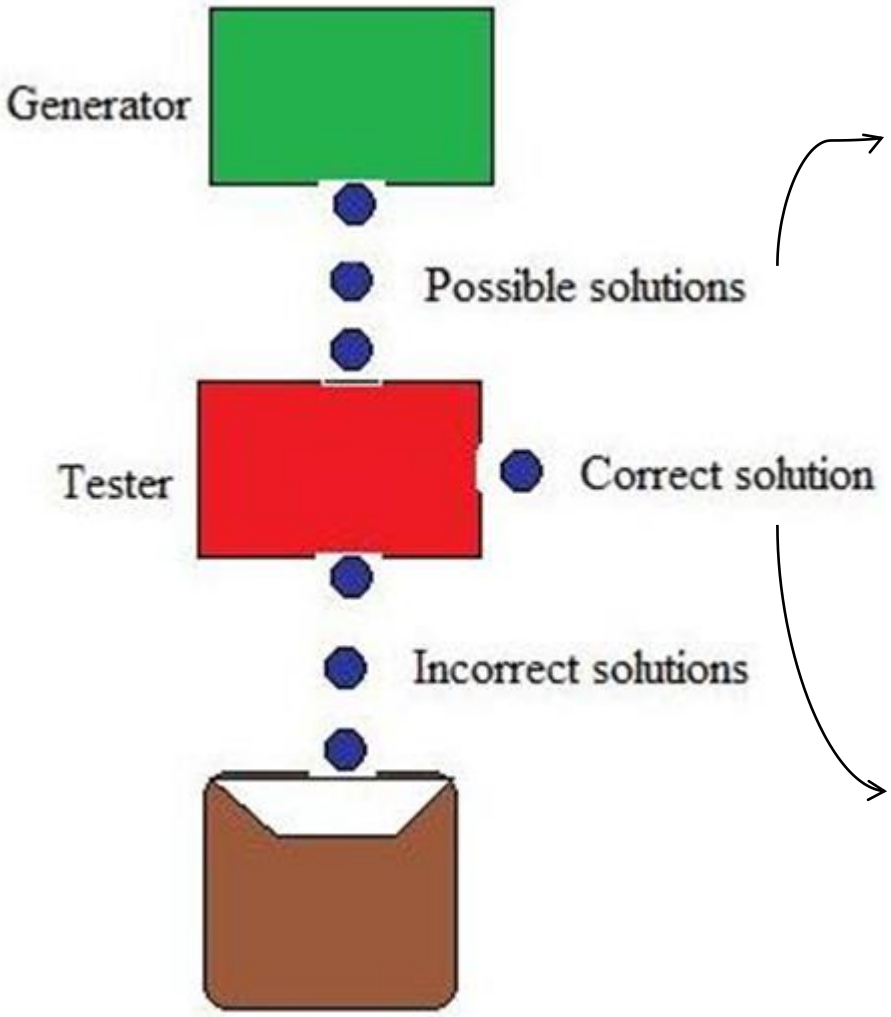
Artificial Intelligence

Open Elective

Class 7: Algorithms

Dr. Santhi Natarajan
Associate Professor
Dept of AI and ML
BMSIT, Bangalore

Generate and Test



- Particular point in problem space
- Path from the start state

- Compare chosen point with acceptable goal state
- Compare end point of chosen path with acceptable goal state
- Quit on solution

Generate and Test

- When done systematically, guaranteed to produce result
- Very exhaustive search, combinatorial problems in simple Generate and Test. Inefficient for problems with large search spaces
- Applied search techniques:
  - DFS – with heuristics
  - British Museum – exhaustive complete search
- Most appropriate technique:
  - DFS with backtracking
- Combination of techniques: DENDRAL
  - Planning: Constraint satisfaction techniques
  - Generate and Test: Planned list used to explore only a fairly limited set of states or paths in graphs
  - Planning often produces inaccurate solutions, as there is no feedback from environment

