Chapter 3
Heuristic Search Techniques

Dr. Latesh Malik

Contents

- All are varieties of Heuristic Search:
  - Generate and test
  - Hill Climbing
  - Best First Search
  - Problem Reduction
  - Constraint Satisfaction
  - Means-ends analysis

Generate-and-Test

- Algorithm:
  1. Generate a possible solution. For some problems, this means generating a particular point in the problem space. For others it means generating a path from a start state
  2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the chosen path to the set of acceptable goal states.
  3. If a solution has been found, quit, Otherwise return to step 1.

Generate-and-Test

- It is a depth first search procedure since complete solutions must be generated before they can be tested.
- In its most systematic form, it is simply an exhaustive search of the problem space.
- Operate by generating solutions randomly.
- Also called as British Museum algorithm.(find an object)
- Dendral: which infers the structure of organic compounds using NMR spectrogram. It uses plan-generate-test.

Hill Climbing

- Is a variant of generate-and test in which feedback from the test procedure is used to help the generator decide which direction to move in search space.
- The test function is augmented with a heuristic function that provides an estimate of how close a given state is to the goal state.
- Computation of heuristic function can be done with negligible amount of computation.
- Hill climbing is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available.

Simple Hill Climbing

- Algorithm:

1. Evaluate the initial state. If it is also goal state, then return it and quit. Otherwise continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
   a. Select an operator that has not yet been applied to the current state and apply it to produce a new state
   b. Evaluate the new state
      i. If it is the goal state, then return it and quit.
      ii. If it is not a goal state but it is better than the current state, then make it the current state.
      iii. If it is not better than the current state, then continue in the loop.

Simple Hill Climbing

- The key difference between Simple Hill climbing and Generate-and-test is the use of evaluation function as a way to inject task specific knowledge into the control process.
- Is on state better than another ? For this algorithm to work, precise definition of better must be provided.

Steepest-Ascent Hill Climbing

- This is a variation of simple hill climbing which considers all the moves from the current state and selects the best one as the next state.
- Also known as Gradient search

Algorithm: Steepest-Ascent Hill Climbing

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to current state:
   a. Let SUCC be a state such that any possible successor of the current state will be better than SUCC
   b. For each operator that applies to the current state do:
      i. Apply the operator and generate a new state
      ii. Evaluate the new state. If is is a goal state, then return it and quit. If not, compare it to SUCC. If it is better, then set SUCC to this state. If it is not better, leave SUCC alone.
   c. If the SUCC is better than the current state, then set current state to SUCC,

: Hill-climbing

This simple policy has three well-known drawbacks:

1. **Local Maxima**: a local maximum as opposed to global maximum.

2. **Plateaus**: An area of the search space where evaluation function is flat, thus requiring random walk.

3. **Ridge**: Where there are steep slopes and the search direction is not towards the top but towards the side.

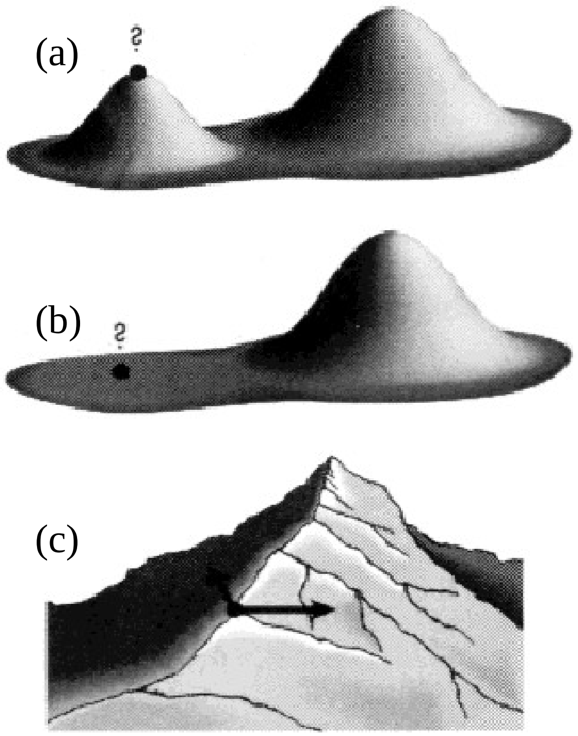
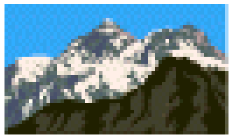


