are a part of representation. Let us understand what approaches can exist along with them with respect to the structure.

We begin with a simple *relational knowledge structure*, wherein we have the database representation. The facts can be mapped into the relations and stored in the database. This kind of representation without any additional procedure to get something out of it is a weak inference mechanism.

Table 7.1 represents a simple example of relational knowledge structure.

TABLE 7.1 Relational Knowledge Structure

Employee	*Salary*	*Experience*
Sameer	30000	3
Kavita	20000	2
Jasmin	20000	2

The next structure is *inheritable knowledge structure*. This type of representation is required as all the knowledge related to the inheritance is not mapped in the earlier case. This is critical while drawing conclusions. Hence, we need a structure that can help in inferring. So, a general hierarchy structure is used, where it is possible to have a proper mapping with inheritance. Figure 7.2 shows this type of knowledge structure.

Let us take the example of cricket. The knowledge structure for any player is based on various parameters or attributes. The player can be a batsman or a bowler. He can be a right-handed player, with some specific height. So, there exists an ISA relationship among a person, a player and a batsman or a bowler. Similarly, we can have an instance to represent the knowledge. The structure is also called *slot-filler structure*.

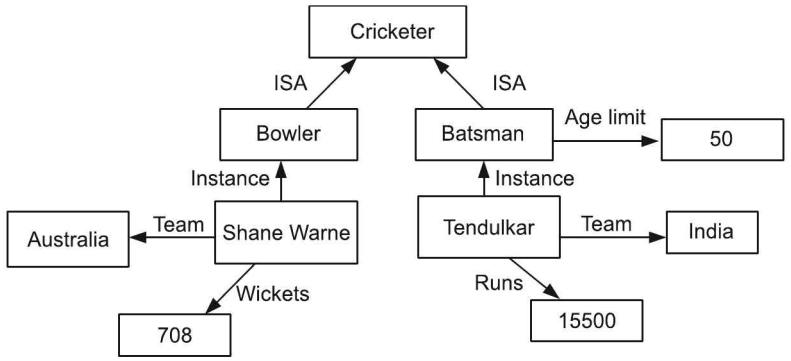


Figure 7.2 Inheritable knowledge.

Semantic network and frame collection are also related to the structure, which are discussed later in the chapter.

We are aware that finally our intention is to come out with the outcomes that will answer our queries. Now, for example, we say that we want to have a solution to a query like does Tendulkar satisfy age limit? (The diagram represents hypothetical data to understand the hierarchy). The process of seeking a solution is

1. To check if direct answer is available. In case, it is not, go to the next step
2. To check for attribute instance

Here, it will be under the category of Batsman. Had it been the case that the query was regarding the parameter like related to cricketer, we would have to move a step up in the hierarchy and got the answer. This process or the approach applies the property of inheritance. This approach actually guides the utilisation of the knowledge.

Another structure is *inferential knowledge structure*. In this type of structure, first order predicate logic is used. So, a typical example is combining the knowledge to get the outcomes. (Details of predicate logic are discussed later in the chapter.) One point to mention here is that there is a need for the inference procedure to have the utilisation of knowledge.

Procedural knowledge structure is another approach for the representation. This structure comes into picture when we need to have the knowledge in detailed form. Somewhere, it specifies steps to be followed and its details. Programming languages are

used for that purpose. LISP is the most common language that finds a place in this case. Another representative structure could be in the form of rules—production rules.

We have now discussed the introduction to structures. As we will proceed through the chapter, we will have a clear picture of the other details about the knowledge representation and use of logic for the same. Let us turn towards the issues of knowledge representation.

Issues of Knowledge Representation

There are various issues of knowledge representation related to information mapping and use of structures. Most obvious questions that arise in knowledge representation are as follows:

1. Which information can be mapped into knowledge?
2. How can it be mapped?
3. How to decide which would be an ideal mapping that will give the most accurate solution to the reasoning process?
4. Is there a way that will help in better representation?
5. What would be the memory constraints?
6. Is it possible to have an access to the relevant part of the knowledge? There could be many more questions also.

Since we have already studied representative structures, we will focus on the issues related to knowledge representation and methods to resolve the same.

1. *Attributes* are the most important ones that have an impact on the representation. It is required to understand and identify the important attributes. This helps in absorbing the important parameters for the KB representation.

2. Similarly, identifying the *relationships among the attributes* in the representation is equally important. By pairing of the attributes or with ISA methods, the relationships are captured. Reasoning about values that are taken up also adds up complexities in the selection process. Hence, proper selection of the attributes impacts the relationships.

3. Handling the issue of level or upto what depth the mapping of the knowledge is to be done defines the *granularity*. This is governed by the availability of facts and the level upto which it is possible to split them and represent them. Sometimes, splitting can prove to be an option to handle the issue, making the easy accessibility, but sometimes, it adds complexities to handle the data.

4. Further, the issue is *representation of the objects as sets*. Use of logic and the inheritance that we have already studied are very well-suited to tackle this issue. Sometimes, a particular property that is applicable to the entire set has an exception. There is a need to clarify in terms of properties before we have the representations. The name that represents the set has to be precise, as this impacts the object selection, letting us know about its membership. Here, extensional definition is used, where the members are listed in the set, and the other is intensional, where rules are provided that determine the belongingness of the object. This tries to restrict the representation of the objects.

5. Finally, *selection of correct structure* is the most important part to have a proper representation. Things, here, as made clear earlier too, are dependent on the domain. Here, the issues range from how to fill in the details to when to have a new structure. Though selecting a knowledge structure that matches a particular problem is very difficult, still

to mention a few, one method can be indexing the structure with significant keywords. This can with regard to English, but then this is specific when the domain description is available in English. Other approach could be the use of pointers. Using this notion, it is possible to have intersections of the sets and to use structure that are precise. One more option could be locating and selecting a major clue in the problem. This helps in defining the structure. It is also equally important that the structure is flexible, which allows revisions as per the requirements. Getting appropriate values is the final requirement. In the sense, whenever we have structure selection, its exploitation is done by getting the results. Generally, a candidate structure is set up and then applied to the problem. But, if the outcomes are not accurate for the problem, then it is the time to change/revise the structure. This can be in terms of attribute values too.

Figure 7.3 represents the issues and their inter-relationships.

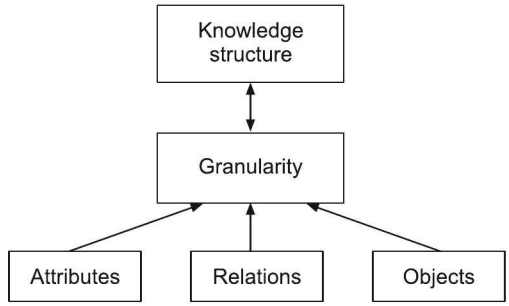


Figure 7.3 Knowledge structure hierarchy.

7.2 KNOWLEDGE-BASED AGENTS

We have already studied about intelligent agent. An *agent* is the one who acts according to the environment. To act, it needs to know the knowledge of how to act. A knowledge base (KB), hence, plays an important role in deciding the actions. This knowledge base is nothing but a representation of the information that helps an agent to act. Knowledge representation language makes use of sentences to represent facts about the world.

An agent's operating environment is based on perception and action. Depending on the current percept, it acts. So, this percept should be available with the knowledge base. Then, the action occurs. This action too needs to be updated to the knowledge base. We can say that the knowledge-based agent performs three-fold task. A typical algorithm is summarised below:

Knowledge-based agent
Input: Percept
Returns: Action
  Give-info (KB, make-percept-sentence (p, t))
  Action ← (KB, make-query(t))
  Give-info (KB, make-action-sentence (a, t))

where p is the current percept, t is the time and a is the action.

From the steps, it is clear that in the initial *give-info* function, it records the current percept *p* and reveals the information about the percept in the form of knowledge. This can be in the form of a sentence. In the next step, the action *a* is returned that is the result of formulation of query for the knowledge that has been received at the previous stage. The last stage is where the KB is updated about the action taken.

The basic difference between a normal agent and a knowledge-based agent is that the actions of knowledge-based agent are not arbitrary. They are dependent on the knowledge. They are described at three levels. At the knowledge level, the agent is provided with the information it should know and its targets. At the logical level knowledge is represented in logical language, while at implementation level, logical sentences are implemented. The implementation level does not hamper any of the knowledge level details.

Knowledge-based agent accepts new tasks through defined goals. They acquire knowledge through learning and flexibility adapts to situations. Before the agent starts its operation, the initial knowledge can be made available in two ways: (i) declarative and (ii) procedural.

Declarative knowledge is embedded in the system in the form of pre-defined rules, while procedural infers about the action with reference to situations.

7.3 THE WUMPUS WORLD

This is a very traditional environment to understand an intelligent agent-based system. The Wumpus World is basically a cave that has some rooms connected to each other by passways. In one of the rooms lies a wumpus (a beast) that eats anyone who enters the room. The wumpus can be shot by an agent, but the agent has only one arrow available. In some of the rooms, there are pits too. The best part of this world is that the agent might get a heap of gold. The PEAS description for this environment is as follows:

P—Performance: (i) 1000 points when gold is found
(ii) –1000 points when falls in pit
(iii) –1 for every move
(iv) –10 when arrow is used

E—Environment: A grid of 4*4, with pits at some squares and gold at one square and agent position at [1, 1] facing right.

A—Actuators: (i) Turn 90° left/right
(ii) Walk one square forward
(iii) Grab or take an object
(iv) Shoot the arrow (agent has one arrow)

S—Sensors: There are five sensors. They capture the following:

1. In rooms adjacent to room of wumpus (excluding diagonal), the agent perceives stench.
2. In the square adjacent to pit (excluding diagonal), agent the perceives breeze.
3. In the room containing gold, the agent perceives glitter.
4. When agent walks in a wall, he perceives a bump.
5. When wumpus is killed, it screams that can be perceived anywhere in the environment.

Figure 7.4 shows the Wumpus World. (Different positions can exist for the wumpus, pits and gold and it is not the case that the room arrangement is same).

The agent in the Wumpus World draws conclusions based of the facts. If the facts are correct, naturally the conclusions will be correct and hence, its actions too. Let us understand the actions of agent in the environment.

The knowledge base of the agent contains five initial conditions that are listed in the sensors. It starts with [1,1]. It knows that this is the position, where it is safe. It needs to move ahead either to reach the room of gold or to be safe. Since at [1,1], it does not perceive a stench or breeze, the adjacent rooms are safe as concluded based on the facts. So, it can move to [2,1] or [1,2]. Let us say it goes to [2,1]. At this point, it gets a breeze. So, it concludes that at [2,2], there could be a pit or at [3,1], there could be a pit.

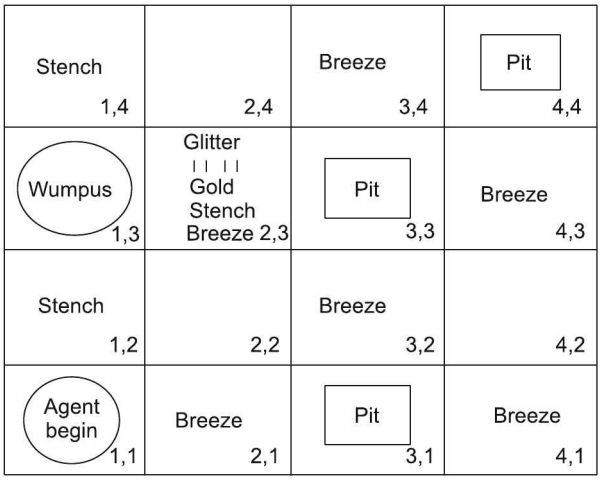


Figure 7.4 The Wumpus World.

So, it moves back. It now goes to [1,2]. It perceives a stench. This indicates the there is wumpus. The wumpus could be at [1,3] or [2,2]. But [2,2] is the position, where the agent assumes that there could be a pit. Since it does not receive a breeze in [1,2], it concludes that pit is absent in [2,2]. But then what about the wumpus? If the wumpus was present in [2,2], it would detect a stench at [2,1]. Hence, it concludes that [2,2] is safe and the wumpus is present at [1,3]. Concluding this is difficult without previous experiences. From here, the agent could go to [2,3] or [3,2]. If it selects [2,3], he will perceive glitter!

Hence, it is the reasoning that leads the agent to take correct actions and this is dependent on the correctness of the facts. So, we conclude that logical reasoning is the essence to have correct conclusions.

7.4 LOGIC

What are we actually doing by studying logic? Are we trying to understand the process of reasoning? We already know that logic basically deals with the study of principles of reasoning. So, what we are trying to put forth in this section is how logic is built, or rather how the logical representations help in the process of decisions.

So, can we say that there are syntax and semantics that are required to be handled in logical representations? Yes. Logic involves syntax, semantics as well as inference procedure. Speaking about the syntax, we know that there can be a variety of ways to represent it. It is not a concern, actually, but it is the way an agent (relating it to the Wumpus World's agent) builds the base so that the reasoning plays an important part. With respect to semantics, though it deals with the meaning, in sentential form, it can either be true or false. There is a need to define logical involvement in two sentences or facts. So, here, we are trying to model it. It can be put forth in the following way: Assume that x and y are the two sentences whose relationship is to be determined. Let us map this relationship as follows:

$$x \models y$$

This indicates that the sentence x entails y, indicating that in the model when x is true, y is also true. For example, suppose

$$\text{KB} = \{p=1\}$$

$$\text{Then, KB} \models \{p + 1 = 2\}$$

The other way round, it states that y is contained in x. In the Wumpus World, when the percepts are combined with the facts or the rules, then the combination constitutes the Knowledge base (KB). Considering the example of Wumpus World, where the agent wants to infer whether a pit exists in the rooms, say [1,2], [2,2] and [3,1] with available information that breeze exists at [2,1] and at [1,1], it experiences nothing, then there are 2^3 possible combinations. With the available facts, the KB is definitely false in the case, where the agent wants to infer about the pit in [1,2]. This is because it does not experience any breeze in [1,1]. To infer, two cases are considered—1. Pit does not exist in [1,2] and 2. Pit does not exist in [2,2]. From the possible models shown in Figure 7.5, it can be inferred that the KB $\models$ case 1. So, when every KB is true, case 1 will be true. But it cannot be judged as to whether the pit exists in [2,2]. This type of inferring is called *model checking*.

So, in logical inferring, there is a notion of truth that is to be maintained. Even it needs to have the property of completeness. Finally, the last word is if the knowledge base is true, then the sentence it derives has to be true.

7.5 PROPOSITIONAL LOGIC

After having an overview of logic part, we begin with the propositional logic; the most simple logic. Why are we studying it? The answer is—for reasoning. It is a mathematical model that provides reasoning regarding the logical value of an expression.

What is propositional logic? It is a logic that is concerned with the propositions and their relationships. Propositional logic is also called *sentential logic*. Propositions are sentences—declarative sentences, say 'Ice is cold' or 'The door is closed', and so on. But 'Is it cold?' or 'Open the door' are not propositions. Hence, they cannot be an explicit order. A declarative sentence states that it can be either true or false, but not both. Propositional logic is the fundamental logic. Let us understand the syntax and semantics first.

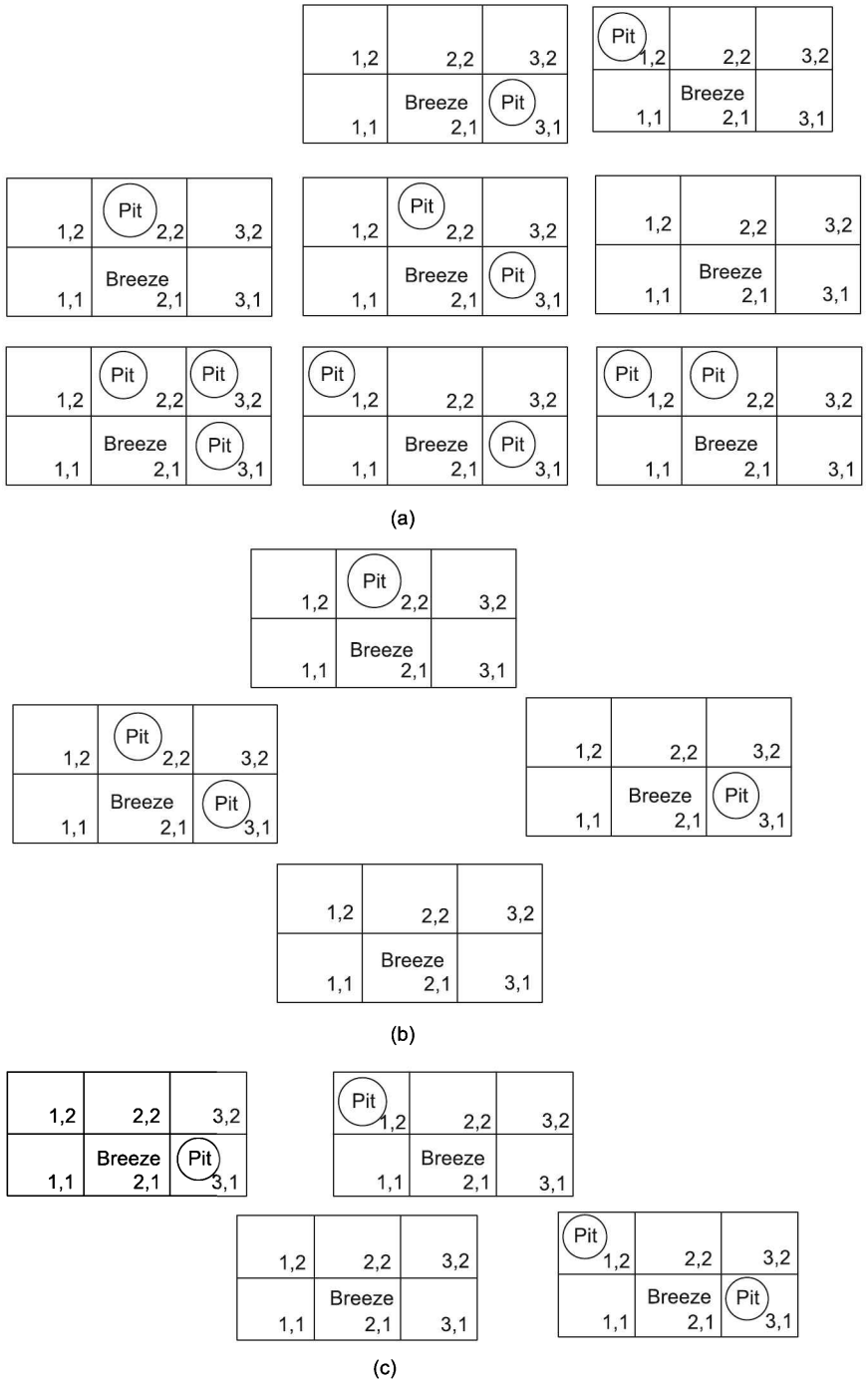


Figure 7.5 Possible (subset) cases in the Wumpus World for pits: (a) Knowledge base (KB), (b) Case 1: Subset representations, (c) Case 2: Subset representations.