Hill Climbing

- Variant of Generate and Test
- Feedback from the environment helps the generator to decide which direction to move in the state space diagram
- Heuristics: How close is the current state to the goal state
- Computation of the heuristic cost function done at the same time of performing the test for a solution
- Evaluation function is use as a way to inject task-specific knowledge into the control process
- Is one state better than the other?
  - Higher value of heuristic function
  - Lower value of heuristic function

Hill Climbing

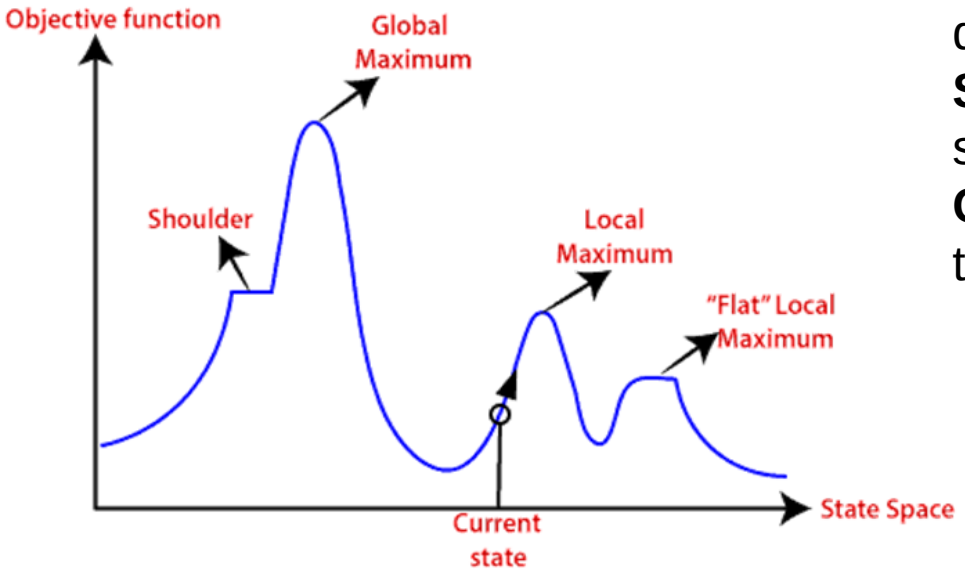
Navigating towards Downtowns

Absolute solution

Travelling salesman

Relative solution

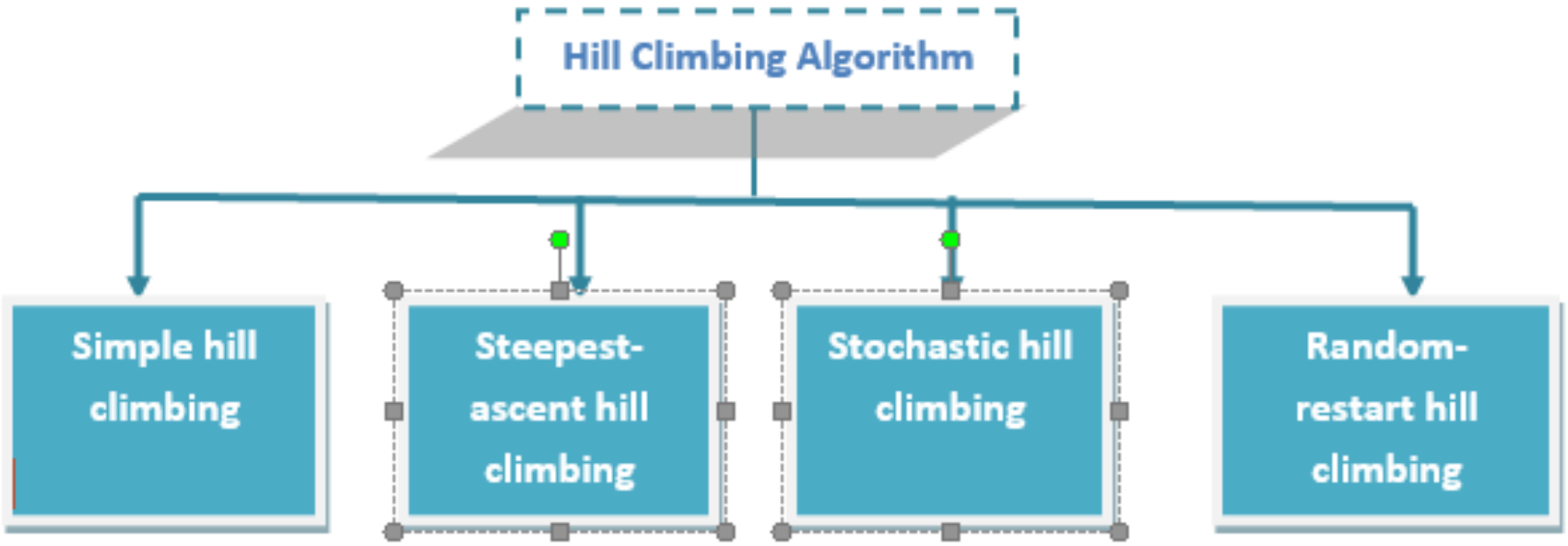
Hill Climbing Landscape



A one-dimensional state-space landscape in which elevation corresponds to the objective function

Global Maximum: It is the highest point on the hill, which is the goal state.
Local Maximum: It is the peak higher than all other peaks but lower than the global maximum.
Flat local maximum: It is the flat area over the hill where it has no uphill or downhill. It is a saturated point of the hill.
Shoulder: It is also a flat area where the summit is possible.
Current state: It is the current position of the person.

Hill Climbing



Hill Climbing

- Evaluate initial state:
  - If initial state == goal state, quit.
  - Else, initial state == current state

- Loop until a solution is found or no new operators are left to be applied in current state
  - Select an operator that has not yet been applied to current state and apply to produce new state:
  - Evaluate new state:
    - ✓ If new state == goal state, quit
    - ✓ If new state better than current state, current state == new state
    - ✓ Else, retain current state and continue in loop

Steepest Ascend Hill Climbing

- Evaluate initial state:
  - If initial state == goal state, quit.
  - Else, current state == , initial state

- Loop until a solution is found or no new operators are left to be applied in current state
  - SUCC: a state such that any possible successor of current state is better than SUCC
  - For each operator applied on current state:
    - ✓ Generate new state
    - ✓ If new state == goal state, return and quit
    - ✓ Else, compare(new state, SUCC).
      - ❖ If new state >= SUCC, SUCC == new state
      - ❖ Else, SUCC == SUCC
  - Evaluate SUCC state:
    - ✓ If SUCC >= current state, current state == SUCC
    - ✓ Else, retain current state and continue in loop

Stochastic and Random Hill Climbing

Stochastic hill climbing

Stochastic hill climbing does not focus on all the nodes. It selects one node at random and decides whether it should be expanded or search for a better one.

Random-restart hill climbing

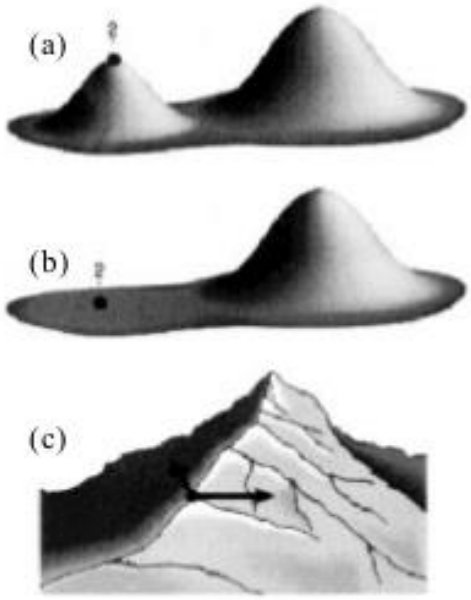
Random-restart algorithm is based on try and try strategy. It iteratively searches the node and selects the best one at each step until the goal is not found. The success depends most commonly on the shape of the hill. If there are few plateaus, local maxima, and ridges, it becomes easy to reach the destination.