Figure 5.9 Local maxima, Plateaus and ridge situation for Hill Climbing

Hill-climbing



- In each of the previous cases (local maxima, plateaus & ridge), the algorithm reaches a point at which no progress is being made.

- A solution is to do a **random-restart hill-climbing** - where random initial states are generated, running each until it halts or makes no discernible progress. The best result is then chosen.

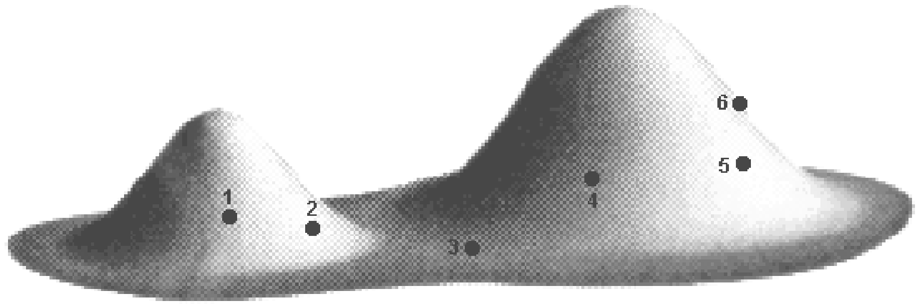
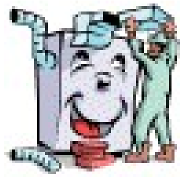


Figure 5.10 Random-restart hill-climbing (6 initial values) for 5.9(a)

- Hill Climbing: Breadth first search
- Steepest ascent hill climbing: Best First Search

Simulated Annealing



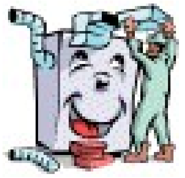
- A alternative to a random-restart hill-climbing when stuck on a local maximum is to do a ‘**reverse walk**’ to **escape the local maximum**.