7.5.1 Syntax, Semantics and Knowledge base Building

Syntax

In propositional logic, there exist two types of sentences—Simple and compound. An atomic sentence or simple sentence consists of a single propositional symbol. This symbol essentially represents if the proposition can be true or false. Does that mean that there are two propositional symbols with fixed meanings for true and false? The answer is yes. But as we study ahead in detail, things will be more clear.

The syntax of propositional logic basically defines the allowable sentences. As stated earlier, it is represented as symbols, they could be, say P, Q, L, M, and so on. A simple assertion is represented as p. This indicates that the proposition is true.

The five operators it has are briefed below:

1. Negation (~ or ¬)
2. Conjunction (∧)
3. Disjunction (∨)
4. Implication (→): If....then
5. Biconditional (↔): iff- if and only if

In propositional logic, we need to be very specific with respect to syntax. The rule followed for precedence is highest from ¬ to lowest ↔. It is also essential that we make use of parenthesis for setting the precedence.

Semantics

Semantics tells about the rules to determine the truth of a sentence. Things are simple when it comes to simple sentence. But with the compound one, the model comprising a number of propositions actually defines the truth value. There can be many models depending on the values of the propositions considering their permutations. So, with 2 propositional symbols playing role, the models will be 2^2, whereas with 3, it will be 2^3. For the operators, let us define truth tables by which we will get clarity of the values they would take up.

Let P and Q be the propositions. The truth tables for the operators defined in earlier section are shown in Table 7.2. The final value, as we can see, is dependent on the values of P and Q.

TABLE 7.2 Truth Tables

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$Q \rightarrow P$	$P \leftrightarrow Q$
False	False	True	False	False	True	True	True
False	True	True	False	True	True	False	False
True	False	False	False	True	False	True	False
True	True	False	True	True	True	True	True

From Table 7.2 we can see that the value of $P \leftrightarrow Q$ is true only when $Q \rightarrow P$ and $P \rightarrow$ Q are true. Consider the scenario of the Wumpus World again. There is a breeze in [1,1], let us represent this as B. Then, we know that a pit exists in [2,1] or [1,2]. Let P

represents pit in room [2,1] and Q represents pit in room [1,2]. Mapping it in propositional logic, we represent it as follows:

$$B \leftrightarrow (P \vee Q)$$

But we cannot have the implication operator, as it results into incompleteness of knowledge.

Building a Knowledge Base (KB)

For a subset of the Wumpus World, a knowledge base is required to be built.

To build any KB, the first step is to decide the propositions. We then construct the rules with the operators and the true/false values of propositions. Consider that i and j represent the room or the grid value. Let $P_{[i,\ j]}$ represents the proposition that is true if there is a pit in i, j. Similarly, let $B_{[i,\ j]}$ represents the proposition that is true when there is a breeze in i, j.

The KB comprises the rules now that are built with the propositions and operators. As an example one rule can be as follows:

Rule 1: $\neg\ P_{[1,1]}$: This rule states that there is no pit in [1,1].

Similarly, the earlier rule that we have derived in the semantics is also a KB-representing rule. We can rewrite the rule 2 as follows:

Rule 2: $B_{[1,1]} \leftrightarrow P_{[1,2]} \vee P_{[2,1]}$. In this way we can define the rules forming our KB for the Wumpus World.

Finally, the KB comprises all the rules or the sentences in the conjunction form, i.e., Rule 1 $\wedge$ Rule 2 ... $\wedge$ Rule n. This, in turn, tells that all the rules are valid and true.

7.5.2 Inference

Now, once we have our KB represented, it is the time that we decide upon the inference. With the propositions that we have discussed considering the Wumpus World, they can have different models depending on the values. By models, as said earlier, we mean the different values which the propositions take in the compound statement. So, by inferring we mean that it should be decided whether KB $\models x$. Here, x is some sentence.

So, when it comes to inference, we need to enumerate the models. Now, while doing so, the number of propositions playing part in it directly hamper the efficiency. The values where KB is true, and thus, the value of x is also true indicate inferring. As an example, assume that we have KB true for some model. Suppose you have a truth table of the values of propositions, rules and KB. A snapshot of the same is given below in Table 7.3.

TABLE 7.3 Values of Propositions—A Snapshot

$B_{[1,1]}$	$B_{[2,1]}$	$P_{[1,2]}$	...	$P_{[3,1]}$	$B_{[2,1]}$	*Rule* 1	*Rule* 2	...	*KB*
⋮									
False	True	False		True		True	True		True
False	True	False		False		True	True		True
False	True	False		True		True	True		True
⋮									

From Table 7.3, we can say that $\sim(P_{[1,2]})$ is true, so we can infer that there is no pit in [1,2]. But the same cannot be true in case of $P_{[3,1]}$, as in one case, it is false. This is just a simple case considered to understand inferring, but just image the number of models that would be generated with the increase in the propositions.

Given a sentence n, the approach for deciding entailment is based on recursive enumeration. The approach is:

Given—KB, x, list of symbols in KB and x.

Check—1. If symbols are empty then
check if model is consistent with KB
if true then check if x evaluates to true
else it is inconsistent
2. Else
recursively construct conjunction
for partial models with symbols in KB and x.

Some concepts: Tautology, Contradiction and satisfiability

1. *Tautology:* A tautology states that the sentence is true if it is true in all models. That means a proposition is always true. For example $(P \vee \sim P)$ will have true value, irrespective of the P values. Hence, it is a tautology, which is sometimes also called *validity*.

2. *Contradiction:* In contradiction, the proposition is always false. For example, $P \wedge \sim P$ will be always false, irrespective of the values taken by proposition P.

3. *Satisfiability:* A sentence or a proposition is satisfiable if it is true for some model. Let us say a sentence x is true in a model m then m satisfies x. m is a model of x.

Refutation

After studying satisfiablity, the question is how can it be determined? Can we say that we need to explore all the models and check them till we get one that satisfies the sentence? The answer is yes. To determine the satisfiability of a sentence is NP-complete problem. Now, when we talk about satisfiability, we can say that most of the problems related to computer science are satisfiable. (Constraint satisfaction and search problems have satisfiability playing a role in solving them).

Understanding the relation between the validity and satisfiability can be explained as follows:

Assume that a sentence x is valid. To prove that x is valid, we need to prove that $\sim x$ is satisfiable. Other way round, x is satisfiable iff $\sim x$ is not valid. Referring to KB representations, we can state that

$x \models y$ iff sentence $(x \wedge \sim y)$ is unsatisfiable.

Proving y from x by checking the unsatisfiability as mentioned above is *proof by refutation* or *contradiction*. This relates to the mathematical solving of problems, where by treating a value as false, we move into contradictions, and hence, carry the proof of refutation.

7.5.3 Reasoning Patterns in Propositional Logic

In reasoning patterns, we use and apply the basic rules in order to derive chains of conclusions to get the outcome or the target. These basic rules are also called *patterns of inference*.

The two most commonly used rules are *modus ponens* and *and-elimination*. Modus ponens is represented as follows:

$$\alpha \rightarrow \beta,\ \alpha \mapsto \beta$$

The rule states that when any sentence is in the form of $\alpha \rightarrow \beta$, and α is given, then we can infer β.

As an example, when we have the rule that (wumpus-ahead $\wedge$ wumpus-alive) $\rightarrow$ shoot and (wumpus-ahead $\wedge$ wumpus-alive); we can infer shoot. Considering one more example (mapping in propositional logic), suppose we have the following rule:

R: $\sim S_{[1,1]} \rightarrow \sim W_{[1,1]} \wedge \sim W_{[1,2]} \wedge \sim W_{[2,1]}$ and given that a stench is ahead

Then, with modus ponens, we get

$$\sim W_{[1,1]} \wedge \sim W_{[1,2]} \wedge \sim W_{[2,1]}, \text{ where } \beta \text{ is inferred}$$

In case of and-elimination, the rule is represented as follows:

$$\alpha_1 \wedge \alpha_2 \wedge \alpha_3 \wedge \alpha_n \mapsto \alpha_i$$

The rule states that from conjunctions, it is possible to infer any conjunction.

This also states that if we have wumpus-ahead $\wedge$ wumpus-alive, then we can infer wumpus-alive. From the above inference with stench, we get simply the and-elimination inference as follows:

$$\sim W_{[1,1]}\ \sim W_{[1,2]}\ \sim W_{[2,1]}$$

Now, the next question is again what is the reason behind considering these rules? These rules are actually the ones that eliminate the need for generating the models. When these rules are applied, they generate sound inferences.

Resolution

The previous point clarifies the rules being sound. We now need to discuss the completeness. Any search algorithm (we have discussed about completeness in Chapter 3) is said to be complete when it is guaranteed to go to a reachable goal. But what if the rules available are insufficient or inadequate? Can we reach the goal?

Resolution is a single inference rule that is discussed in this section, which gives a complete inference algorithm when coupled with complete search algorithm.

Coming back to the Wumpus World's example, some rules are considered here to understand the resolution. Consider the case where the agent is in [1,2], a case where he has returned from [2,1] to [1,1] and then has gone to [1,2]. Here, he perceives a stench, but no breeze. Now, we add some more rules.

Rule a: $\sim B_{[1,2]}$

Rule b: $B_{[1,2]} \leftrightarrow (P_{[1,1]} \vee P_{[2,2]} \vee P_{[1,3]})$

Rule c: $\sim(P_{[2,2]})$

Rule d: $\sim(P_{[1,3]})$

Rules c and d imply that pit is not present in [2,2] and [1,3]. Similarly, we can derive that there can exist a pit in [1,1] or [2,2] or [3,1]. Rule *e* represents the same as follows:

Rule e: $P_{[1,1]} \vee P_{[2,2]} \vee P_{[3,1]}$

But where is the application of resolution and how do we apply it? The resolution rule is applied in the rules c and e. By this we mean to say that $\sim P_{[2,2]}$ in rule c resolves with $P_{[2,2]}$ in rule e. By applying this, we get

Rule f: $P_{[1,1]} \vee P_{[3,1]}$

But the initial rule that we have in KB building is that $\sim P_{[1,1]}$. This again resolves with the rule f. Hence, what we get is

Rule g: $P_{[3,1]}$

This inference rule states that if there is a pit in [1,1] or [3,1] (by rule f) and there is no pit in [1,1] (by resolution), then there is definitely a pit in [3,1].

The steps applied to infer the rules f and g are called *unit resolution inference rules.*

The propositions involved in the process are also called *literals.* Resolution is actually the basis for complete inference procedures. Any search algorithm that is complete and applies the resolution rule can derive any conclusion that is entailed in KB. Suppose we know that a proposition X is true. It is not possible to have a resolution to generate $X \vee Y$, but we can determine if $X \vee Y$ is true or not. This is referred to as *refutation completeness.*

Conjunctive normal form (CNF): From the rules that are mentioned in the resolution, it is noticed that the resolution is applied only to disjunctions. But then, the rules can be in conjunction form too. What we need to do is to convert them into CNF, which is a conjunction of disjunctions. CNF is conjunction of clauses, where clauses are disjunctions of literals.

While converting, the following steps are to be carried out:

1. If a bidirectional implication exists, it has to be eliminated in following way:

   $\alpha \leftrightarrow \beta$, is replaced with $\alpha \rightarrow \beta$, $\beta \rightarrow \alpha$.

2. If an implication exists, then it has to be eliminated in the following way:

   $\alpha \rightarrow \beta$ is replaced with $\neg\ \alpha \vee \beta$.

3. Use equivalence to have ~ or $\neg$.

By applying the DeMorgan's theorem, we can rewrite the rules as follows:

(i) $\neg(\neg\alpha) \equiv \alpha$
(ii) $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$
(iii) $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$

Once these rules are applied, we can have the resolution inference rule applied.

Let us take a simple example. Assume α and β are in CNF. Therefore, $\alpha \wedge \beta$ is also in CNF.

Now, conversion of $\alpha \vee \beta$ can be carried out as follows:

1. In a given scenario, if α and β are literals, then

$$\alpha \vee \beta \text{ is in CNF}$$

2. If $\alpha = \alpha_1 \wedge \alpha_2 \ldots \wedge \alpha_1$, then

$$\alpha \vee \beta = (\alpha_1 \vee \beta) \wedge (\alpha_2 \vee \beta) \ldots \wedge (\alpha_1 \vee \beta)$$

Assuming β is literal, then

$$(\alpha_1 \vee \beta) \wedge (\alpha_2 \vee \beta) \ldots \wedge (\alpha_1 \vee \beta) \text{ is in CNF}$$

Otherwise β is conjunction of β_1 to β_k and using distribution, we can convert each α_i and β_i to CNF.

Resolution algorithm: After having studied *refutation* in the previous section, we move ahead with the resolution algorithm. In refutation, proof is with contractions. For example, to prove that KB $\models y$ we need to show (KB $\wedge \sim y$) is unsatisfiable. While applying the resolution algorithm, we

1. Convert (KB $\wedge \sim y$) into CNF.
2. We get some resulting clauses.
3. The resolution rule is to be applied to each clause.
4. The complementing pairs, are resolved to generate a new rule or a clause.
5. Add this to the KB if not already present.
6. Goto step 3 till any of the following two conditions occur:
   (i) No new clauses can be added in which KB does not entail α.
   (ii) Applying the resolution yields an empty rule, indicating KB entails α.

Forward and Backward Chaining

Resolution put forth an inference mechanism that is complete. But practically speaking, in most of the cases, we do not need to have the resolution. This is because the KB comprises restricted clauses or rules (clauses where at most a single proposition or literal is positive). A clause that is disjunction of literals of which at most one is positive is a *Horn clause*.

For example, the clause has $\neg S_{[1,1]} \vee \neg$ Breeze $\vee B_{[1,1]}$ is a Horn clause ($\neg S_{[1,1]}$ is the state of the agent at 1,1). But a clause like $\neg B_{[1,4]} \vee P_{[2,1]} \vee P_{[1,2]}$ is not a Horn clause. The Horn clause containing exactly one positive literal is called *definite clause*. This literal is called *head* and other literals are said to be *body*. When there is not any negative proposition in a definite clause, it is called *fact*. A clause (for example, $\neg P_{[3,1]} \vee \neg P_{[1,3]}$ is same as writing $P_{[3,1]} \wedge P_{[1,3]}$) is called *integrity constraint*. It points out the errors.

Forward chaining and backward chaining are the algorithms that are used for inferring. While applying the algorithm, the knowledge base comprises the Horn clauses. What are we trying to infer in these algorithms? Let us assume we have a query that is required to be resolved. This can be resolved by going through the KB to get the answer. This query is a proposition, say whether any position in the room has a pit or not. So, we are trying to find whether this proposition is entailed in the knowledge base.

Forward chaining: In forward chaining, the process starts with the known facts. From the known facts, it adds to the conclusion. This process is carried out till we reach out the query that we have to resolve or till no inference can occur further. The approach is mapped to an AND-OR graph. Here, the constraint is that till all the conjuncts are known, it does not proceed ahead.

The steps can be summarised as follows:

1. Start with the known facts.
2. If the premise of the implication in the clauses are known, then add conclusion to the facts.
3. Go to step 2 till
   (i) We infer the value as true or false for the query, or
   (ii) No inference can occur.