Problems in Hill Climbing

: 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.



Local maxima, Plateaus and ridge situation for Hill Climbing

Overcoming Problems in Hill Climbing

Backtrack for local maxima

- Reasonable, if the new node is in another direction that looked as promising or almost as promising as the one that was chosen earlier.

Big jump for plateaus

- Do this to try and get to a new section of the search space. If the rules available describe only single small steps, apply them several times in the same direction.

Combine rules for ridges

- Moving in several directions at once. Apply two or more rules before doing the test.

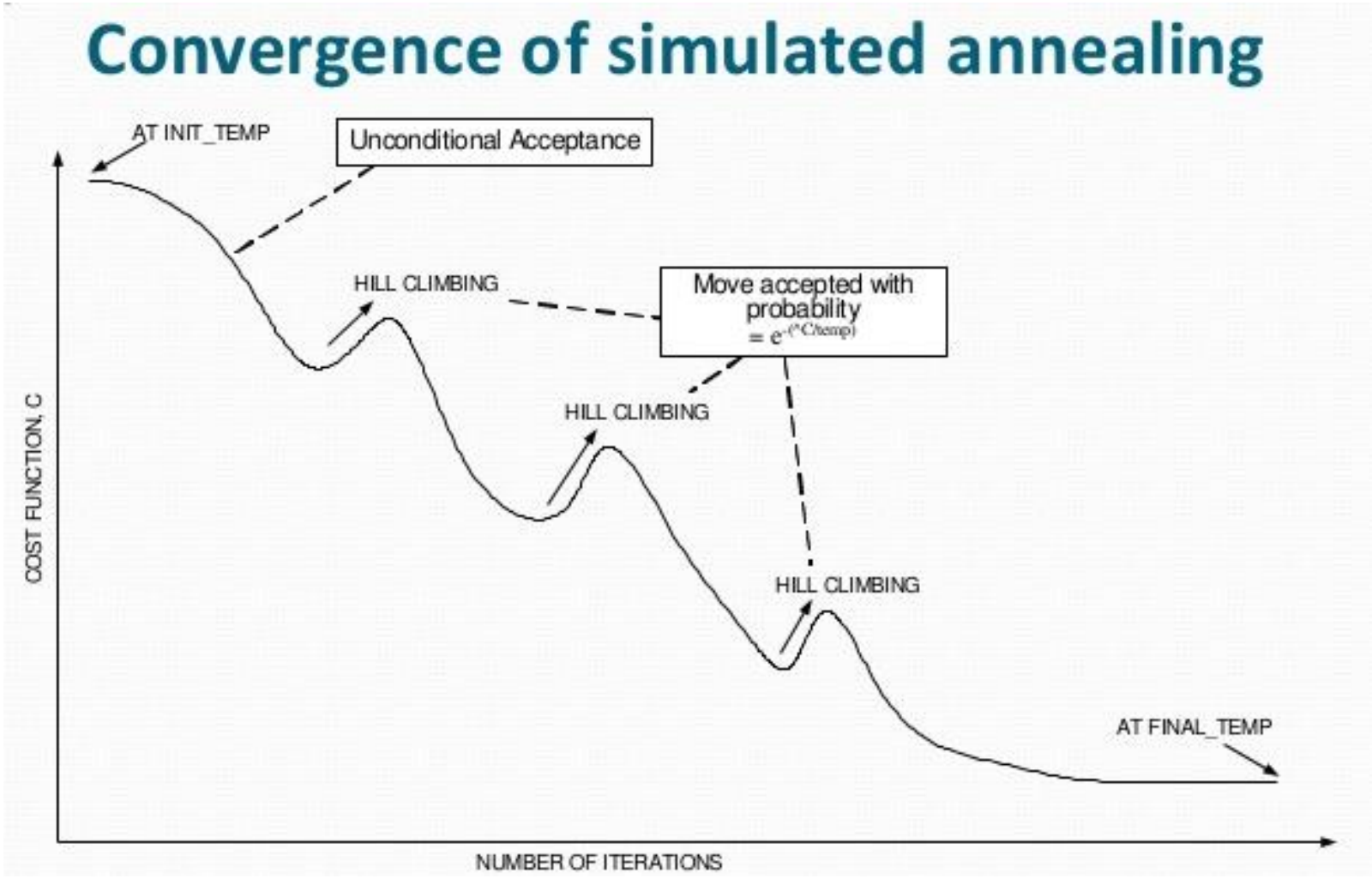
Drawbacks of Hill Climbing

- Unsuited for problems, where value of heuristics function suddenly drops off as we move away from solution, typical cases are those where there is a threshold function