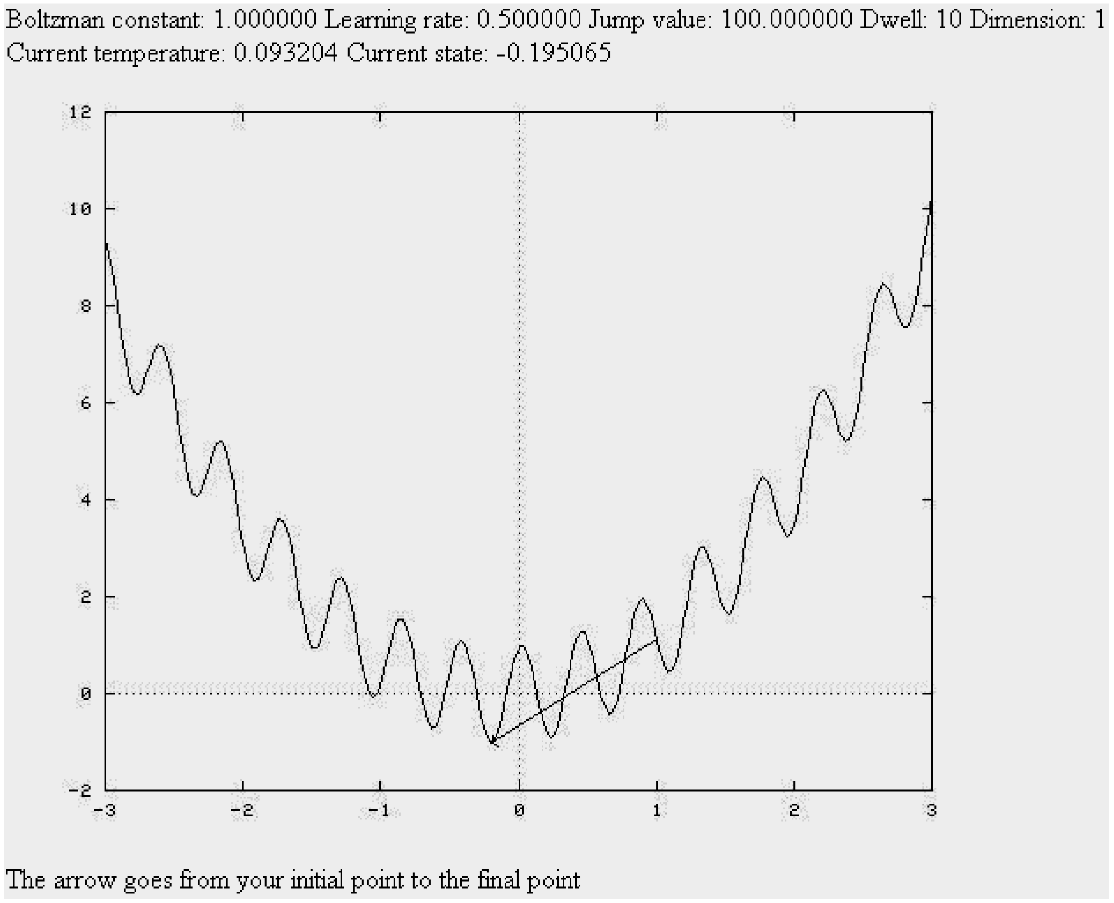
- This is the idea of simulated annealing.

- The term simulated annealing derives from the roughly analogous physical process of **heating** and then **slowly cooling** a substance to obtain a strong crystalline structure.

- The simulated annealing process lowers the temperature by slow stages until the system ``freezes'' and no further changes occur.

Simulated Annealing




Boltzman constant: 1.000000 Learning rate: 0.500000 Jump value: 100.000000 Dwell: 10 Dimension: 1
Current temperature: 0.093204 Current state: -0.195065

The arrow goes from your initial point to the final point

Figure 5.11 Simulated Annealing Demo (http://www.taygeta.com/annealing/demo1.html)

Simulated Annealing

- Probability of transition to higher energy state is given by function:
  - $P = e^{-\Delta E/kt}$

  Where ΔE is the positive change in the energy level

  T is the temperature

  K is Boltzmann constant.