This approach takes linear time. This approach is also said to be *data-driven approach*, where we try to derive considering the available facts.

Backward chaining: In backward chaining, the processing starts with the query. So, we are moving from the goal to infer the facts that would tell us about it. If the query is true, halt. (Do nothing). But if not, then find the implications that infer the query. If all the premises of these implications can be proved to be true, we infer the query to be true. This method is also called *goal-driven method*. In turn it forms a part of goal-directed reasoning.

Comparing both the approaches, the time required in this is less than the forward one. The only reason behind this is that it goes through only the relevant facts and not others. Which method should be used ideally? It is a combination of both the methods that is used by the agent during the course of inferring.

7.6 PREDICATE LOGIC

We have already studied the propositional logic—one way that helps in knowledge representation and reasoning. In this section, predicate logic is discussed. Predicate logic also known as *first order logic* is said to be more expressive. The reason of studying other form is propositional logic becomes hard and complex, when it comes to representation of the complex environments. Later, in the chapter, a comparison is made between the two logics.

Predicate logic allows to describe the objects involved and their relationships. Consider following example:

All kids are naughty.
Suzy is a kid.
Then, Suzy is naughty.

Expressing the above example in propositional logic is difficult to have valid sentences or clauses. They can be better represented in predicate. Predicate logic is powerful tool that can express and give reasoning and it is built on the top of propositional logic, making the use of ideas.

7.6.1 Representing Facts in Logic: Syntax and Semantics

The predicate logic, as the name suggests, handles the representations in the form of predicate. Any sentence has as a subject and a predicate. For example, consider a sentence—'The car is red'. Here, 'car' is the subject and 'is red' is the predicate. While representing, it would be $R(x)$, where R is red and x is any object. This is a very basic example. We now go into the details of representations. A sentence in predicate logic is built up from constants, predicates and functions. A sentence can be an atomic sentence or sentences connected to each other.

We can represent it in the following way:

Sentence → Atomic sentence |
Sentence connective sentence |
Quantifier variable_name,..... sentence |
~ sentence

Further, the atomic sentence comprises predicate. The predicate has terms, which could be functions, constants or variables.

Atomic sentence → Predicate (Terms)
For example, Predicate: Blue, Academic, and so on. They express the relationships.

Term → Function(Terms) |
Constant |
variable_name

Constants: Mary, A, etc. They are the objects for which we want to talk about.
Variable_name: p, q, r, ..., etc.
Function: It allows to refer to the other objects indirectly. For example, father_of

The connectives and the quantifiers are used in the formulation of the sentence.

Connectives: $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$
Quantifiers: $\forall$, $\exists$

We need to make a note of the difference between the function and the predicate. As said earlier, a function is indirect reference to the objects. Predicate comprises the functions. It will be more clear in the example explained later. We start with the simple ones.

Simple Sentence

Sita is the mother of Rohan.
Representation: Mother(Sita, Rohan)

Complex Sentence

Reeta's uncle and Rohan's daddy booked a flat.
Representation: Booked (Uncle(Reeta), Daddy(Rohan))

Another example can be Father(Sita) = Ramesh

Connectives

Simple example of use of connectives can be:

Father(Rohan) $\wedge$ Father(Shyam)
$\sim$ Dancer(Riya) $\Rightarrow$ Dancer(Sita)

Quantifiers

There are two types of quantifiers—which are as follows:

1. $\forall$: Universal quantification (pronounced as 'for all')
2. $\exists$: Existential quantification (pronounced as 'there exists an x such that' or 'for some x')

Universe of discourse: There is a concept of universe of discourse in predicate logic. It is a set of all the things that we talk about. This is the set of objects that we can assign to a variable in a propositional function. For example, in a wide sense, it can be a set of integer that possibly defines the boundaries.

Universal quantification **($\forall$):** Syntax: $\forall\ x\ P$; where P is a logical expression.

$\forall\ x$ is called universal quantifier and P is the scope of the quantifier. The x is said to be bound by the scope of the quantifier.

In the example discussed earlier, we had a sentence "All kids are naughty."

It can be represented as follows:

$$\forall\ x\ \text{Kid}(x) \Rightarrow \text{Naughty}(x)$$

This says that for every x, if x is kid, then x is naughty.

Make a note that typically, the statements containing the words 'every', 'each', 'everyone' indicate universal quantifier.

Existential quantification **($\exists$):** Syntax: $\exists\ x\ P$; where P is a logical expression.

This indicates that P is true for at least one value. ('There exists'.)

For example, Some people are kind.

This can be written as

$$\exists\ x\ \text{Kind}(x)$$

This says that there exists some x that is kind.

The statements containing 'some' or 'at least one' indicate existential quantifier.

These quantifiers are treated as unary connectives and have a higher precedence over the binary connectives.

7.6.2 Instance Representation and ISA Relationships

ISA and instance play a very important role in knowledge representation. These two attributes exhibit the property of inheritance. The following sentences help us in understanding these attributes:

Shyam is an engineer.
All engineers are intelligent.

Here, intelligent and engineer are the classes. ISA shows the class inclusion. ISA (engineer, intelligent) indicates that engineer class is contained in the intelligent class. Whereas, instance, as the name suggests, indicates the membership belonging to the class. Instance(Shyam, engineer) indicates the class membership.

7.6.3 Comparison of Predicate and Propositional Logic

After studying the predicate and the propositional logic in detail, let us compare them.

In terms of representation, propositional logic requires a separate unique propositional symbol. So, for representing any particular fact, it takes many symbols. (This one must have noticed in the Wumpus World). Imagine the number of symbols it would take if there were n locations and m people and you need to represent the fact of movement of some person.

Predicate logic is rich in terms of ontology. It includes the facts, relationships and the objects, whereas propositional logic is based on facts. Predicate logic also makes a compact representation available.

With respect to the complex sentences, in the introduction part of predicate logic itself, we have shown that the propositional logic cannot handle them.

7.7 UNIFICATION AND LIFTING: INFERENCE IN FOL

Unification and lifting are concerned with the inference in first order logic (FOL). It introduces the notion of logical inference. In inferring, we need to find out the results. To achieve this, quantifiers are required to be removed. This is possible by use of propositional logic. So, the basic idea is to infer in FOL; the rules of KB are converted to propositional logic and then the propositional inference is used.

We will proceed step-wise starting from the inference rules for quantifiers to the unification algorithm.

Inference Rule for Quantifiers

While inferring from quantifiers, two common rules are used—rule of universal instantiation and rule of existential instantiation.

Rule of universal instantiation (UI): Assume that the knowledge base contains the axiom—'All students who are kind are intelligent'. This can be represented as follows:

$$\forall x \text{ Student}(x) \wedge \text{Kind}(x) \Rightarrow \text{Intelligent}(x)$$

For this example, if we want to infer for Shyam or Rohan, we would have

$$\text{Student(Shyam)} \wedge \text{Kind(Shyam)} \Rightarrow \text{Intelligent(Shyam)}$$

In the same way, it can be inferred for Rohan. What we are doing here is making the use of substitution. The rule of universal instantiation is used for substitution. The rule states that by substituting a ground term (i.e., a term without variables) for the variable, we can infer any sentence. The rule is as follows:

Let *Subs* (θ, S) denote the final outcome after applying the substitution, where θ is the substitution applied to the sentence S.

$$\frac{\forall \vee S}{Subs(\{v/t\}, S)}$$

The rule says that for any variable v of S, the substitution occurs with t, where t is ground term. It is because of this rule that we have derived the previous results. So, $Subs\{v/\text{Shyam}\}$ was used.

Rule of existential instantiation (EI): For any sentence S, and variable v, and a constant symbol c (this should not appear anywhere in the knowledge base), the rule is given below:

$$\frac{\exists \vee S}{Subs(\{v/c\}, S)}$$

Again, take a hypothetical KB having the following sentence:

$$\exists x \text{ Car } (x) \wedge \text{Drive } (x, \text{Shyam})$$

To infer we can have Car(abc) $\wedge$ Drive(abc,Shyam).

In this sentence, by this rule, it implies that as long as the constant abc does not appear anywhere in the KB, the inference is correct. So, a new name is to be used, this is called *Skolem constant*.

One thing to mention here is that with the application of UI, the new KB is logically equivalent to the old one. UI can be applied several times, but with EI, as it is applied once, new KB is not equivalent, but satisfiable.

Reducing to Propositional Inference

In order to reduce from predicate logic to propositional, we need to have the rules for inferring the non-quantified sentences from the quantified ones. The technique of propositionalisation is discussed here. The substitution methods mentioned in the previous section are to be used. In case of existential quantified sentence, it can be substituted by one instantiation, whereas in case of universal, it is replaced with all possible instantiations. Let us take the following KB:

$\forall$x Student(x) $\wedge$ Kind(x) $\Rightarrow$ Intelligent(x)
Student (Shyam)
Kind (Shyam)
Friends (Shyam, Rohan)

Now, since we want to have the reduction to take place, after applying the substitution methods, we can have the representation as

Student (Shyam) $\wedge$ Kind (Shyam) $\Rightarrow$ Intelligent (Shyam)

Student (Rohan) $\wedge$ Kind (Rohan) $\Rightarrow$ Intelligent (Rohan)

The KB now has just the axioms that can be mapped as propositional symbols. The new KB would contain Student (Shyam), Kind (Shyam), Intelligent (Shyam), Student (Rohan), Kind (Rohan), Intelligent (Rohan).

Problems with propositionalisation: It is obvious to us that propositionalisation creates many irrelevant sentences. Now, we are aware of the substitution of Intelligent (Rohan).

But then is there a need to generate Kind (Rohan) if in the KB it is specified Friends (Shyam, Rohan)?

This inferring generates a lot of facts. If we are having p predicates of k-ary and n is the number of constants, then the outcome of propositionalisation would be $p * n^k$.

7.7.1 Unification

It is the process of finding the substitutions that make different logical sentences look identical, i.e., the process of finding substitutions for predicate parameters. One would think that these inferences are so simple; you look at the sentences and understand. But it is required that the machine understands them, and hence, there is need of process of substitution.

For example, $\forall x$ Student(x) $\wedge$ Kind $(x) \Rightarrow$ Intelligent (x)
Student (Shyam)
Kind (Shyam)

From this, we infer that Shyam is intelligent.

First Order Inference Rule

While inferring that Shyam is intelligent, we have applied the substitution method. First, x is found such that x is student. Then, x has to be kind and after that, we infer that x is intelligent. We have carried out the substitution of $\{x/\text{Shyam}\}$. Assume we do not have the fact of Kind (Shyam), instead we have

$$\forall y \text{ Kind}(y)$$

Will it be possible to conclude that Shyam is intelligent? Naturally, as Shyam is student and that everyone is kind, this inference stands true.

So, we have $\{x/\text{Shyam}\}$ and $\{y/\text{Shyam}\}$.

The identicalness achieved is student(x) and Kind(x) with student(Shyam) and Kind(y). This generalised process is the rule of generalised modus ponens.

The rule states that for the atomic sentences, a_i, a_i' and q, there is a substitution θ such that $Subs(\theta, a_i') = Subs(\theta, a_i)$ for all i.

$$\frac{a_1', a_2', a_3', \ldots, a_n', (a_1 \wedge a_2 \wedge a_3 \ldots a_n) \Rightarrow q}{Subs(\theta, q)}$$

a_1 : Student(x)
a_2 : Kind(x)
q : Intelligent(x)
a_1' : Student(Shyam)
a_2' : Kind(y)
θ : $\{x/\text{Shyam}\}$ and $\{y/\text{Shyam}\}$
$Subs(\theta, q)$: Intelligent(Shyam)

The generalised modus ponens is a lifted version of modus ponens. It raises the modus ponens from the propositional to FOL. This is called *lifting*.

Unification Algorithm

Now, finally coming to the discussion of unification, how do we determine the substitution of θ? The rules of inference that are lifted need to find substitutions that make different logical sentences look identical. This is called *unification process.*

The algorithm is given below:

The algorithm Unify(a, b) takes two sentences and returns a unifier for them, if there exists one. In short, we now decide the value for θ.

Unify(a,b) = θ, where $Subs(\theta,a) = Subs(\theta,b)$

Let us try to answer the query Friends(Rita,x)—who all are friends of Rita?
Assume in the KB, we have the following:

Friends(Rita,Seema)
Friends(x,Maya)
Friends(y,Neha)

Applying the unification, we get:

Unify(Friends(Rita,x),Friends(Rita,Seema)) = {x/Seema} = θ
Unify(Friends(Rita,x),Friends(x,Maya)) = Fail= θ – *
Unify(Friends(Rita,x),Friends(y,Neha)) = {x/Neha,y/Rita} = θ

The case marked as * is failed due to the use of same variable name. That is, x cannot be Rita and Maya at same time. It is required that the names should be different. To achieve this, standardising is done. The variable name is changed, and hence, we could get

Unify(Friends(Rita,x),Friends(z,Maya)) = {x/Maya, z/Rita} = θ

The algorithm returns substitution to make x and y look identical.
Let us write a generalised algorithm for it.

1. If x and y are variables or constants,
   (i) x and y are identical return NULL
   (ii) Else, if x is variable and x occurs in y, then fail, else return {y/x}
   (iii) Else, if y is variable and y occurs in x, then fail, else return{x/y}
   (iv) Else fail
2. If the arguments mismatch, return fail.
3. If the predicates do not match, return fail.
4. Let *Subs* = { }
5. For *ctr* =1 to the number of arguments
   (i) Go to step 1 (call again) with *i*th argument of x and y and add result to *Sol.*
   (ii) If *Sol* = fail, return fail.
   (iii) If *Sol*! =NULL.
   Again apply *Sol* to the remaining part of x and y.
   Add *Sol*: *Subs* = *Sol* + *Subs*
6. Return *Subs*

Unification plays a very important part when it comes to natural language parsers. It has a strong mathematical concept and is important in the other AI programs as well.

7.8 REPRESENTING KNOWLEDGE USING RULES

This topic essentially covers the use of rules for the representation of knowledge. After discussing the details of propositional and predicate logic, now, the rules for encoding the knowledge are discussed. We have studied about the representation of sentences in logic formats, but the rules that actually shape the knowledge are very crucial. Let us study the representations. This section essentially covers the gist of the earlier part studied in terms of reasoning, i.e., the necessities in rule-based system with KB representation.

7.8.1 Declarative and Procedural Knowledge

The need for the representation of knowledge in terms of rules is finally to get the solution to our problem. In the previous sections, we have looked into the assertions. Here, we take them further to resolve the problem.

Two types of representations exist—declarative and procedural. In case of declarative, knowledge is specified; but extent upto which the knowledge is required to be put up is not specified. The assertions mentioned in the previous section are declarative. In order to use a declarative representation, it needs to be augmented with a program that specifies what is to be done and how it is to be done. The assertions can have resolution theorem applied. These assertions can be viewed as program. While doing so, the implication statements actually define the reasoning. In simple words, we can say that the declarative representation is of facts.

In case of procedural representation, the control information required to make use of the knowledge is embedded in the knowledge. So, here, an interpreter is required that understands the instructions in the knowledge. It can have heuristic too to have the result generated. Let us take the following example of the KB:

Bird(Parrot)
Bird(Sparrow)
Feathers(Pigeon)
$\forall x$ Bird(x) $\Rightarrow$ Feathers(x)

Let us say we want to solve a query to find out some y that has feathers.
The query representation will be as follows:

$\exists y$: feathers(y)

What is the expected answer? It will give us the entire three—sparrow, parrot as well as pigeon. So, it is dependent on the order in which the assertions are actually being written. If some control knowledge is added to make it procedural, the answer that we might get is pigeon. This control knowledge could be the order in which the facts are to be evaluated, say depth first search.

Which representation should be selected? Generally, viewing the knowledge as procedural or declarative is not an essential characteristic for building the KB, but it is the method that permits for knowledge extraction.

7.8.2 Logic Programming

It is a programming language paradigm in which logical assertions are viewed as programs. It comprises facts and rules. The facts are the axioms and the rules determine whether the axioms are sufficient to determine the truth of the goal. We discuss about **pro**gramming **log**ic (Prolog) in brief to understand the paradigm. Prolog comprises Horn clauses. In Prolog, the interpreter has a fixed control strategy. The working of the control strategy is explained below:

1. It begins with the problem statement, mapping it to the final goal to be achieved.
2. Checks for assertions and rules to see if the goal can be proved.
3. For this decision to apply a rule or fact, unification process is also invoked.
4. It uses backward reasoning to get the solution.
5. It may apply depth first strategy and backtracking.
6. When it comes to the choice, consider the order while selecting the next rule or fact.

7.8.3 Forward and Backward Reasoning

In Prolog, the reasoning is backward. But in practice, there are two types of reasoning—forward and backward.

Forward Reasoning

In forward reasoning, the process starts with the starts states. A tree is built considering the different moves or intermediate steps that could be the solutions. At each point of time, the next level is generated with the selection of rule, whose left side matches with the root. The process is from left to right evaluation of a rule. This process is continued till the goal state is achieved.

