Heuristic Search Techniques

Contents

- A framework for describing search methods is provided and several general purpose search techniques are discussed.
- 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
- If a sufficient number of monkeys were placed in front of a set of typewriters, and left alone long enough, then they would eventually produce all the works of shakespeare.
- Dendral: which infers the struture 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.

: 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.

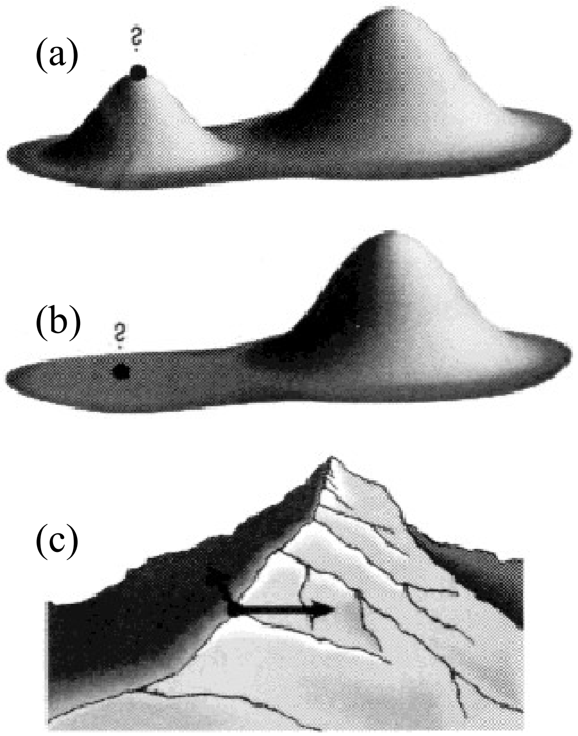


Figure 5.9 Local maxima, Plateaus and ridge situation for Hill Climbing

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 chos

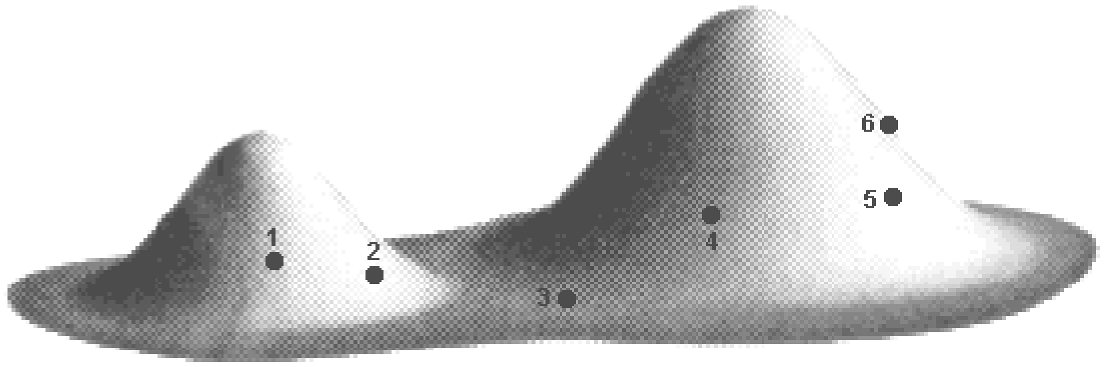
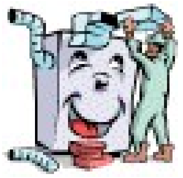


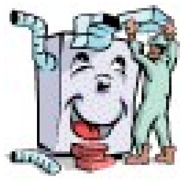
Figure 5.10 Random-restart hill-climbing (6 initial values) for 5.9(a)