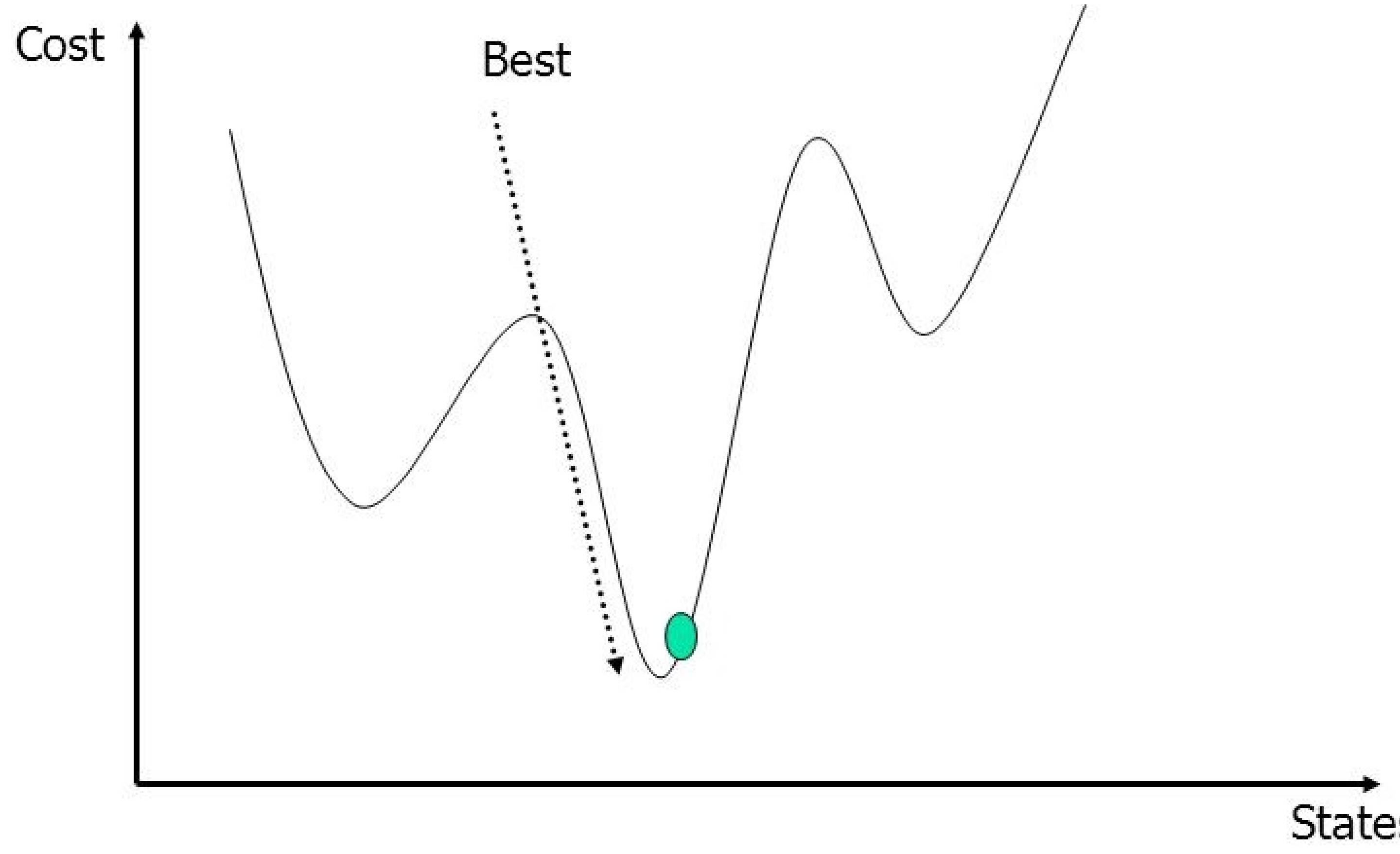
Differences

- The algorithm for simulated annealing is slightly different from the simple-hill climbing procedure. The differences are:
  - The annealing schedule must be maintained
  - Moves to worse states may be accepted

Simulate Annealing: Implementation

- It is necessary to select an annealing schedule which has three components:
  - Initial value to be used for temperature
  - Criteria that will be used to decide when the temperature will be reduced
  - Amount by which the temperature will be reduced.

Simulated Annealing



Best First Search

- Combines the advantages of both DFS and BFS into a single method.
- DFS is good because it allows a solution to be found without all competing branches having to be expanded.
- BFS is good because it does not get branches on dead end paths.
- One way of combining the tow is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does.

BFS

- At each step of the BFS search process, we select the most promising of the nodes we have generated so far.
  - This is done by applying an appropriate heuristic function to each of them.
  - We then expand the chosen node by using the rules to generate its successors

1. Define a list, OPEN, consisting solely of a single node, the start node, *s*.
2. IF the list is empty, return failure.
3. Remove from the list the node *n* with the best score (the node where *f* is the minimum), and move it to a list, CLOSED.
4. Expand node *n*.
5. IF any successor to *n* is the goal node, return success and the solution (by tracing the path from the goal node to *s*).
6. FOR each successor node:a) apply the evaluation function, *f*, to the node.b) IF the node has not been in either list, add it to OPEN.
7. looping structure by sending the algorithm back to the second step.