Backward Reasoning

It is exactly reverse of forward reasoning, where the process starts with the goal. The rules whose right side matches with the root are considered. So, these rules define the next level that would be required for the generation. So, the process is from right to left. Once a rule is applied, its left side is then looked upon and searched in the right side again. This is continued till we reach the initial state.

Is it possible that the same rules are being used in the reasoning process? Yes. But then, which reasoning is to be applied is dependent on the topology of a problem.

In the eight-puzzle problem discussed in earlier chapters, the direction did not matter. But it does matter when

1. There are more goal/start states.
2. If new facts are likely to be evolved, then forward reasoning is better.
3. The branching factor is at the decision node.
4. Finally, the justification required may make the user to use the reasoning he/she thinks.

The two rule systems that now are taken into account are the forward and backward chaining. Prolog and MYCIN are the two systems that are under this category. A system that is directed by goals is *backward chaining*, whereas in case of *forward chaining*, it is data driven. We feel that forward chaining is very simple to process, but the rule application and matching in forward chaining are more complicated.

Is it possible to have a combination of forward and backward reasoning? The answer is yes. As an example based on some facts, if one tries to infer or conclude, especially in diagnostic cases, it is better to apply backward reasoning.

7.8.4 Matching

The question that is addressed here is how can the rules be extracted that are matched in the course of inferring a position with the current state? This needs to perform matching that is to be carried out between the current state and the preconditions of the rules. Different methods of matching are discussed below:

Indexing

A very simple thought that comes to our mind is to simply check every precondition with the current state. Are we performing linear search? Naturally, this is not the option that would be looked at because of the following two reasons—1. As discussed earlier, the large number of rules that would be existing, would end up, thereby making it the most inefficient method. 2. It is not obvious to say that whether the preconditions would be satisfied by any particular state.

The most obvious answer to this problem is the use of indexing. We need to access the appropriate rules and for that, we use index. So, an index is typically used in this case. If we have a chess board, an index is assigned to the board positions. Further, with the use of hashing technique, the key that represents the same position comprises the same rules which give a specific position. So, with the same key, all the required rules can be found. Will this method have any pitfalls? The representation has all the positions and cannot handle its generalisation. Rules can be indexed by the pieces and their positions. Though there is a drawback, indexing is a very important aspect of the matching process.

Matching with Variables

When we say that the rules are being matched, one more problem that can arise is the unmatching of the preconditions with that of the current situation. What should be done under this scenario? So, again a search is to be carried out to check whether there exists a match between the preconditions and the current state.

Generally, the unification algorithm is used to solve this, and is useful too with the single-state description scenario. Forward and backward method too can be employed here. Even unification can be applied again, but it is better to have many-many match. Here, many rules are matched against many elements. Let us discuss about an algorithm for this many-many match—RETE.

The algorithm maintains the network of the rules and it uses state descriptions to determine which rules can be applicable. This is possible, as the rules are not changed randomly and the state descriptions do not change. Structural similarity in the rules is also

used. A very simple example of this is has-strips and has-spots. Actually, these predicates are structurally same. The algorithm is also persistent in case of variable binding.

The Rete algorithm is based on facts—rule pattern matching problem. It preserves information about the structure of network with matches.

The network comprises nodes that are individual condition elements. Every node has two sets.

1. Set containing all the working memory elements, where the condition node matches.
2. Combinations of working memory elements along with the bindings that generate consistent match of the conditions.

This setup helps in avoiding repetitive testing of all rules in each cycle. Thus, the nodes impacted during addition of new facts are checked.

For example, we have rule

if $P_1(A, 5)$ and $P_2(A, C)$ then res(A, C)

if $P_1(A, 6)$ and $P_2(A, C)$ then res(A, C)

It begins with the structure, as shown in Figure 7.6.

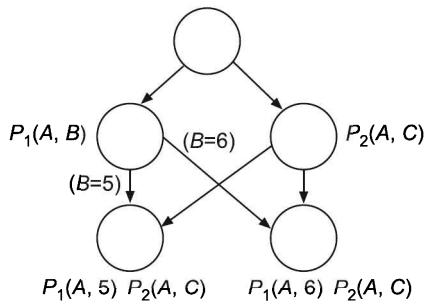


Figure 7.6 Rule pattern matching.

If a new fact $P_1(7, 5)$ is added, then it indicates mismatch in rule, as shown in Figure 7.7, where the data is propagated.

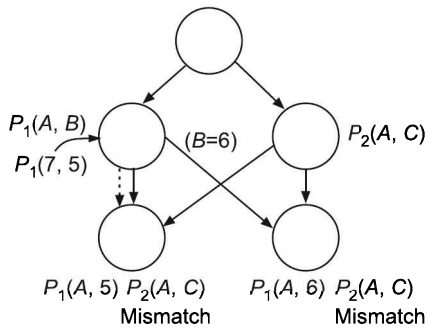


Figure 7.7 Rule propagation.

Thus, Rete algorithm performs efficient matching.

Approximate and Complex Matching

Another aspect of matching is required when the preconditions describe the properties that are not explicitly mentioned in the current state. Here, altogether a new set of rules are required to describe how only some of the properties can be inferred. There could also be a case, where we need to have the preconditions applied approximately. The most common example here is the speech. Mapping this while considering the noise and interference, it will be very difficult to approximate.

In some cases, matching the rules to the problem statement is done. ELIZA—a very old AI program that simulated the behaviour of a Rogerian therapist had used this matching. It matched the left side of rules against the last sentence a user would type. A simple dialog between ELIZA and a woman is given below:

Woman: I need your help.
ELIZA: How can I help you?

Or possibly, it would answer saying 'Tell me more about yourself', if the woman says, 'I am the only child in the family', and so on. One thing that you must have noticed is that there is a pattern matching; the patterns ask about some specific word that has been used by the woman. The entire sentence need not be matched. This approximation makes ELIZA put forth its accountability in an effective manner.

Conflict Resolution

In matching process, the rules need to be matched and it is equally necessary to decide the order in which this is to be carried out. Though it is the search method, which is responsible for matching, the decision can be built in the matching process. This stage of matching is referred to as *conflict resolution*. This preference assignment can be made on the following bases:

1. Rule that is matched
2. Objects that are matched
3. Action that the matched rule performs

7.9 SEMANTIC NETWORKS

We have already mentioned about semantic networks in the introductory section with respect to the knowledge representation. Here, we provide details of the same.

The basic idea behind semantic network is that knowledge is better understood as a set of concepts that are related. In other words, the semantic networks represent the conceptual relationships. A semantic network comprises nodes that are connected to each other by arcs. There are different variants of this network and each of them essentially does the same thing. It is noticed that the boxes or ovals are used to represent the objects or the categories of objects.

Common Semantic Relations

Though there is not a fixed set of relationships to express, we highlight the most common ones that invariably occur in every logic.

1. *Instance:* x is an instance of y, if x is a specific example of the general concept y. For example, Tendulkar is an instance of batsmen.
2. *ISA:* x ISA y if x is a subset of more general concept y. For example, batsman ISA cricketer.
3. *Haspart:* x is haspart of y, if y is part of the concept x. For example, stumps haspart bails.

Considering the same example of mapping the knowledge of some cricket domain, we can draw the semantic network. The semantic network for same is depicted in Figure 7.8.

From Figure 7.8, it is understood that the team and the runs are the values of attributes. Arrows indicate the point from object to its corresponding value. The logic for the above semantic network can be ISA(Batsman, cricketer), instance (Tendulkar, Cricketer), team (Tendulkar, India) and so on.

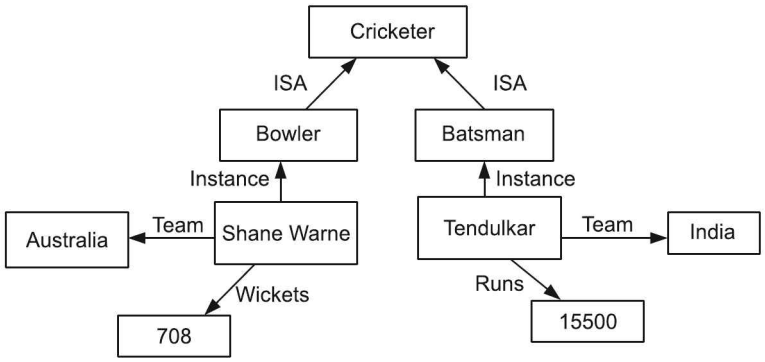


Figure 7.8 Semantic network.

ISA and instance are actually inheritable instances. Now, if we have two predicates, is it possible to map in semantic network? Definitely, we can do so. By creating the corresponding nodes and representing them with their relationships, it is possible to have a semantic network.

Inference in Semantic Network

While inferring in the semantic network, two mechanisms are followed, viz. intersection search and inheritance.

In *intersection search*, the intersection between the nodes is found, and in turn, the relationships too. This is achieved by spreading the activation and making the nodes visited. Whereas, in case of *inheritance*, the ISA and instance allow to implement this process. It provides the means for enabling default reasoning. While making the inferences, it is also required that the difference between the links (which hold values) is clearly mentioned. For example, the runs taken by players Shyam and Ram need to be explicitly differentiated, say by means of greater than link, where we need to compare them.

Figure 7.9 could be transformed into Figure 7.10. The arrow between Runs_1 and Runs_2 is the comparison of the runs.

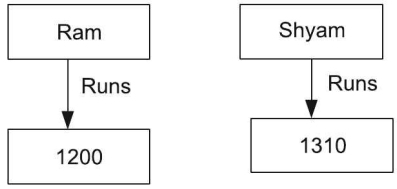


Figure 7.9 Inference with semantic network: Example.

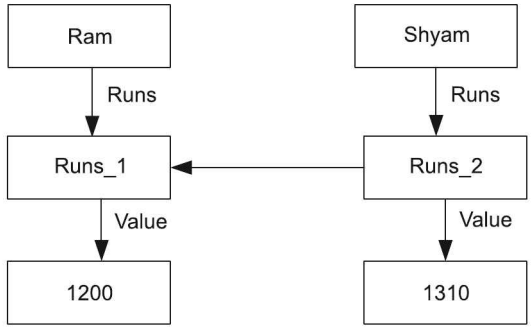


Figure 7.10 Inference with semantic network: Example.

7.9.1 Partitioned Semantic Networks

Partitioned semantic networks are the extension of semantic networks. They overcome the drawbacks of semantic networks or extend their knowledge expression. Most often, in semantic network, there exists vagueness. This occurs while finding the meaning of the tokens. The inference mechanism too is inefficient, and heuristic inadequacy exists as well. Partitioned semantic network forms a more sophisticated logical structure. Here, the network is split into spaces that consist of nodes and arcs and each space is then considered to be a node.

Consider an example, where a lawyer argues that the case is strong (Figure 7.11).

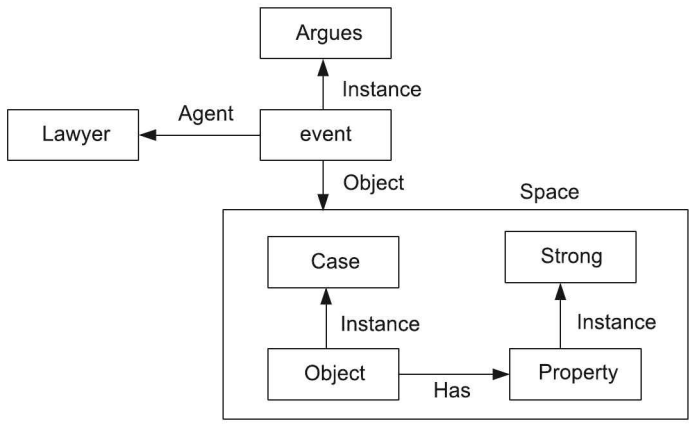


Figure 7.11 Partitioned semantic network.

From Figure 7.11, it is understood that the 'case is strong' is partitioned into a space that in turn, refers to an object or a node. So, the partitioned networks allow for the propositions, without commitment to the truth and the expressions to be quantified.

These networks enable to distinguish between the objects and to control the search space, proving them logically as well as heuristically adequate.

7.10 FRAME SYSTEMS

Frame systems or rather *frame-based systems* are knowledge representation systems that use frames. Frames are also regarded as extensions of the semantic networks. Semantic networks are concerned with labelled representations. But when the task gets more complex, what is required is a structured representation. A frame system handles this. A *frame* is a collection of attributes or slots. It is a structure to represent the concepts that have some associated value in the real world.

7.10.1 Minsky's Frame

The frame structure was proposed by Minsky in 1975. It was proposed that the knowledge should be organised into chunks, called *frames*. These frames capture the essence of the concepts by clustering relevant information. So, a large amount of procedurally expressive knowledge should be a part of the frames. Clubbing and organisation of such frames form the frame system.

Representation of Frames

Each frame has a name and slots (think of the concepts that we have studied in object-oriented programming—OOP). A simple frame is represented below (refer to Figure 7.8):

Tendulkar
  Instance: Batsman
  Runs: 15500
  Team: India

We can have the frames for 'cricketer'. This would be a class. The frames for 'Tendulkar' and 'Shane Warne' are instances. We can also define a meta-class, which contains the elements that are classes themselves.

7.11 CASE GRAMMAR THEORY

Case grammars provides an all together different approach to the problem with regard to the field of linguistics for studying the human languages. Thus, the theory plays a role in natural language processing. It shows that how syntax and semantic interpretations can be combined. So, it represents one more way for knowledge representation. The grammar rules are such that they describe the syntactic regularities instead of semantics, but when it comes to the structures that are been produced by them, they have semantic

relations rather than the syntactic ones. We will discuss about this theory in detail in Chapter 12.

7.12 INFERENCE

Inference is a mechanism by which reasoning takes place. They go together and it is very much essential to have an appropriate inferring mechanism so as to carry out reasoning. This section, deals with the different categories of reasoning by which new conclusions are derived.

7.12.1 Deduction, Induction, Abduction

They are the basic types of inferences. We introduce the concept of each of them as follows:

Deduction

Deduction is the process of deriving a conclusion from the given axioms and facts. So, drawing a conclusion based on the available knowledge and observations is deduction. This conclusion can be drawn by applying the modus ponens rule.

A very simple example to explain this can be as follows:

Axiom: All kids are naughty.
Fact/Premise: Riya is a kid.
Conclusion: Riya is naughty.

Inference in deduction is also referred to as *logical inferring*. Deduction is often called *truth preserving* also. Calculus can be used for this inferring.

Induction

Induction is deriving a general rule (axiom) from the background knowledge and observations.

For the same example, we can say that
If we have the knowledge with us that

Riya is a kid.
Riya is naughty.

then we can have the general axiom that

Kids are naughty.

Abduction

A premise is derived from a known axiom and some observations. Consider the same example, but now the information available is different.

Information available:

All kids are naughty.
Riya is naughty.

We can infer that Riya must be kid.

Thus, we get the differences and understand the process of inferring. But which is to be used and at what point of time? Abduction is typically for the diagnostic and expert system, and one must have guessed that deduction is concerned with FOL to get the inferring.

7.13 TYPES OF REASONING

The type of decision-making and reasoning changes as per the application and domain involved. This section highlights some of the reasoning approaches.

7.13.1 Common Sense Reasoning

Common sense reasoning is concerned with the way we think. This is essential when we are representing the KB. Do the machines have common sense? We need to embed it. While doing so, we need to formalise the reasoning. This is possible with the use of formal logic. In humans, we relate common sense with the ability of taking good judgement. But in case of AI, this is the technical sense with the facts and understandings. This reasoning is required especially in natural language processing. Various tools are available for the common sense knowledge base like ConceptNet that is an extended version of WordNet.

7.13.2 Hypothetical Reasoning

In hypothetical reasoning, different assumptions are looked at to see what can be followed or inferred from them. It can be termed as the reasoning about different worlds. When and where is the requirement of this type of reasoning? It is applied for diagnostic system, where we can assume some hypothesis and then make predictions followed by their verification. A method of hypothetical-deductive reasoning is applied in this case.

The steps involved in the hypothetic reasoning are as follows:

1. Evaluate each of the hypothesis.
2. Hypothesis is selected and tested from which predictions are generated.
3. Using experimentation, predictions are validated for correctness. If these are correct, then the hypothesis is accurate, else the hypothesis is not valid.

7.13.3 Analogical Reasoning

It is the process of reasoning from one particular object to another. Conclusions are derived from experience or similarity-based situations or conditions. For example, P is like Q. If P contains A and B is in Q, then we can infer that A is like Q. While reasoning, we find out the analogy in the given data. This is like finding the 'like' thing. In order to use the analogy

1. One starts with a target, where a new understanding is to be created.
2. Matching domain is found and further, the matching terms as well.
3. Ahead of it, related terms are found and then attributes are transferred from matching to target.

SUMMARY

The chapter highlights the need for proper knowledge representation and reasoning. Representation of the knowledge should be such that the inference is efficient. With the different ways of representing knowledge, logic administers the overall mechanism. To have a sound inference, accurate and precise logic is a must. Propositional logic and first order logic are used in the knowledge formulation. Semantic networks help in understanding the relationships and prove to be beneficial in understanding the hierarchy structure. Further, the methods of unification and lifting make the inferring simpler. The partitioned semantic networks and the frames further help in getting the knowledge representations better. Still with the known facts, it is definitely not a complicated task to handle, but with limited and uncertain knowledge, they need to be handled in a different way. The next chapter deals with the same.