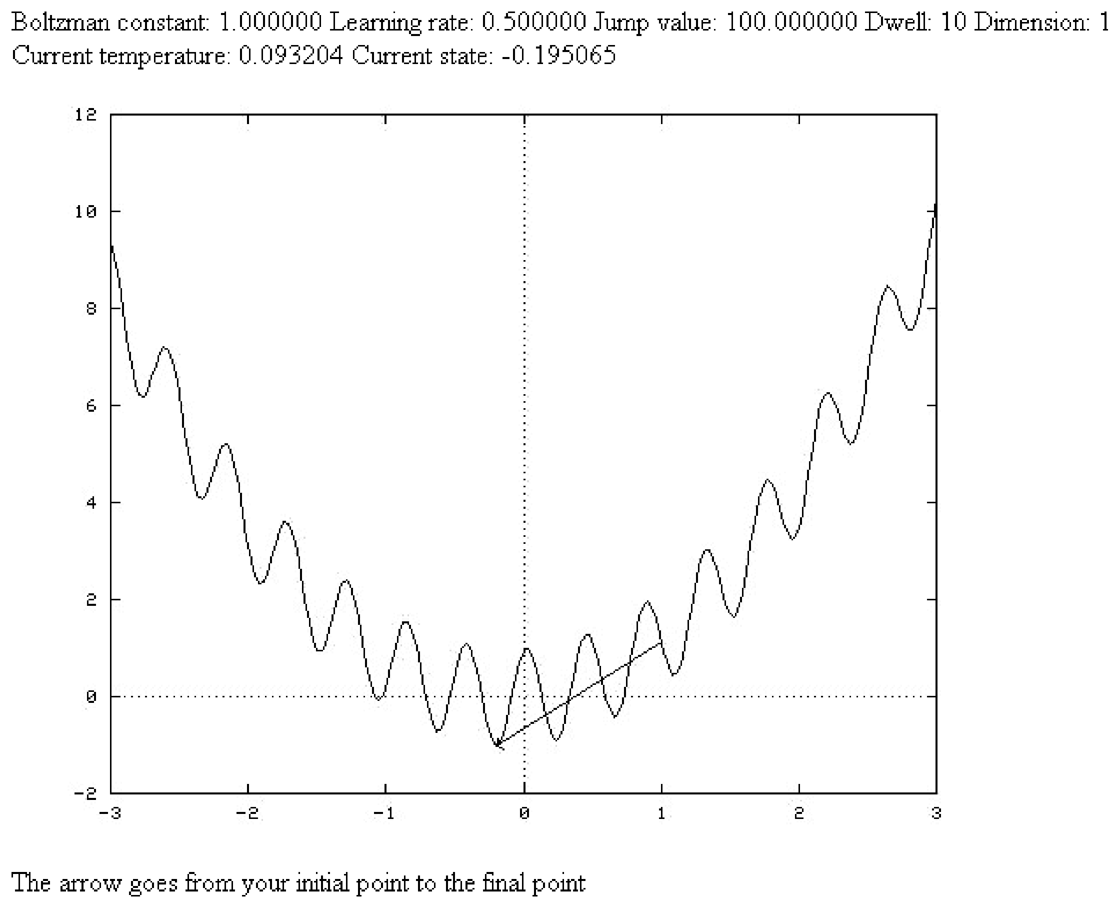
Simulated Annealing



- 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)

Best First Search

- Combines the advantages of bith 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
- Similar to Steepest ascent hill climbing with two exceptions:
  - In hill climbing, one move is selected and all the others are rejected, never to be reconsidered. This produces the straightline behaviour 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. Further, 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.