EXAMPLE on Best-First Search

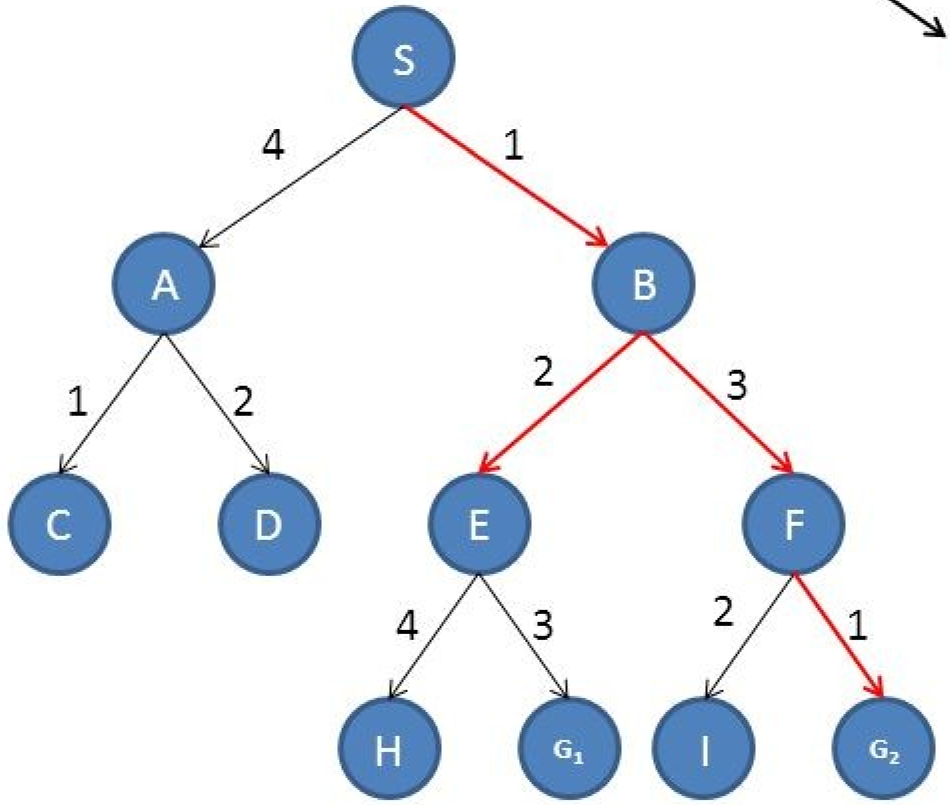
open=[S_0]; closed=[ ]

open=[B_1 , A_4]; closed=[S_0]

open=[E_2 , F_3 , A_4]; closed=[S_0 , B_1]

open=[F_3 , $G1_3$, A_4 , H_4]; closed=[S_0 , B_1 , E_2]

open=[$G2_1$, I_2 , $G1_3$, A_4 , H_4]; closed=[S_0 , B_1 , E_2, F_3]



SEARCH PATH = [S_0, B_1, E_2, F_3, $G2_1$]

Cost = 1+3+1=5

Difference Hill Climbing & BFS

- In hill climbing, one move is selected and all the others are rejected, never to be reconsidered. This produces the straight line behavior that is characteristic of hill climbing.
- In BFS, one move is selected, but the others are kept around so that they can be revisited later if the selected path becomes less promising.
- The best available state is selected in the BFS, even if that state has a value that is lower than the value of the state that was just explored. This contrasts with hill climbing, which will stop if there are no successor states with better values than the current state.

BFS : simple explanation

- It proceeds in steps, expanding one node at each step, until it generates a node that corresponds to a goal state.
- At each step, it picks the most promising unvisited node.