KEYWORDS

1. **Knowledge:** It is the information that helps in taking decisions or solving problems at hand.
2. **Knowledge base:** Knowledge is stored in a structure called knowledge base, which is often referred to as KB. A KB is formed with the available facts and updated in the process of acquiring more knowledge that is used by an agent.
3. **Relational knowledge structure:** It is a knowledge structure where the facts can be mapped into the relations and stored in the database.
4. **Inheritable knowledge structure:** It is a structure that enables inheritance of the properties to improve the inference.
5. **Slot-filler structure:** It is a structure that belongs to the category of inheritable structure and allows easy accessibility and retrieval of the required information.
6. **Inferential knowledge structure:** It is a knowledge structure that makes use of logic in representation and maps the inferred facts.
7. **Procedural knowledge structure:** Procedures with the knowledge embedded in them form this type of structure.
8. **Knowledge based agent:** It is an agent whose actions are not arbitrary but are based on some valid knowledge. KB is used by the agent.
9. **Horn clause:** It is a clause that is disjunction of literals of which at most one positive literal exists.
10. **Unification:** It is the process of finding the substitutions that make different logical sentences look identical.
11. **Partitioned semantic network:** It is the extension of semantic networks, where the network is split into spaces that consist of nodes and arcs and each space is then considered to be a node.

12. **Frames:** It is also the extension of the semantic networks. It is a structure to represent the concepts that have some associated value with regard to the real world.
13. **Minsky's frames:** Early frame structure was proposed by Minsky. These frames capture the essence of the concepts by clustering some relevant information.
14. **Deduction:** It is the process of deriving a conclusion from the given axioms and facts.
15. **Induction:** It includes deriving a general rule (axiom) from the background knowledge and observations.
16. **Abduction:** In abduction, a premise is derived from a known axiom and some observations.

MULTIPLE CHOICE QUESTIONS

1. Given a fact and an axiom/premise, the reasoning falls under
   (a) Induction (b) Deduction (c) Abduction (d) None of these
2. A procedure approach that produces proof by contradiction is
   (a) Abduction (b) Refutation
   (c) Logic programming (d) None of these
3. Which of the following is involved between the mapping of initial facts to the final facts?
   (a) Forward chaining (b) Forward representations
   (c) Reasoning (d) Both (b) and (d)
4. The basic difference between the normal on simple agents and the knowledge-based agents is
   (a) Knowledge-based agents can capture the entire percept, whereas the simple cannot.
   (b) Knowledge-based agents are suited for some specific tasks.
   (c) Simple agents can take arbitrary actions, whereas knowledge base do not.
   (d) All of the above
5. For a query like 'Where is my mobile?' to be resolved, which approach is appropriate?
   (a) Forward chaining (b) Forward checking
   (c) Modus ponens (d) Backward chaining
6. Consider the following statements:
   $S(x)$: x is a primary school student.
   $I(x)$: x likes to play games on i-pad.
   Every primary student likes to play games on i-pad' can be mapped as (where universe of discourse is a set of students)
   (a) $\forall x(S(x) \rightarrow I(x))$ (b) $\exists x(S(x) \rightarrow I(x))$
   (c) $\forall x(I(x) \rightarrow S(x))$ (d) $\exists x(S(x) \vee I(x))$

7. Which of the following clearly defines a frame system?
   (a) An inference system
   (b) A system that maps the facts and beliefs
   (c) A form of knowledge representation
   (d) A system with a set of facts and their instances
8. Which of the following best justifies the knowledge?
   (a) It is a known information
   (b) It is a set of reasoning system
   (c) It is used for inferring
   (d) It is a set of assumptions

CONCEPT REVIEW QUESTIONS

1. What is a knowledge base?
2. Are inferring and reasoning same? Discuss.
3. What are the different representations of the knowledge base?
4. Explain and discuss semantic networks, partitioned semantic networks and frames.
5. Write the following using first order logic
   (i) Some intelligent students study intelligent systems.
   (ii) Every student who opts for intelligent systems is a determined student.
   (iii) There is at least one student of intelligent systems, who does not like the AI assignments.

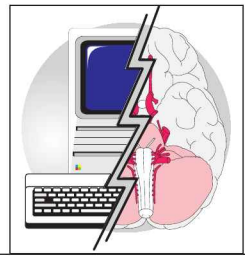
CRITICAL THINKING EXERCISE

1. For the following example, write the KB in propositional logic. Explain how the inference would occur.
   Ram is an IT professional.
   IT professionals are hardworking.
   All hardworking people are wealthy.
   All wealthy people pay heavy income tax.

PROJECT WORK

1. Develop a program to determine the state of wumpus, with the initial states and the knowledge discussed in Figure 7.4. Which type of KB representation will you use and why?
2. Using semantic network, represent different players in Indian Cricket team and their properties. Write a query to determine most prolific batsman on Australian soil using this semantic network.

CHAPTER 8



Uncertain Knowledge and Reasoning

Learning Objectives

- To understand the reasoning process
- To understand the importance of probability theory with uncertain knowledge
- To appreciate the role of probability with reasoning
- To understand the perception and its value in reasoning process
- To appreciate and gain knowledge of making decisions with practical applications where uncertainty exists

INTRODUCTION

Let us begin our discussion on uncertainty with a simple example. Consider a scenario where you need to go to college for a particular lecture that is scheduled at 5 p.m. Let us assume that you have to leave your place x minutes before the lecture. You have a question—will these x minutes help you reach on time? So, you are uncertain about it. You have problems like the information availability could be partial with respect to the road conditions, and you are uncertain about the traffic. Things get more complicated if it starts raining. In addition, your parents do not allow you to take the vehicle. So, the information you have from the environment is uncertain. But still you tend to rely on this incomplete information for the decisions that you need to take. Now, when it comes to decision-making, we need to deal with this uncertain knowledge efficiently and at the same time, we need to reason rationally about this information. However, the uncertainty in knowledge also introduces uncertainty in the reasoning process. To get maximum benefit from this knowledge, it is essential to represent it in a proper way so as to use it in decision-making.

The basic question that comes to our mind is how to represent or describe the knowledge. While taking decisions in our daily life, we take decision on the basis of our ability and

the available uncertain knowledge or known facts. In case of a machine, we embed the intelligence. The decision-making capability of a machine is based on the intelligence, which requires sound reasoning and concise representation of the knowledge. This aids in inferring the machine to act on the environment. For this, it is necessary that the machine has a good reasoning capability and the representation of the knowledge should be such that the machine is able to deduce the inference and act with respect to the environment.

Speaking about the reasoning process in broader terms, it is an act to come to a conclusion from the premises using some methodology. In every knowledge acquisition and reasoning system, the focus lies on logic. Here, logic is the selection of particular actions or rules so as to reach the destination. Thus, we can relate reasoning to be one approach for problem solving. In the logic-building process, where there is uncertainty, it is equally essential to consider actions, where the risks involved are less. The ability to reason governs the intelligence and with uncertain knowledge, it is more complex. Let us get familiar with the terminologies and details in this uncertain environment.

8.1 UNCERTAINTY AND METHODS

We can always say that uncertainty is a fundamental feature of the real world. In order to deal with it intelligently, it is important to apply logic and take appropriate actions. Various factors result into uncertainty. It could be due to omitted data, unavailability of the entire data, and partial availability of events and so on. This uncertainty is handled using the realm of probability theory.

Uncertainty is handled by using various ways. The simplest one is the default or the monotonic logic. Relating to the example discussed in introduction, you would assume about traffic and road conditions, as well as vehicle availability to take decisions. But the assumptions are actually not reasonable. We need to handle the contradictions that arise. We can have a certainty factor that calculates the measure of belief and disbelief. Belief can be represented using probability, and hence, we can apply probability theory. We will describe the parameters and the key features involved in uncertainty before going into its details.

Dempster–Shafer theory is the one that allows for proper distinction between reasoning and decision theory. It uses the concept of belief and plausibility. Belief is referred to as all available evidences, whereas plausibility is all evidences compatible, but not consistent with the hypothesis.

Fuzzy logic and the probability theory are also the methods for representation of uncertainty. The basic idea behind fuzzy logic is that a truth value may be applied partially. Fuzzy logic uses possibilities and assigns a value between 0 and 1 to determine the degree to which it applies. A document containing information about human body definitely belongs to the class of science. But a document that has information about human body, and say the geographical area around is said to partially belong to the class of science. So, with fuzzy logic, we define degree of belongingness over here.

Probability theory assigns a probable value when you have uncertain information. Conditional probabilities capture and represent uncertainties, which involve complex relationships. Here we try to infer on the basis of Bayesian inference. Instead of rules, the relationships among variables are represented. Considering the computation and representation power, Bayesian probability is used for handling uncertainty.

8.2 BAYESIAN PROBABILITY AND BELIEF NETWORK

Probability can be interpreted from two views—subjective and objective. In objective view, it represents the physical probability of the system, whereas in case of subjective, it estimates the probability from the past experience. Hence, it calculates the degree of belief. This belief can change under new evidence. This subjective probability is called *Bayesian probability*. So, we can say that Bayesian probability is the process of using probability for predicting the likelihood of certain events occurring in the future. Since the beliefs are subjective, all the probabilities occurring in the Bayesian probability are conditional.

Bayesian probability is used in many cases. Consider a simple example, where you want to predict whether X person is likely to be selected in the cricket team. In this case, things would be dependent on number of matches he/she has played, whether he/she is an all-rounder or batsman and so on. Using Bayesian probability, we are in the position to model these relationships and infer. Under uncertain scenario, Bayesian probability has a very deep impact in the decision-making. Bayesian probability is viewed as an extension of logic that enables reasoning with uncertain statements.

Let us proceed with the discussion on the basic concepts of probability and the terminologies.

Propositions and Hypothesis

Let us understand what a proposition is. We are already familiar with the propositional logic. Propositions are typically the sentences which are either true or false but not both. Here, in probability, the notion is explained as follows:

The crucial element, here, is the random variable. A *random variable* can be an item whose status is unknown. Every random variable has a domain that determines the values it can take. The domain can fall under any of the categories like Boolean, discrete or continuous. As an example, let out be a variable, which can take value <true, false> for the statement—Is the batsman out?

Hypothesis refers to proposed explanation for an observed phenomenon. It is also referred to as *educated guess*. Typically, it is expressed in the form of if-then. For example, a hypothesis could be 'If a prisoner learns some skilled work while in prison, then he is less likely to commit crime after release'. This is typically called *testing hypothesis*. Similarly, we can have another tested hypothesis like 'A good teacher can increase the student's satisfaction'. Now, in any statement like 'If A occurs then B', A is called the *antecedent* and B is the *consequent*. A is also referred to as *hypothesis*.

Probability

Probability can be defined as the likelihood of some event occurring. Its value lies between 0 and 1, where 1 indicates a common event and a rare event would be close to 0. Event are said to be one or more outcomes.

There are different types of probability, which are as follows:

1. *Classical probability:* It can be termed as the priori theory. In the sense for an event A, the probability of occurence of A is given as follows:

Probability = Number of favourable outcomes resulting in event A/Total number of possible outcomes

2. *Joint probability:* It is the probability of all events occurring together. For events A and B, it can be written as $P(A, B)$.

3. *Marginal probability:* It is an unconditional probability of an event A, irrespective of the occurrence of event B. It is calculated by adding joint probabilities of A over all occurrences of event B. This gives the marginal probability of event A.

4. *Prior probability:* Prior probability or unconditional probability corresponds to belief without any observed data or before the availability of the evidence.

5. *Conditional probability:* It is the probability of some event A, given occurrence of some other event B. It is expressed as $P(A \mid B)$ and mathematically, it can be written as

$$P(A \mid B) = P(A, B)/P(B)$$

In formal definition, we say that the conditional probability of an event A is the probability that the event will occur, given the knowledge that an event B has already occurred. This is written as

$$P(A \mid B): \text{probability of } A \text{ given } B$$

In case, the events are independent, the conditional probability of A given B is simply the probability of A, i.e., $P(A)$.

But if they are not independent, then we define it as the probability of intersection of both the events. This is given as

$$P(A \text{ and } B) = P(B)P(A \mid B)$$

Now, the conditional probability is defined as follows:

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)}$$

6. *Posterior probability:* It is the conditional probability of an event after considering the relevant evidence.

Considering the example given in the previous section, a prior probability can be $P(\text{out}) = \text{true} = 0.3$. This is valid and used when no other information is available like 'batsman was injured or was batting against the best bowler in the world'. Whereas in conditional probability, the posterior probability can be $P(\text{out/injured})$. So, it is the probability of batsman getting out, given he/she is injured on previous ball.

Product Rule

We know that the conditional probability is written as $P(A \mid B) = P(A, B)/P(B)$. The equation can be rewritten as

$$P(A, B) = P(A \mid B)P(B) \tag{8.1}$$

This is called *product rule*. Similarly, the equation for conditional probability for $P(B \mid A)$ will be

$$P(B \mid A) = P(A, B)/P(A) \tag{8.2}$$

Applying the product rule to Eq. (8.1) and Eq. (8.2), we get

$$P(A \mid B) = P(B \mid A)P(A)/P(B) \tag{8.3}$$

This is called *Bayes' theorem*. It is used to infer the probability of hypothesis under observed data or evidence. This is termed as *Bayesian inference*. Let us detail it in the next section.

8.2.1 Bayesian Inference

Bayesian inference is a statistical method to infer for an unknown situation. It involves collecting the evidences that are consistent or inconsistent with a hypothesis. As more and more evidences are gathered, the degree of belief in a hypothesis changes like normal reasoning. With enough evidences, the hypothesis can become true or false. Equation (8.3) can be rewritten as follows:

$$P(H \mid E) = P(E \mid H)P(H)/P(E) \tag{8.4}$$

where E is evidence and H is hypothesis.

Here, P(H) is the prior probability. $P(E)$ is the marginal probability and $P(H \mid E)$ is the conditional probability. It can also be mapped to a cause and effect problem or even to a diagnostic one.

To make it clearer, let us take an example. Let e be the proposition that 'X person has elbow pain'. Let t be the proposition that 'X has tennis elbow'. So, if we want to derive at a diagnosis, we will have the mapping as follows:

P(person having elbow pain) = $P(e)$
P(person having tennis elbow) = $P(t)$
P(person is having tennis elbow who has elbow pain) = $P(t \mid e)$ (to be found out)
P(person is having pain in elbow given that he is having tennis elbow) = $P(e \mid t)$
So, it would be

$$P(t \mid e) = P(e \mid t)P(t)/P(e)$$

So, in Bayesian inference, it uses the initial belief on the hypothesis and estimates the degree of belief as the evidences become available.

8.2.2 Belief Network

A *belief network* or a *Bayesian network* is a probabilistic graphical model that represents a set of random variables and their conditional dependencies through a directed acyclic graph (DAG). In the graph, the nodes represent random variables and the directed edges represent the conditional dependencies among the variables. The dependent node of a node is called *child node*, whereas the one on which it is dependent is the *parent node*. The directed edge points to the child node to represent the dependency. The Parent node is called *cause* and the child node is called *effect*. The nodes that are not connected are independent of each other. Every node is associated with the probability function. It takes values of the parent function and gives the probability of the variable that it represents.

A Bayesian network essentially provides the entire description of a domain. The network is seen as joint probability distribution. Each entry in this probability distribution is represented by the product of the appropriate elements of conditional probability table (CPT).

Consider a hypothetical example to understand the concept. We need to understand the probability of selection of Ram or Shyam with regard to coach. Figure 8.1 represents this in Bayesian network.

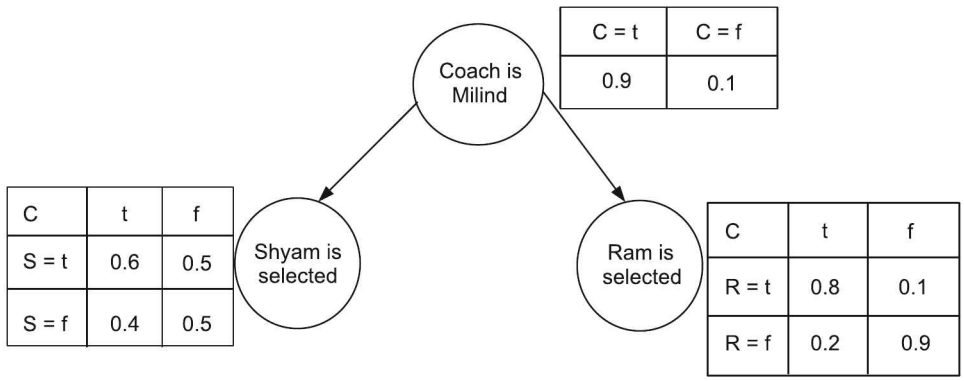


Figure 8.1 Bayesian network.