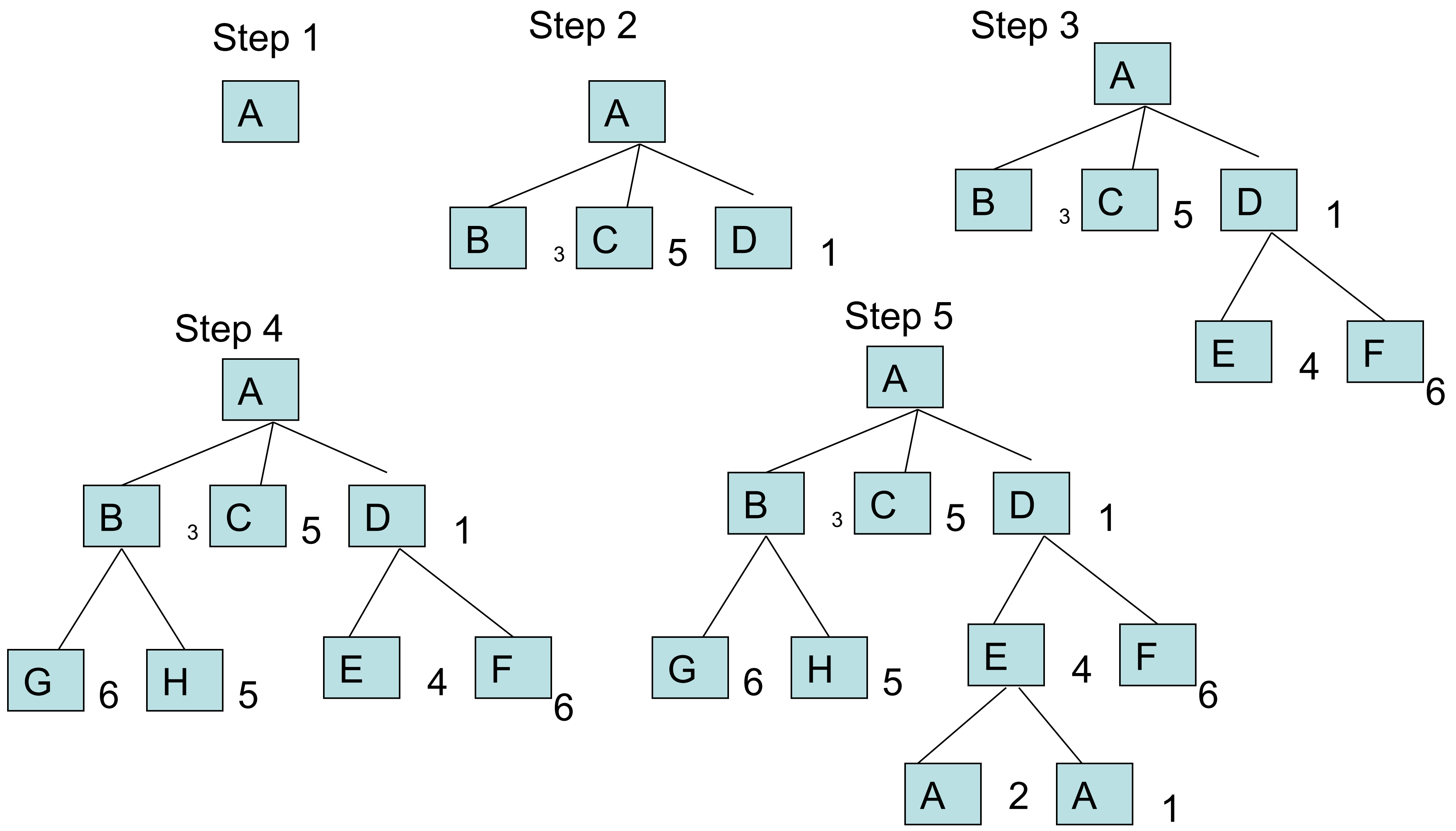
OR-graph

- It is sometimes important to search graphs so that duplicate paths will not be pursued.
- An algorithm to do this will operate by searching a directed graph in which each node represents a point in problem space.
- Each node will contain:
  - Description of problem state it represents
  - Indication of how promising it is
  - Parent link that points back to the best node from which it came
  - List of nodes that were generated from it
- Parent link will make it possible to recover the path to the goal once the goal is found.
- The list of successors will make it possible, if a better path is found to an already existing node, to propagate the improvement down to its successors.
- This is called OR-graph, since each of its branhes represents an alternative problem solving path

Implementation of OR graphs

- We need two lists of nodes:
  - OPEN – nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined. OPEN is actually a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function.
  - CLOSED- nodes that have already been examined. We need to keep these nodes in memory if we want to search a graph rather than a tree, since whenver a new node is generated, we need to check whether it has been generated before.

BFS


Step 1
A
Step 2
A
B 3 C 5 D 1
Step 3
A
B 3 C 5 D 1
E 4 F 6
Step 4
A
B 3 C 5 D 1
G 6 H 5 E 4 F 6
Step 5
A
B 3 C 5 D 1
G 6 H 5 E 4 F 6
A 2 A 1

A* Algorithm

- BFS is a simplification of A* Algorithm
- Presented by Hart et al
- Algorithm uses:
  - f’: Heuristic function that estimates the merits of each node we generate. This is sum of two components, g and h’ and f’ represents an estimate of the cost of getting from the initial state to a goal state along with the path that generated the current node.
  - g : The function g is a measure of the cost of getting from initial state to the current node.
  - h’ : The function h’ is an estimate of the additional cost of getting from the current node to a goal state.
  - OPEN
  - CLOSED

A* Algorithm

1. Start with OPEN containing only initial node. Set that node’s g value to 0, its h’ value to whatever it is, and its f’ value to h’+0 or h’. Set CLOSED to empty list.
2. Until a goal node is found, repeat the following procedure: If there are no nodes on OPEN, report failure. Otherwise picj the node on OPEN with the lowest f’ value. Call it BESTNODE. Remove it from OPEN. Place it in CLOSED. See if the BESTNODE is a goal state. If so exit and report a solution. Otherwise, generate the successors of BESTNODE but do not set the BESTNODE to point to them yet.