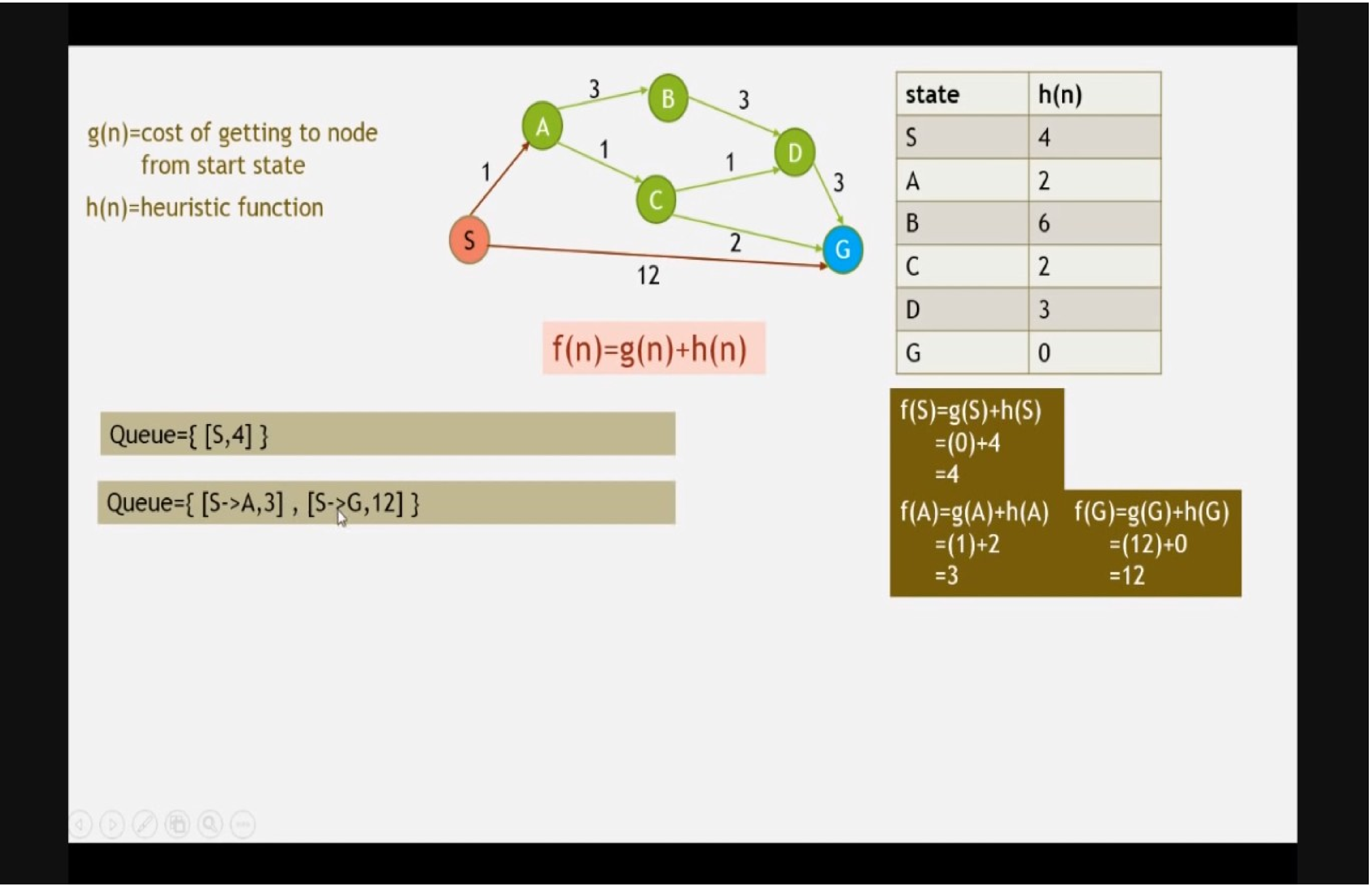
1. Initialize the open list
2. Initialize the closed list put the starting node on the open list (you can leave its **f** at zero)
3. while the open list is not empty