The node 'Coach is Milind' does not have any parent node. Its value represents the prior probability. This value can be taken from the past data. Here, C = coach, S = Shyam selected, R = Ram selected. Now, we can use this example to draw the inference. In what way can Bayesian be used in this case? Let us understand.

Suppose we want to compute the probability of Ram being selected in the team, then it is computed as follows:

Let $P(C)$ = Probability that the coach is Milind
$P(\text{no } C)$ = Probability that the coach is not Milind

$$P(R) = P(R \mid C) * P(C) + P(R \mid \text{no } C) * P(\text{no } C)$$
$$= (0.8 * 0.9) + (0.1 * 0.1)$$
$$= 0.73$$

Similarly, we can compute the probability that Shyam is selected as follows:

$$P(S) = P(S \mid C) * P(C) + P(S \mid \text{no } C) * P(\text{no } C)$$
$$= (0.6 * 0.9) + (0.5 * 0.1)$$
$$= 0.59$$

These computations are the computations of marginal probabilities. Now, let us apply the Bayes's theorem. Suppose we know that Shyam is selected, but do not know that the coach is Milind, then the computation of the values will be based on the known evidence. Thus, it will be

$$P(C|R) = \frac{P(R|C)*P(C)}{P(R)}$$

$$= \frac{0.8 * 0.9}{0.73}$$

$$= 0.98$$

This gives the probability that the coach is Milind. This information can further be used to decide whether Shyam is selected. This method is known as *propagation*, where with the help of evidence, the probabilities are propagated.

8.3 PROBABILISTIC REASONING

Probabilistic reasoning summarises uncertainties arising in the environment from the different sources in the past and derives probable outcome. The environment may be static or dynamic. The past experiences represent statements or hypothesis. Whenever a new statement is generated on the basis of previous facts, it can also be referred to as *hypothesis*. Let us try to differentiate between logical and probabilistic reasoning. In case of logical reasoning, the hypothesis could be right or wrong. But in case of probabilistic reasoning, the possible truth of the hypothesis is calculated. As discussed in the earlier section, Bayesian inference falls under the probabilistic reasoning, or in other words Bayesian reasoning is also referred to as probabilistic reasoning, where the probability of hypothesis is modified by further data.

We have already discussed about Bayes's theorem. It is given by Eq. (8.4).

$$P(H|E) = P(E|H)P(H)/P(E)$$

where, H is hypothesis and E is evidence. H and E are the multivariate variables. Each atomic variable defines/stands for the properties or attributes in the environment. So, remember that the value of the variable remains fixed during reasoning. The prior probabilities are simplified with the use of Bayesian network. One of the biggest advantages of Bayesian network is the simplicity in calculation of the probabilities with the representation of the large variables.

So, we can always say that the Bayesian network is an important step in inferring for future. But it has some drawbacks. It is feasible in small and medium-sized network. With large and complex network, the number of variables and the relationships are increased, and on the top of it, the CPT calculations exist for each node. So, it is very essential to have these calculations through machine learning algorithms that can automate the process. (Machine learning is discussed in further chapters.) Let us brief about the various algorithms.

In connection with belief networks (BN), it involves two tasks–learning the parameters and learning the structure. In structure learning, the relationships among variables are identified from historical data on the basis of their conditional independence, with the underlying concept to maximise the posterior probability. An exhaustive search process is required to find out the possible relationship that is exponential in the number of variables. In case of parameter learning, the CPT associated with the variable are computed. In case of discrete values, probability values are used, whereas in case of continuous-valued attributes, Gaussian distributions are used as probability values.

Once the BN is built, we can infer. Since the computation is very complex with BN, there are different approaches that are used for inferring, viz. top-down and bottom-up. In top-down, effects are predicted with prior knowledge of causes, whereas in bottom-up, it is from evidence to cause. The most common algorithm that is used here is bucket elimination, where the elimination of unobserved query variables takes place by the calculation of inference equation and its elimination. Markov chain simulation—Markov Chain Monte Carlo (MCMC) algorithm uses random sampling in generation of next states to infer.

This is all about reasoning in static environment that gives us an overview of some of the approaches. But what will happen when the environment changes? Let us take an example that a person is coming late to work and there is traffic jam. Mapping it to our previous hypothesis and evidence, H will be 'There is traffic jam' and E will be 'A person is coming late to work'. The final outcome required is the prediction of 'coming late' when traffic jam exists. Here, it is assumed that the values of evidence and hypothesis are not changing. But what will happen in a dynamic scenario? This is discussed in the next section.

8.4 PROBABILISTIC REASONING OVER TIME

When we talk about agents, we cannot imagine their operation in a static environment. Consider a robot, where states of robot are its different positions at different time. When a robot is moving in a room, its next state is dependent on the previous one and its surroundings (environment) at that time. In case of reasoning for robots, it is very much essential to know the random variables at each time slice, say t. A dynamic environment can be imagined as a collection of small fragments taken at regular interval of time from a larger one. So, at every interval, it has a set of random variables, some of which are observed and some are not. The length of the time slice or the interval is dependent on the problem, and is discrete in nature. Mapping it symbolically, we can say X_t represents the set of unobserved variables at time t and E_t represents the set of observable evidence variables.

An observation at time t is E_t, which has some values e_t. Relating this to the example of traffic, E_t contains an evidence that the person is late and X_t contains whether there is a traffic jam. Further, we can say that X_1, X_2, X_3 are the unobserved state variables and E_1, E_2, E_3 are the evidence variables representing the case of whether the person is late or not at the different time slices. Once we have the variables defined, it is important to model the environment. We can represent the relationship graphically following the causal order. So, as cause precedes the effect, the variables can be placed in the causal order for each time slice, as it is done in BN. Hence, in dynamic environment, dynamic BN is used, where the nodes are random variables over the time t. Since each time slice has its own state and evidence, the relationships are more unbound and complex. Further, every variable in every time slice has its own CPT, so large CPTs could be created, where there might exist a large number of parents as well. In order to limit the CPT, it is assumed that the law which governs the process of change in environment does not change.

Markov Assumption

Let us first try to understand and resolve the problem of unbounded number of parents. Parents of every node are limited to a finite number by the Markov assumption. Markov assumption states that any state depends on the finite history of states. Processes satisfying this assumption were studied in depth by A. A. Markov and are called *Markov process* or *Markov chains*. Simplest of all is the first order Markov process. In this case the current state depends only on the previous state.

So, the state that is evolved later is contained in conditional probability distribution $P(X_t | X_{t-1})$.

This is also referred to as *transition model* for the first order process. Hence, the corresponding conditional independence assertions states that

$$P(X_t | X_{0:t-1}) = P(X_t | X_{t-1}) \tag{8.5}$$

The BN structure for first order Markov process is depicted in Figure 8.2. To add further, we also assume that the evidence E_t depends only on the current state. So, we have

$$P(E_t | X_{0:t}, E_{0:t-1}) = P(E_t | X_t) \tag{8.6}$$

This conditional distribution $P(E_t | X_t)$ is referred to as *sensor model.*

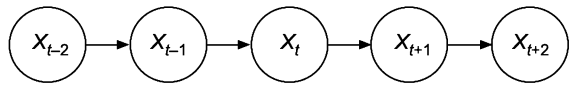


Figure 8.2 First order Markov process.

Further, the second order Markov process states that the current state depends on the previous two states. Its equivalent probability distribution is $P(X_t | X_{t-2}\ X_{t-1})$ and its representation is depicted in Figure 8.3.

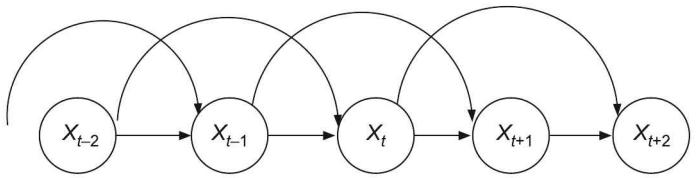


Figure 8.3 Second order Markov process.

Accordingly, for the example we have discussed, the conditional distributions are shown in Figure 8.4. $P(\text{traffic}_t | \text{traffic}_{t-1})$ represents the transition model and the sensor model is $P(\text{Late}_t | \text{traffic}_t)$. The arrows indicate the dependencies.

To summarise, the joint distribution over all the variables at any time t is given as

$$P(X_0, X_1, ..., X_t, E_0, E_1, ..., E_t) = P(X_0) \Pi_{i=1}^{t} P(X_i, X_{i-1}) P(E_i | X_i) \tag{8.7}$$

Here, $P(X_0)$ is the prior probability of state at time $t = 0$. The independent assumptions correspond to a Bayesian structure. But the selection of the Markov process is dependent on the problem. If the prediction is not accurate, the order can be changed.

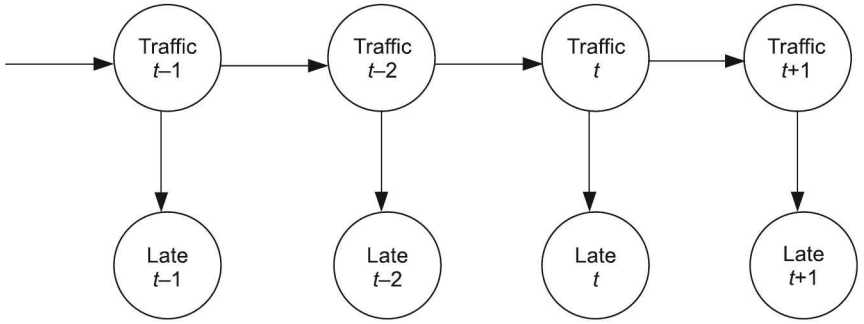


Figure 8.4 Dynamic Bayesian network.

Models

In the dynamic environment, the network model and the inference are carried out using three different models—*hidden Markov model (HMM), Kalman filter* and *dynamic Bayesian network* (DBN). We have already discussed about DBN. Let us proceed with hidden Markov model. In many cases, all the variables in the problem domain are not visible or observable. In such cases, instead of DBN, HMM is used. HMM describes the temporal probabilistic model, where the state of the process is described by a single discrete structure. The example mentioned in introduction can be considered as HMM due to the presence of single-state variable that is traffic.

Kalman Filter assumes that the values of the random variable are continuous and its associated probability distribution is Gaussian. To explain it, we consider an environment, where in crowd, a familiar face is seen by you. You scratch your head to recollect where you had met that person earlier or whether you know him. You still try further to locate that face in the crowd so that you do not lose the track and are able to infer. This problem is an inference, where a noisy model exists.

Inferring

In the dynamic environment, the basic inferring tasks are described here. The computation model is selected based on the inferring required by the application.

1. *Filtering:* It is also referred to as *monitoring*. It is the task of computing the posterior probability of the current state or the current belief state. So, we compute $P(X_t \mid E_1 E_2 \ldots E_t)$. We take into consideration that evidence arrives in a stream starting at $t = 1$. The reason behind the name filtering is that the noise in the past data can be eliminated by computing the current belief state. At the same time, it is also called monitoring because in some cases, the state variables can be updated over a period of time. In the example of traffic jam mentioned in introduction, 'coming late' is monitored over a period of time. So, we are computing the probability of traffic jam.

2. *Prediction:* Inference is drawn here to predict the future. So, we compute the posterior probability, given all evidences for the future state. $P(X_{t+k} \mid E_1 E_2 \ldots E_t)$. We calculate it till some value of $k \mid k > 0$ is obtained. With respect to the example, we predict the probability of traffic jam, given some observation about 'coming late'.

3. *Smoothing:* It provides estimate of past from all evidences upto present. It is equivalent to computation of $P(X_k | E_1 E_2 \ldots E_t)$, where, $t > k > 0$. Coming back to the example, we want to predict the traffic jam that has occurred last week, given all the observations of 'coming late' till today.

4. *Most likely explanation:* In this approach, the objective is to find out the most likely sequence of the states that would be generated, given all the evidences. So, we compute argument-max $\text{argmax}_{X_{1:t}} P(X_{1:t} | E_{1:t})$. This is also referred to as *Viterbi decoding*. So, with the example of traffic jam, we can say that if a person comes late for consecutive 4–5 days and on 6th day, he comes on time, we can give an explanation that there was a traffic jam earlier and later on, there is not. Algorithms with this task are useful for speech recognition, where the most likely sequence of works can be found.

So, learning is a major task to generate any model for the problems in the dynamic environment. The goal of learning is to learn parameters along with the probability distribution. Here, the expectation maximisation (EM) method is most commonly used. In this method of learning, inferences are drawn on set of parameters to perform fixed interval smoothing, and further, these estimates are used to update the model. The updated model provides new estimates and the process of iteration continues till convergence, in the sense, till posterior probability reaches maxima. Hence, learning depends on estimation. Learning in DBN is to learn for intra-slice relationship, which is a DAG as well as inter-slice. Each node in time slice t has a parent in time slice t–1. As discussed earlier, we are assuming that the variables are same over all time slices. So, the problem reduces to feature selection from the set of variables as the transition occurs over time slices. When all the variables are observed, we can use the standard feature selection algorithms like forward and backward step-wise selection, and leaps and bound. In case, when the information is partially available, EM is used to learn the structure of the model.

8.5 FORWARD AND BACKWARD REASONING

Consider any reasoning system. It typically constitutes of a rule base and an engine to search the rule base for appropriate rules under some available facts. We have already discussed about it in the previous chapter. The question here is how to apply it in case of uncertainty.

For many problems, it is just not possible to list out all the possible options in advance and let the system select the correct one. While discussing about uncertainty with forward and backward reasoning, make a note that the terms backward chaining/backward reasoning and forward reasoning/forward chaining are used.

Forward chaining, as the name suggests, starts searching a solution path from the initial state to the goal state. It matches with the rules and performs action. In this process, it might generate some conclusions that are not applicable. Generally, under uncertainty, if we require multiple options, then this type of reasoning is used. This reasoning has proved to be efficient when the available facts are less. Suppose we have some set of rules $A \rightarrow D$, $(C, D) \rightarrow F$, $(F, B) \rightarrow Z$ for the facts A, B, C. With forward chaining, we get the search tree as depicted in Figure 8.5. The search tree generated is an OR tree.

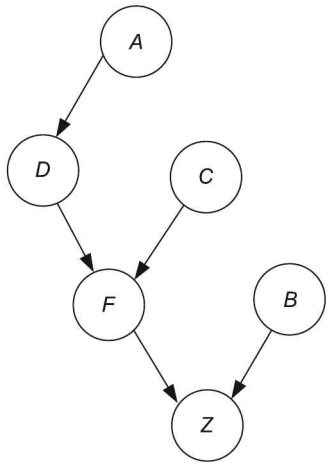


Figure 8.5 Forward chaining.

Backward chaining is goal-driven. It is the process of searching for solution path connecting goal. States are the combinations of the required conclusions or goals, whereas the transitions are defined by rules. Each rule is said to be the method of generating conclusions from the required conclusions. In this case, the inference starts with the goal state, with searching action rules to establish the sub-goals. Here, the node branches in different ways. Any rule that satisfies goal represents an alternative branch. Since more than one condition is represented, it might generate (again uncertainty) multiple sub-goals. Here, the branches from any node are divided into groups of AND branches. Again, each group is an alternative branch or an OR node. Hence, backward chaining represents AND/OR tree. For the same example, given in the case of forward chaining, the tree is shown in Figure 8.6.

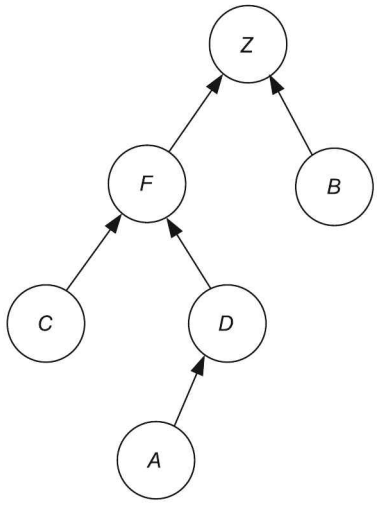


Figure 8.6 Backward chaining.

Under uncertainty, we are not at all aware of the circumstances, and things can change dynamically, yet selection of the best alternative from the available knowledge is the most efficient method. With a known goal, backward reasoning is efficient, but when it comes to number of ways to derive at conclusion, forward reasoning is useful.

8.6 PERCEPTION

We have discussed about uncertainty with reasoning. We can say that an agent's operation is governed by the uncertain environment and it needs to act upon dynamically. But to act so, it is required that it gets the knowledge about the surroundings. This is where perception comes into picture. So, it is the perception that is responsible to make this available. Let us discuss about it in detail.

Reasoning, learning, perception and acting in the complex environment form the basis for intelligent behaviour. *Perception* can be well defined as the ability to sense and interpret. Hence, it deals with the simulation of intelligent behaviour of the system. In real world, in order to automate the control of machines, it is necessary that the machine acts on the situations dynamically depending on the environment. So, real-time control is the biggest challenge of controlling systems. Hence, under this scenario, real-time decisions play a very critical role in running the automation system. The system takes decision under uncertainty as the environment changes and applies reasoning prior to the action. In case of machine, perception is the ability to sense the images, sounds and interpret their sources and properties.