A* Algorithm (contd)

- For each of the SUCCESSOR, do the following:

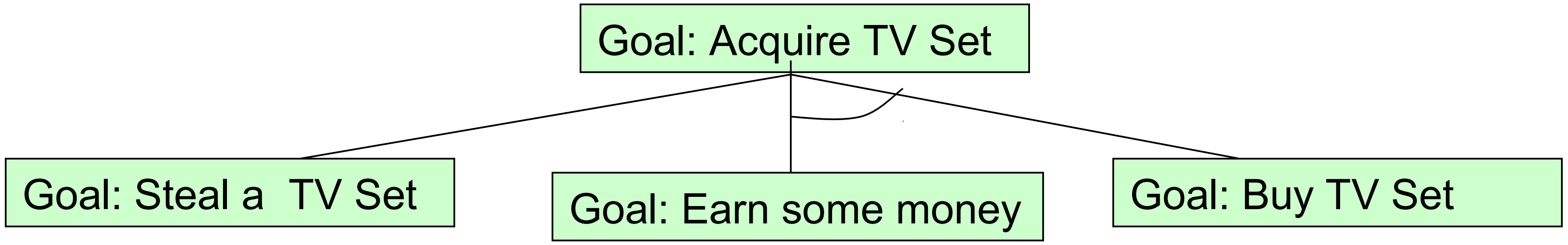
a. Set SUCCESSOR to point back to BESTNODE. These backwards links will make it possible to recover the path once a solution is found.
b. Compute g(SUCCESSOR) = g(BESTNODE) + the cost of getting from BESTNODE to SUCCESSOR
c. See if SUCCESSOR is the same as any node on OPEN. If so call the node OLD.
d. If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so, call the node on CLOSED OLD and add OLD to the list of BESTNODE's successors.
e. If SUCCESSOR was not already on either OPEN or CLOSED, then put it on OPEN and add it to the list of BESTNODE's successors. Compute f'(SUCCESSOR) = g(SUCCESSOR) + h'(SUCCESSOR)

Observations about A*

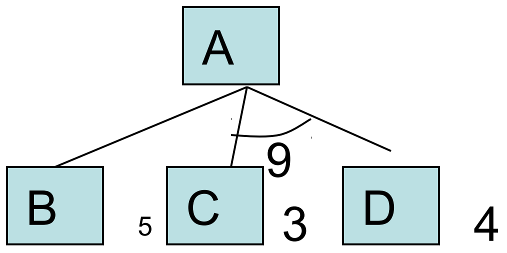
- Role of g function: This lets us choose which node to expand next on the basis of not only of how good the node itself looks, but also on the basis of how good the path to the node was.
- h’, the distance of a node to the goal.If h’ is a perfect estimator of h, then A* will converge immediately to the goal with no search.

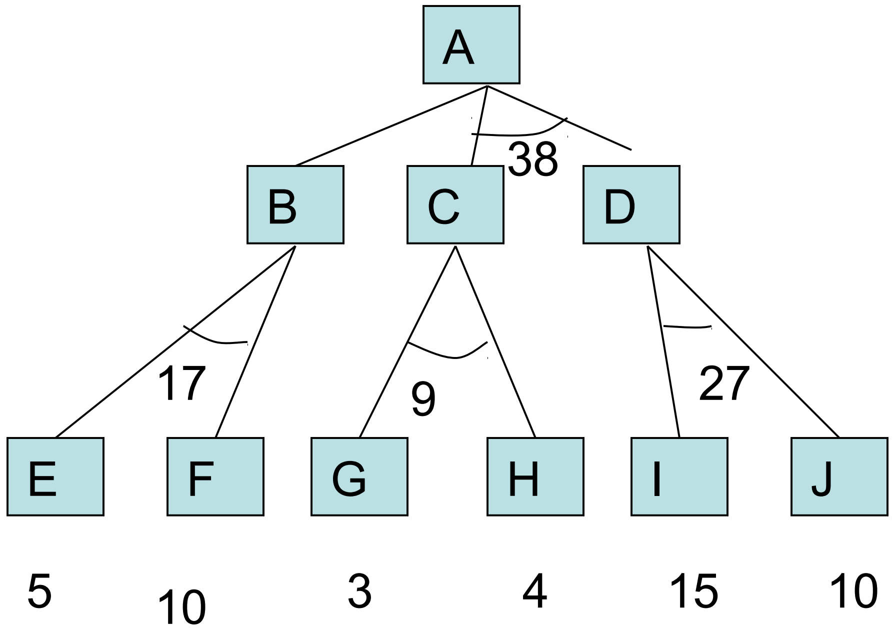
AND-OR graphs

- AND-OR graph (or tree) is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.
- One AND arc may point to any number of successor nodes, all of which must be solved in order for the arc to point to a solution.



AND-OR graph examples




A
B
C
D
38
17
9
27
E
F
G
H
I
J
5
10
3
4
15
10