   a) find the node with the least **f** on the open list, call it "q“

   b) pop q off the open list

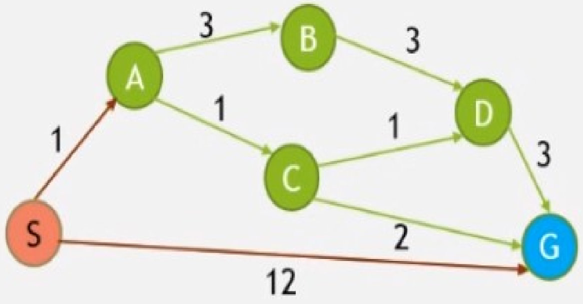
   c) generate q's successors and set their parents to q

```
d) for each successor
i) if successor is the goal, stop search
end (for loop)
e) push q on the closed list end (while loop)
```



g(n)=cost of getting to node from start state

h(n)=heuristic function



f(n)=g(n)+h(n)

state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

Queue={ [S,4] }

Queue={ [S->A,3] , [S->G,12] }

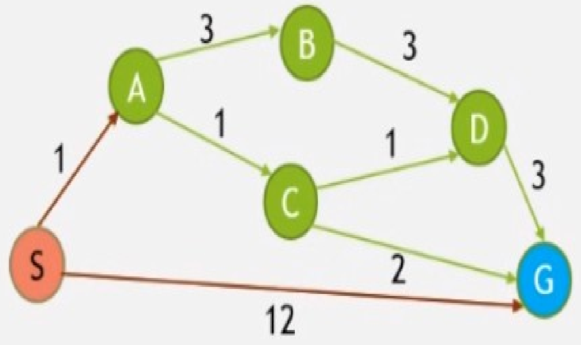
f(S)=g(S)+h(S)
=(0)+4
=4

f(A)=g(A)+h(A)
=(1)+2
=3

f(G)=g(G)+h(G)
=(12)+0
=12

g(n)=cost of getting to node from start state

h(n)=heuristic function



f(n)=g(n)+h(n)

state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

Queue={ [S,4] }

Queue={ [S->A,3] , [S->G,12] }

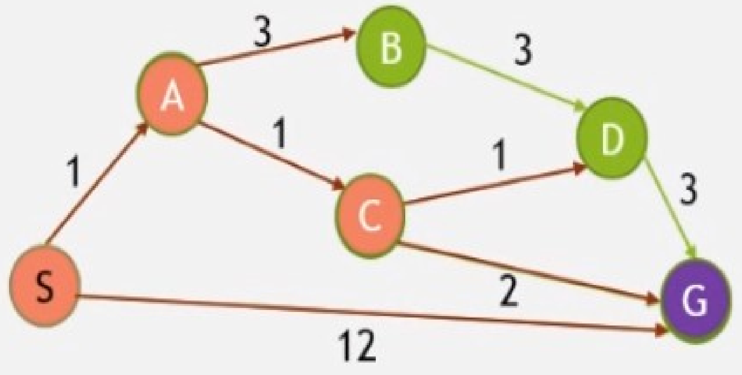
$$f(S)=g(S)+h(S) =(0)+4 =4$$

$$f(A)=g(A)+h(A) =(1)+2 =3$$

$$f(G)=g(G)+h(G) =(12)+0 =12$$

g(n)=cost of getting to node from start state

h(n)=heuristic function



state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

$f(n)=g(n)+h(n)$

Queue={ [S,4] }

Queue={ [S->A,3] , [S->G,12] }

Queue={ [S->A->C,4] , [S->A->B,10] , [S->G,12] }

Queue={ [S->A->C->G,4] , [S->A->C->D,6] , [S->G,12] , [S->A->B,10] , [S->G,12] }

Final path : S->A->C->G with cost = 4

f(S)=g(S)+h(S)
=(0)+4
=4

f(A)=g(A)+h(A)
=(1)+2
=3

f(G

f(C)=g(C)+h(C)
=(1+1)+2
=4

f(B

f(D)=g(D)+h(D)
=(1+1+1)+3
=6

f(G

Problem Reduction

- FUTILITY is chosen to correspond to a threshold such than any solution with a cost above it is too expensive to be practical, even if it could ever be found.