Let us take an example of robot. It is an agent that operates in the environment dynamically. While moving from one place to another, it uses its vision to locate the obstacles and accordingly decides its path. Here, different sensors like the camera, microphone and so on collect the information about the obstacles. So, in turn, we are moving towards vision and speech recognition. We can now define vision or say *computer vision* as the ability to analyse visual input, whereas with speech recognition, it is the analysis of voices, speeches and interpreting them to act upon.

To begin with vision, there is a need for accurate perception systems to open broader areas of computer applications. Since, vision is all about image, let us have an idea about how things work here. When an image is generated in any state, a question comes to our mind—what is required from it? This is wholly dependent on the type of application for which one wants to have the data perceived. There are fundamental operations that one would categorise to exploit the knowledge from it. These are classification or pattern recognition, signal processing, measurement analysis and object location. Even though the problems like object identification can be decomposed into pattern matching, there are still issues like 2D–3D mapping, number of objects existing and so on. Ambiguity in the image, amount of knowledge required for the data and the presence of noise add up to more problems. Extraction of edges and regions, inferring 3D orientations along with image texture help in the entire process of knowledge gain. Still to conclude, any approach for vision is governed by the knowledge representation with respect to the application.

To summarise, we can say that *machine perception* is the ability of the machine to read input from the sensors, reason them and act accordingly. Machine interacts with the dynamic environment. It is very likely that partial information is collected, and hence,

the probability theory is the best suited approach for reasoning in machine perception. As discussed in the previous sections, we can have DBN and other models used for the same. Chapter 16 discusses these concepts in elaborated form.

8.7 MAKING SIMPLE DECISIONS

Decision-making is an intelligent process of selecting an action from a set of alternatives. In this process, uncertainty about the alternatives is minimised to make a final choice. The process does not eliminate the uncertainty, but makes a decision that allows to select alternative with the highest probability from several possible alternatives.

We can always say that probabilistic reasoning is the core of the decision-making. It can handle uncertainty in the real world by means of probabilistic distribution. With Bayesian, it is quiet straightforward to calculate the probability value of any variable attached in the problem space. But a decision-maker combines the preferences that are expressed by utilities along with the probabilities to make decisions.

We begin with some basic concepts about utility and then proceed to the decisions. Utility theory has preferences. In case of our previous example of reaching on time for lecture (discussed at the beginning of the chapter), though we have many options at hand that make it possible to reach on time, you cannot go and wait outside the college for the entire day! So, the point is to have preferences. As stated in the utility theory, every state is somewhat useful or rather has a degree of usefulness, and preference is given to the ones with higher utility. Hence, a decision theory can be represented as:

Decision Theory = Probability theory + Utility theory

Details about it are discussed in Chapter 13. The basic idea of decision theory is that any agent is said to be rational iff it selects the action resulting into highest expected utility that is averaged over all possible outcomes of the action. This principle is called *maximum expected utility (MEU).*

We proceed with the general mechanism of making rational decisions—influence diagrams. It is also referred to as decision networks that basically combines Bayesian and has additional nodes indicating utility or actions.

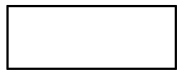
Influence Diagram (Decision network)

An influence diagram (ID) is a directed graphical representation of the decision-maker's preferences with respect to the real-world uncertainty and sequence of possible decisions. An ID, or more precisely, a decision network is a representation of the information regarding the agent's current state, actions and the possible outcomes. Let discuss types of node used for the representation of decision network.

Chance node: A chance node represents random variables. It is similar to the way we have in BN. It represents the uncertain variables. Each chance node has probability values that influence the decisions. It is represented in the following shape.

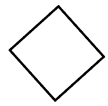
Chance mode

Decision node: It is at this node that the decision-maker has to make a choice of actions. Every decision node is attached to the chance nodes, which in turn, influence the decisions. A decision node is represented in the following shape.



Decision node

Utility node: It represents an agent's utility function. Its parents are the variables that describe the outcome that directly affects the utility. It is represented in the following shape.



Utility node

To get more clarity, we start with the details and mapping of utility function.

Let $A = (a_1, a_2, ..., a_n)$ be the set of actions and $H = (h_1, h_2, ... h_n)$ represents the set of variables that influence the actions. $U(A, H)$ represents a utility table that holds the values for each action item and the variable that it is associated with. The decision is made by maximising the utility values associated with the action. Mathematically, it can be represented as

$$EU(a_i) = \sum_H U(a_i)p(H \mid a_i) \tag{8.8}$$

$EU(a_i)$ represents the expected utility of action a_i. $p(H \mid a_i)$ is the conditional probability of variables h_i, where $h_i \in H$, given action a_i is executed. This conditional probability is calculated from CPT while traversing the BN of these variables. We explain it with the traditional example of the umbrella by Jensen. In this example, the decision-maker wants to take decision whether to carry umbrella before leaving the house. So, the decision node here is the umbrella. Uncertainty is about the weather conditions. So, we can say weather conditions and weather forecast are the uncertain variables that are represented as chance nodes. Utility is the satisfaction in carrying the umbrella. Figure 8.7 represents the decision network for the example.

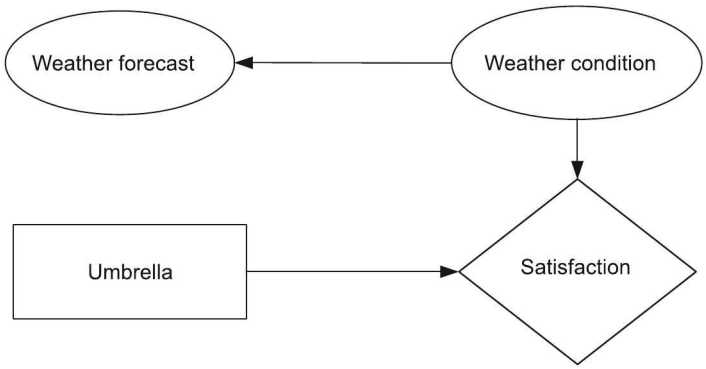


Figure 8.7 Influence diagram for umbrella example.

Let us assume typical CPT for chance nodes.

For weather condition, the prior probability distribution is shown in Table 8.1.

TABLE 8.1 Prior Probability

Weather condition	
Rain	0.7
Sun	0.3

Similarly for weather forecast, this corresponds to posterior probability, since it is influenced by weather condition, as shown in Table 8.2.

TABLE 8.2 Posterior Probability

Forecast	*Weather condition*	
	Rain	*Sun*
Good	0.6	0.2
Bad	0.4	0.8

To take decisions about taking umbrella, a utility table is to be defined. This is shown in Table 8.3.

TABLE 8.3 Utility Table

Umbrella	*Weather*	*Satisfaction*
Carry	Rain	90
Carry	Sun	20
Don't Carry	Rain	15
Don't Carry	Sun	10

Using Eq. (8.8), we compute and fill the utility and select the one that maximises it. Let us say we want to compute the expected utility (EU) for not carrying the umbrella. In such a case, the computations will be

EU (don't carry the umbrella) = (0.7 * 15) + (0.3 * 10) = 13.5

Similarly, EU(carry the umbrella) = (0.7 * 90) + (0.3 * 20) = 69

Thus, the maximum utility is the selection of an action that has greater EU. Hence, it is 'carry the umbrella'.

Further, the EU to carry the umbrella, given that the forecast is good, is computed as follows:

$$\text{EU}(\text{carry} \mid \text{good weather}) = \sum_{w} P(w \mid \text{good}) U(\text{carry}, w)$$

First, let us compute P(Weather condition | Forecast = good). It is shown in table 8.4 below:

TABLE 8.4 Probability Table for Weather Condition

Weather condition | *Forecast = good*	
Rain	0.7 * 0.6 = 0.42
Sun	0.3 * 0.2 = 0.06

EU(carry the umbrella | forecast=good) = 0.42 * 90 + 0.06 * 20 = 39

Similarly, EU(don't carry the umbrella | forecast = good) = 0.42 * 15 + 0.06 * 10 = 6.9

Thus, an optimal decision is to carry the umbrella, even though the forecast is good.

Selection of any action is dependent on the decision network evaluation. The algorithm for making the decisions is as follows:

1. Set the evidence variable for the current state of evaluation.
2. For every possible value of the decision node that it can take up,
   (a) Set and assign the decision node to the value.
   (b) Compute the posterior probabilities (we can compute this using Bayes's theorem).
   (c) Compute the resulting utilities—expected ones for the action.
3. Select an action with the highest utility.

8.8 MAKING COMPLEX DECISIONS

Often, decision-making becomes hard or difficult owing to the complexity of the problem. In such situations, we take up sequence of actions to accomplish the goal unlike the simple decision-making. When we talk about sequence of actions, it means that the decision-maker has to interact with the environment repeatedly for every decision. Typically, a decision-maker observes the environment in order to take decisions and then acts. Future information is also required for it. So, this process continues till the goal is achieved. The only reason to observe is to gain information for the future action.

In a simple scenario, it has one decision variable and set of action items. Whereas, in case of complex one, the set of decision variable D corresponds to $D = \{D_1, D_2, ..., D_n\}$ and each D_i has a set of alternative actions associated with it. In simple decision theory, utility is assigned to each action's outcome and in complex scenario, each of the multiple actions are associated with utility. Some of these actions may generate risk, while some may generate reward. So, the final decision is the optimal balance of the reward and the risk. There can be a possibility that single or multiple decision-makers are involved in the complex process.

The complexity of decision-making problem is dependent on the environment in which the agent is operating. We are already familiar with the different environments in which the agent operates like deterministic, dynamic, episodic and so on. In case of complex decision-making, it could be the combination of one or more of these environments. A variety of industrial applications exist, where sequences of decisions are to be made to achieve the business goal. Let us take an example of a manufacturing industry. Here, factory control, resource planning, inventory control are planned and executed to maintain the production level. In any industrial application, planning and control is a critical problem. These critical problems can be modelled as sequential decision-making problems. Problem environments for sequential decision-making are mainly of two types—competitive environments like

in games, e.g. chess, checkers, etc. and co-operative like robots, portfolio management, inventory management and so on. In co-operative environment, agents ensure that the individual actions contribute to optimise the team activity, whereas in competitive environment, each agent maximises its own actions and minimises others' actions to win the game. Let us take an example of a retail shop. Assume that in a particular area, there are a number of retail shops, and each tries to improve its turnover. To achieve this, each one starts reducing the prices or gives promotional offers without knowing what others are doing. At the end, some of them gain few customers, while some loose. This is a typical competitive and non-co-operative environment. In a co-operative environment the action of agents are not hidden. Consider an example of two cars coming from opposite directions in a narrow hilly road. Here, the drivers have two options—to continue to move in the same direction, meet at some point and then take reverse, and to let the other go. The goal for the drivers is same, so they signal each other before taking any action.

Thus, we can say that simple decision-making is the application of static or dynamic decision-making, whereas sequential decision-making inherits all the problems and behaviour of static and dynamic decision-making. The static and dynamic probabilistic reasonings are used for decision-making and solving process. But if the decision theory of simple decision-making is extended to the problem of sequential one, computational complexity in the problems of real world increases substantially.

In any state, it is the agent who needs to interact with the environment prior to taking any action, and its probability to reach the outcome is governed by the previous states. This iteration makes reasoning complex and expensive one over long chain of probabilities. To make this simple, the sequential decision-making problem is solved in Markovian environment.

In general, a sequential problem is formulated as state-space model. The problem starts with initial state, say S_0. With every action, the state is changed and the goal is achieved through a set of states say, $S = \{S_0, S_1, S_2, ..., S_n\}$ that are defined over time step t. A transition model also comes into picture $T(s, a, s')$:$P(S_t \mid S_{t-1}, a_{t-1})$ that specifies the probability of reaching s' from the state s, where action a is performed. Each state is associated with a utility function $u(s)$ and reward $R(s)$. This value is equivalent to reward, which may be positive or negative. Negative reward can be treated as risk. An agent always tries to maximise the rewards. This formulation is also referred to as *Markov decision process* (MDP). Figure 8.8 depicts the decision network.

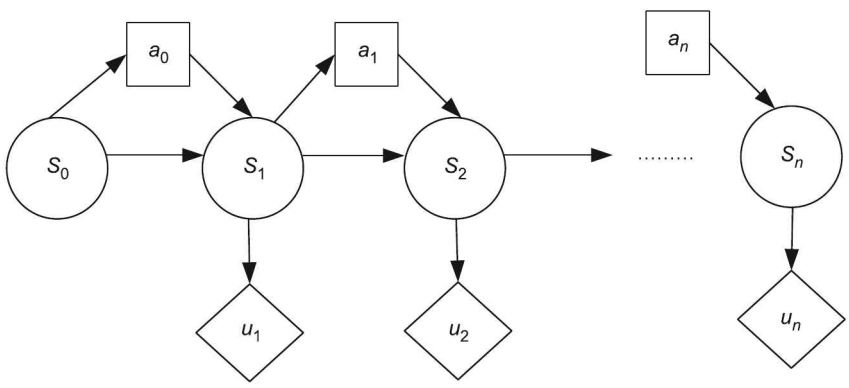


Figure 8.8 Markov decision diagram.

The action at each state is selected from a set of actions. The choice of action at time step t is called policy π and is denoted as $\pi(s)$: policy for state s.

In Markovian environment, the assumption in the state transition probability depends only on the finite number of parents. The first order Markov process is the simplest one, but in decision-making process, values for future action are also considered. There are two types of solutions here—finite horizon and infinite horizon. In *finite horizon*, the utility values are considered for n steps ahead, whereas in case of *infinite horizon*, they are considered for all the steps in the problem domain. Generally, infinite horizon is easy to deal with considering the fact that policy does not change with time and there are no stopping criteria. A utility value for future state is like ₹ 1000 is more valuable today than after a year. Then, the utility value for future states will be less. Hence, a discount factor is used while extrapolating these values. Utility over all states is

$$U(S_0, S_1, S_3, ..., S_n) = u(S_0) + \gamma u(S_1) + \gamma^2 u(S_2) + ... \tag{8.9}$$

where γ is discount factor that lies between [0, 1].

If short-term reward is considered, then low value of γ is used and for long-term, a large value is used. Further, the expected utility of all the states is given by

$$\text{EU}(S) = \sum t\gamma^t P(S_t | S_{t-1}) u(S_t) \tag{8.10}$$

To infer, a policy with the highest expected utility is the optimal policy. Once optimal policy is reached, corresponding actions at each state are selected as the best actions. Let us discuss about the algorithms for finding out these policies in the next section.

Value Iteration

In this step, the expected utility at each step is maximised and the best action is selected. This process is repeated over all the states. The decision is optimised in reverse order, i.e., from the goal state to the initial state. So, to begin, each state is initially assigned with some value and then these values are updated in reverse order.

The $U(S)$ lets selection of action that maximises the expected utility of subsequent state-policy*$(S) = \max_a \sum_{S'} M^a_{SS'} \cdot U(S')$

where $M^a_{SS'}$ is probability of reaching to state S' from S with action a and $U(S)$ is utility of S'.

If this utility is expected sum of discounted rewards R, then the utility is expressed as

$$U(S) = R(S) + \gamma \max a \sum_{S'} M^a_{SS'} U(S') \tag{8.11}$$

Equation (8.11) is *Bellman's equation*. It is now used to compute the utility iteratively, which forms the basis of value iteration. The iteration step is now represented as

$$U_{i+1}(S) = R(S) + \gamma \max a \sum_{S'} M^a_{SS'} U_i(S') \tag{8.12}$$

The algorithmic steps are as follows:

Input: M and R
  U and U' for state S initialised to zero.
do:
  $U = U'$ and delta = 0
  for each state S:
    $U'(S) = R(S) + \gamma \max_a \sum_{S'} M^a_{SS'} U(S')$
    if $(U'(S) - U(S)) >$ delta then
      delta = $(U'(S) - U(S))$
until delta < err$(1 - \gamma)/\gamma$
return U.

So, for infinite horizon, the same best action can be chosen. Here, time is not important, so even stationary policies can be adopted. If we replace γ in the Bellman's equation by γ^k for large k, reward tends to zero or reward is insignificant. So, in Eq. (8.11), the expected utility for k number of steps is considered for value iteration. Policy found at kth iteration is used in the next steps.

The time complexities of value iteration are of the order $O(k|A||S|^2)$. Remember, the complexity increases with the number of random variables in action A and state variable S. This can be handled using dynamic decision network (DDN) with the assumption of conditional independence of random variables.

Policy Iteration

In this approach, instead of state utilities, the policy is searched at each step. This iteration is carried out in two steps. In first step, policy is evaluated for each state by calculating utility value and then policy is improved by selecting action that maximises the expected utility of that state.

The algorithm is as follows:

Input: M and R
  U and U' for state S initialised to zero
Policy: initially random
  while (unchanged = true)
  U = evaluation policy (policy, U, M, R)
    unchanged = true
    for each state S:
if $\max_a \sum_{S'} M^a_{SS'} U(S') > \sum_{S'} M^{\text{Policy(s)}}_{SS'} U(S')$ then

Policy $(S) = \max_a \sum_{S'} M^a_{SS'} U(S')$

unchanged = false

In this section, we have seen that policy optimisation is important in making complex decisions. Optimal policy has to be found through the iterations over the states and when the model of the environment is known. State transition model $T(s, a, s'): P(S_t | S_{t-1}, a_{t-1})$,

utility or the reward values are integral part of the environment, but in the real world, these models may not be readily available. What is available are the states and the actions. In such scenarios, the approach of machine learning is used to generate the model of environment.

8.9 OTHER TECHNIQUES IN UNCERTAINTY AND REASONING PROCESS

We now have a look at other approaches concerned with uncertainty. With a detailed study on the decision process, the Bayesian probability and its role in uncertainty, we consider other aspects in the reasoning process and new ventures for handling the knowledge along with understanding the benefits of appropriate use of the knowledge.

8.9.1 Non-monotonic Reasoning

Before we begin with non-monotonic reasoning, let us understand what monotonic reasoning is. The predicate logic or the inferences that we work on are under the category of monotonic reasoning. In the sense, if we enlarge the set of axioms, things won't change. In simple words, we call a logic to be monotonic if the truth of the proposition does not change even when new information, i.e., axioms are available or added.

The reason to have non-monotonic reasoning is again uncertainty. The knowledge about the world is incomplete, hence it leads to non-monotonic reasoning. Monotonic reasoning falls short of the following:

1. Anticipation of outcomes.
2. Assumptions about things and
3. Drawing conclusions to plan further.

In non-monotonic reasoning, the truth of the proposition can change when new information is available. Default reasoning is the most common form of non-monotonic reasoning. We draw conclusions based on what is most likely to be true. The two approaches of non-monotonic reasoning are non-monotonic logic and default logic.

Non-monotonic Logic

The non-monotonic logic is basically an extension of first order predicate logic. This is done in order to accommodate modal operator. The reason behind this is to allow consistency.

For example, consider the following knowledge base:

Typically, engineering students can write C programs.
First year students cannot write C program.
Shyam is an engineering student.

Can we conclude that Shyam can write C program? It is plausible to conclude that. But can we say Shyam is first year student? The previous conclusion is not valid and needs to be retracted. Hence, a new conclusion holds. Remember, different interpretations give rise to different non-monotonic logics.

Assumptions: A basic understanding is that only positive information available is represented here. Negative information is not explicitly represented. If the positive fact

does not appear in the knowledge base, it is assumed that the negative one holds. This is referred to as *closed world assumption.*

So, we are proving consistency. To show that a fact X is true, attempt to prove $\neg X$. If we fail, we can conclude P is consistent.

Default Logic

Default logic basically introduces new inference rule. It extends the classical logic by non-standard inference rules.

Considering the same examples, we can say that

$$\frac{\alpha(X):\beta(X)}{\gamma(X)}$$

Here, $\alpha(X)$ is prerequisite, $\beta(X)$ is justification and $\gamma(X)$ is consequent.

This can be interpreted as if X is an engineering student and we can consistently assume that engineering students can write C programs, then we infer that X can write C programs.

8.9.2 Data Mining

Data mining in a broader sense, is about analysis of the data to extract some meaningful information. This information could be related to profits, making business investments, helping in applying new strategies and so on. A lot of data mining tools exist today in the market and have proved to be a boon. It handles the large-scale information and performs the analysis. While doing so, the knowledge representations and the warehouses have a significant impact. Uncertainty plays a key role in impacting the decision. This is affected to a great extent when forecasting is required in terms of business strategies to be employed or any other important decision is to be taken. The way knowledge is grouped, say classes or clusters influences the process. Moreover, the relationships, associations and the patterns should also be in place to prove the mining effective.

8.9.3 Fuzzy Logic

The fuzzy logic basically defines the degree of truth. Unlike Boolean values, it defines the degree of membership. It takes the values of 0 and 1 as the extreme cases of truth, but also considers in between states. Fuzzy is actually the way we think. Fuzzy logic can be viewed as the way reasoning process works. Very simple examples of fuzzy logic can be as follows: Seema is having high fever. She is slightly tall. Here, we are not exactly specifying the illness or the height. It is the notion of fuzzy that is put forth in these cases. So, there could be totally different sets which are defined for extreme fever as well as for high fever. This is discussed in detail in Chapter 10.

8.9.4 Knowledge Engineering

It is the process of structuring, formalising and operationalising the information and the knowledge contained in the problem domain. This is required so that we are able to get the

desired results. It is the process of eliciting knowledge. When there is a complex knowledge, it is difficult to have proper representations. There can be difference in the sources from where the information is made available. To add further, there can be representations of varied types as well. These all things account to more complexities in the engineering process. There is a need to have proper knowledge engineering. MYCIN is an example of knowledge-based systems (KBS). For past few years, modelling frameworks that have become most popular are model-based incremental knowledge engineering (MIKE) and KADS. Figure 8.9 depicts process followed by KBS.

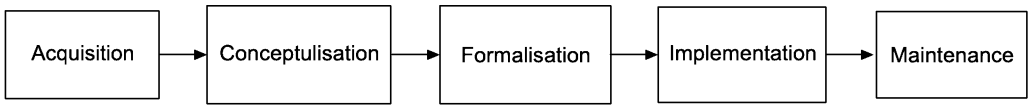


Figure 8.9 Process followed by KBS.

8.9.5 Ontological Engineering

Ontologies describe domain knowledge in a generic way. They provide agreed understanding of the domain. We are aware that the problem-solving methods describe the reasoning process of a KBS. Ontology defines the terms as well as the rules to combine the terms and relations. Ontological engineering basically studies the methods for building the ontologies.

Representations of the abstract concepts of the actions and time are the examples of ontological engineering. It is very closely related to KBS. In fact, the modelling framework used in KBS is influenced by ontologies.

In ontologies, organisation of the concepts takes place. They comprise the relations for the concepts with their sub-classes, and the mapping for the functions of the concepts. The instances and axioms too form the part of the ontology. But how can we build such taxonomy for the concepts? It is formed by identifying the sub-classes or the partitions with regard to the concept, whose ontology is to be built. Even the ISA sort of relations can prove to be useful in building them.

To understand the concept, let us take an example. Generally, when we use any search engine, a personalised search can be made possible by the use of ontologies. This can be used when the user enters some new concepts and a search is required on it. An already built ontology can be proved useful in this search to get the related searches. Further, the built ontology can also be refined based on the user queries.

8.9.6 Dempster–Shafer Theory

Evidential logic is based on the Dempster–Shafer theory, which is based on the beliefs. It is rather the degree of belief that is warranted when given the evidence. It comprises two propositions—plausibility and belief. To each of them, an interval is assigned in which the degree of belief lies. Belief naturally has its value between 0 (no evidence) and 1 (certain). Similarly, for plausibility, the value is also 0 to 1, indicating the extent to which the evidence favours.

Plausibility is defined as follows:

$$PL(p) = 1 - BEL(\neg p)$$

where, p is a particular statement or a set of parameters.

It is just not the belief that helps us in decisions. For some hypothesis where we have no information, we can say that the likelihood lies in the range of 0 to 1. But as information gets available, the interval becomes more precise. This is unlike the Bayes's theorem, where the prior probabilities are already determined.

Let us take an example to understand the details of this theory. Here, the set of conclusions is represented as follows:

$$\Theta = \{\theta_1, \theta_2, \theta_3, \ldots, \theta_n\}$$

where θ_1 to θ_n are conclusions.

This theory is concerned with the evidences that support the conclusions, rather a subset among the conclusion's set. There is a notion of power set (we have studied this in set theory). The power set comprises all possible subsets of Θ.

Let us take an example. To reach any location, many parameters are involved, representing Θ set = {traffic_jam, signal_not_working, bus_breakdown, rain}.

For some given conditions, let us assume we want to predict if we can reach on time or not. Here, Dempster–Shafer theory helps us in handling this by making the use of hypothesis. By this, we mean that using a density function that is probable helps us in making decision. Let us call this function to be m. Now, the function $m(s)$ measures the amount of belief. If theta has n elements, then there are 2^n subsets. We need to assign m values so that all the subsets equal to 1. This $m(s)$ or mass $m(s)$ represents proportion of relevant and available evidence that supports s in the power set. Let us take three parameters out of the above to form our Θ = {T, R, B} (T = Traffic, R = Rain, B = Bus breakdown). The mass or rather the probabilities are set based on some observations. Let these observations be taken from a person who has seen that the signals are not working. $P\{\} = 0$. Let the following values represent them for the said parameters (Table 8.5).

TABLE 8.5 Mass, Belief and Plausibility Values

	m(s)	*Belief*	*Plausibility*
{*T*}	0.1	0.1	0.4
{*B*}	0.2	0.2	0.7
{*R*}	0.1	0.1	0.6
{*T*, *R*}	0.1	0.4	0.8
{*B*, *R*}	0.3	0.7	0.9
{*T*, *B*}	0.1	0.4	0.9
{*T*, *B*, *R*}	0.1	1.0	1.0

As discussed earlier, the probabilities or the m values are based on some observations. Now let us discuss the computations of belief. For example, $BEL\{T, R\} = m(T) + m(R) +$

$m(T, R)$. Hence, it comes to 0.1 + 0.2 + 0.1 = 0.4. In the same way, other computations are done. In case of plausibility computations, the value of PL(s) is computed as the sum of m values that have intersection with s. So, for PL{T, R}, it is given as $m(T)$ + $m(R)$ + $m(T, R)$ + $m(B, R)$ + $m(T, B)$ + $m(T, B, R)$ = 0.8. In the same manner, other computations are done. The gap between the belief and plausibility is regarded as uncertainty. PL(s) can also be defined as follows:

PL(s) = 1 – DISBEL(s). Hence, we can compute disbelief as DISBEL(T) = 1 – 0.4 = 0.6 and so on. Further, the theory states that we can combine the beliefs. Assume some independent probability distribution or the m value is being available. That is, new values are available for the conditions from some other person who watches a policeman controlling the traffic. Under such a scenario, the rule states that

$$m_{\text{new}} = \frac{\sum_{P \cap Q = R} m_1(P) \cdot m_2(Q)}{1 - \sum_{P \cap Q \neq \phi} m_1(P) \cdot m_2(Q)}$$

where, m_1 and m_2 are the probability functions over Θ. P and Q are the combination of pairs that are subset of Θ.

With this, we can compute new m value. Suppose you get some more set of information about T, B, R from some other source. In that case, the new values for m can be computed for the conclusions. For example, we get new information from the new source as shown below (Table 8.6).

TABLE 8.6

	$m_2(s)$
{T}	0.3
{B}	0.1
{R}	0.1
{T, R}	0.2
{B, R}	0.1
{T, B}	0.1
{T, B, R}	0.1

The computation for $m_{\text{new}}(T, R)$ is shown below. We refer to the previous observations as m_1.

$$m_{\text{new}}(T, R) = [(0.1 * 0.2) + (0.1 * 0.1) + (0.2 * 0.1)]/[1 - (0.31)]$$

$$= [0.02 + 0.01 + 0.02]/[0.69] = 0.05/0.69$$

Thus, we can compute new beliefs and plausibility for the new available information and the evidence that is acquired. The main advantage of using this theory is that, as we have discussed earlier, this adds to our KB, thereby giving us evidence for further inferring. The theory assists in making the beliefs more strong with the availability of the evidences.

SUMMARY

In this chapter, we have discussed about how the problem environment is captured and represented using random variables and how this information is used further to provide reason and inference for an unknown situation.

Uncertainty in the real world is best handled using probability theory. The problem environment can be static or dynamic. In dynamic environment, a set of random variables is used to represent the world at each point. Bayesian network is used to represent the dependencies and infer in the static world. It is assumed that finite previous states influence the present state of the world. With these assumptions, dynamic environment is represented with Markov models.

Once inferences are drawn for unknown situations, one can attach utility value with each state of world to take decisions. Decision-making may be simple with the choice of one action at any stage. Influence diagram is graphical approach to represent it. Things become complex when multiple actions are involved. Complex or sequential decision-making is modelled using Markov decision process, with the assumption of dynamic world. Dempster–Shafer theory helps in capturing the beliefs and is applied in inferring process.

Perception is another intelligent behaviour. It simulates intelligent behaviour in machine. A machine needs to reason and change the actions with respect to the dynamic environment. It percepts the world through sensors, reasons on accumulated knowledge and then takes action. Non-monotonic reasoning too plays a vital role in the uncertain environment with respect to the knowledge base formulation. Figure 8.10 summarises the dependency of reasoning on decision-making and how they are handled in the static and dynamic environment.

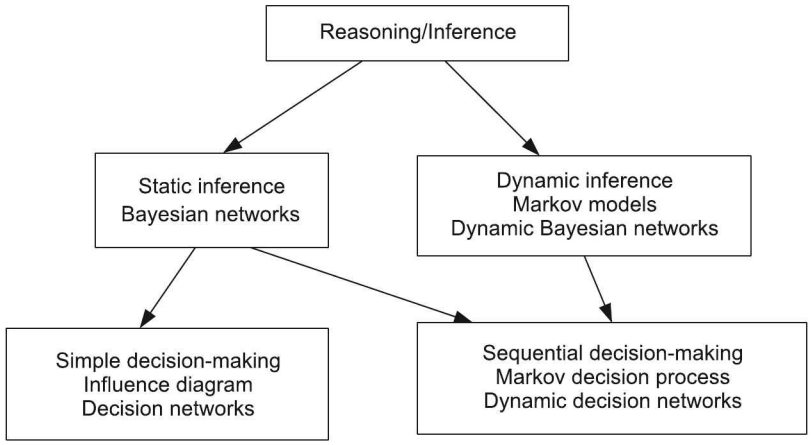


Figure 8.10 Reasoning and decision-making.

KEYWORDS

1. **Bayesian probability:** Bayesian probability provides degree of belief. It allows to reason with hypothesis.
2. **Hypothesis:** It refers to proposed explanation for an observed phenomenon.
3. **Belief network:** Bayesian network is a probabilistic graphical model that represents a set of random variables and their conditional dependencies through a directed acyclic graph (DAG).
4. **Filtering:** It is also referred to as monitoring (in Markov model context). It is the task of computing the posterior probability of the current state or the current belief state.
5. **Prediction:** Inference is drawn here to predict the future (in Markov model context). So, we compute the posterior probability, given all evidence for the future state.
6. **Smoothing:** This involves estimating past from all evidences upto present time.
7. **Influence diagram:** It is a directed graphical representation of the decision-maker's preferences with respect to the real-world uncertainty and sequence of possible decisions.
8. **Principle of maximum expected utility:** This involves selection of actions that help in maximising the expected utility.
9. **Value iteration:** It is an approach in MDP to compute the optimal policy and the value.
10. **Policy iteration:** It is another approach in MDP, where instead of state utilities, the policy is searched at each step.
11. **Dempster–Shafer theory:** This theory, based on evidence, assists in inferring on the basis of belief values and combines different evidences.

MULTIPLE CHOICE QUESTIONS

The questions can have more than one answer:

1. To have rational decision-making in Bayesian networks, we need to add
   (a) Action nodes (b) Decision nodes
   (c) Probability nodes (d) Utility nodes
2. In Markovian environment, the state transition probability depends only on the number of
   (a) Actions (b) Parents (c) Utility nodes (d) None of these
3. _____ describes data mining.
   (a) Grouping of knowledge in efficient way
   (b) Extracting meaningful information by the use of techniques
   (c) Reasoning and inferring for the structured data
   (d) None of the above

4. Which of the following is true for decision networks?
   (a) Compute the decisions based on probabilities.
   (b) Compute utilities which help in decisions
   (c) Compute expected utilities and the decision node value
   (d) All of them
5. In Markov decision process (MDP), which of the following is true?
   (a) An action with the lowest discount factor is selected
   (b) An action with the minimum expected value is selected
   (c) An action with the lowest penalty is selected
   (d) An action with the maximising reward is selected
6. In MDP, the discounting factor plays a role of
   (a) Helping in converging to solution
   (b) Getting sooner rewards
   (c) Getting a lower penalty rate
   (d) All of the above
7. Dempster–Shafer theory is concerned with the evidences to support
   (a) Any single outcome
   (b) Highest reward outcome
   (c) Subsets of outcomes
   (d) None of the above
8. Uncertainty can be regarded as a gap between
   (a) Belief and disbelief
   (b) Belief and plausibility
   (c) Plausibility and belief
   (d) None of the above

CONCEPT REVIEW QUESTIONS

1. What is Markov assumption?
2. Describe the Markov process of *k*th order and write the corresponding transition model.
3. What are the commonalities and differences between static and dynamic Bayesian networks?
4. Referring to the Markov model for inventory management, apply value iteration and policy iteration to find out the optimal policy.

CRITICAL THINKING EXERCISE

1. Consider an example where X person has planned a trek in Himalayas. Suppose you are to assist in the decision-making of whether X should go ahead with the plan or not. Which model would you employ under the uncertainty management?

2. Assume we have some data regarding hotel booking of a hotel. How would you model the uncertainty with regard to no-shows for the hotel if the season is rainy season?

PROJECT WORK

1. Develop a model for college admission process using Markov decision process.