Algorithm : Problem Reduction

1. Initialize the graph to the starting node.
2. Loop until the starting node is labeled SOLVED or until its cost goes above FUTILITY:
    a. Traverse the graph, starting at the initial node and following the current best path, and accumulate the set of nodes that are on that path and have not yet been expanded or labeled as solved.
    b. Pick one of these nodes and expand it. If there are no successors, assign FUTILITY as the value of this node. Otherwise, add its successors to the graph and for each of them compute f'. If f' of any node is 0, mark that node as SOLVED.
    c. Change the f' estimate of the newly expanded node to reflect the new information provided by its successors. Propagate this change backward through the graph. This propagation of revised cost estimates back up the tree was not necessary in the BFS algorithm because only unexpanded nodes were examined. But now expanded nodes must be reexamined so that the best current path can be selected.

Constraint Satisfaction

- Constraint Satisfaction problems in AI have goal of discovering some problem state that satisfies a given set of constraints.
- Design tasks can be viewed as constraint satisfaction problems in which a design must be created within fixed limits on time, cost, and materials.
- Constraint satisfaction is a search procedure that operates in a space of constraint sets. The initial state contains the constraints that are originally given in the problem description. A goal state is any state that has been constrained “enough” where “enough”must be defined for each problem. For example, in cryptarithmetic, enough means that each letter has been assigned a unique numeric value.
- Constraint Satisfaction is a two step process:
  - First constraints are discovered and propagated as far as possible throughout the system.
  - Then if there still not a solution, search begins. A guess about something is made and added as a new constraint.

Algorithm: Constraint Satisfaction

1. Propagate available constraints. To do this first set OPEN to set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until OPEN is empty:
    a. Select an object OB from OPEN. Strengthen as much as possible the set of constraints that apply to OB.
    b. If this set is different from the set that was assigned the last time OB was examined or if this is the first time OB has been examined, then add to OPEN all objects that share any constraints with OB.
    c. Remove OB from OPEN.
2. If the union of the constraints discovered above defines a solution, then quit and report the solution.
3. If the union of the constraints discovered above defines a contradiction, then return the failure.
4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this loop until a solution is found or all possible solutions have been eliminated:
    a. Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.
    b. Recursively invoke constraint satisfaction with the current set of constraints augmented by strengthening constraint just selected.

Means-Ends Analysis(MEA)

- We have presented collection of strategies that can reason either forward or backward, but for a given problem, one direction or the other must be chosen.
- A mixture of the two directions is appropriate. Such a mixed strategy would make it possible to solve the major parts of a problem first and then go back and solve the small problems that arise in “gluing” the big pieces together.
- The technique of Means-Ends Analysis allows us to do that.

Summary

Four steps to design AI Problem solving:

1. Define the problem precisely. Specify the problem space, the operators for moving within the space, and the starting and goal state.
2. Analyze the problem to determine where it falls with respect to seven important issues.
3. Isolate and represent the task knowledge required
4. Choose problem solving technique and apply them to problem.

Summary

- What the states in search spaces represent. Sometimes the states represent complete potential solutions. Sometimes they represent solutions that are partially specified.
- How, at each stage of the search process, a state is selected for expansion.
- How operators to be applied to that node are selected.
- Whether an optimal solution can be guaranteed.
- Whether a given state may end up being considered more than once.
- How many state descriptions must be maintained throughout the search process.
- Under what circumstances should a particular search path be abandoned.

Reference

- Introduction to Artificial Intelligence by Russell Norving