- Local method: the next move is decided by evaluating only the immediate consequence, not by exploring consequences exhaustively.
- Driving downtown:
  - One way street and deadend: knowledge prerequisites
- When we convert local hill climbing into global hill climbing by adding extra knowledge, sometimes computational advantages of local hill climbing is lost.

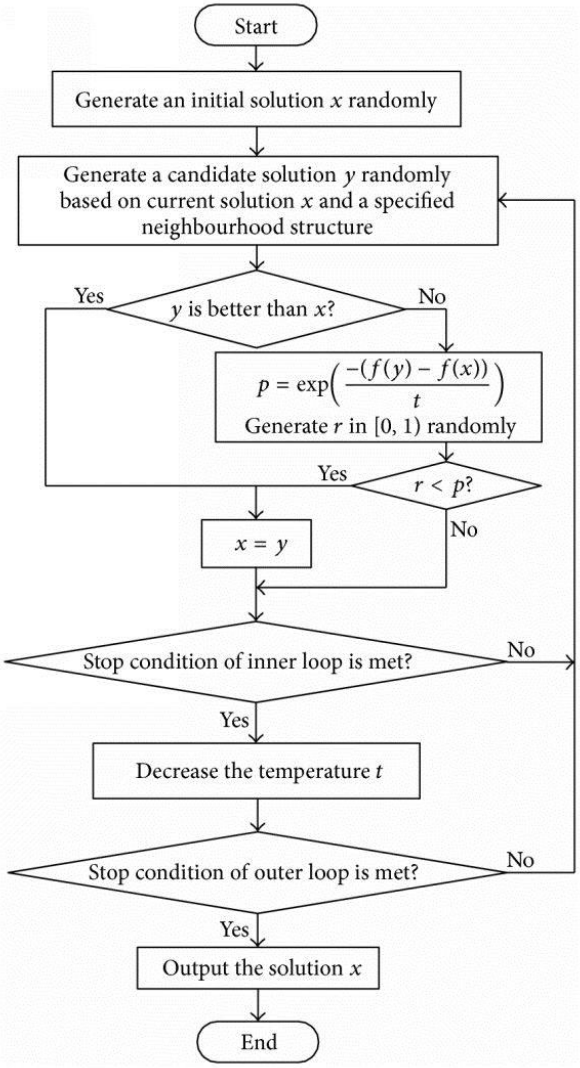
Simulated Annealing

- Process in metallurgy, where metals are slowly cooled to make them reach a state of low energy where they are very strong
- Use objective function and minimize it, instead of maximizing a heuristic function.
- SA is a global optimization technique
- Iterative improvement algorithm
- Perform enough exploration of the whole search space early so that the final solution is relatively insensitive to the starting state
- Lowers the possibilities of getting stuck in local maximas, plateaus and ridges

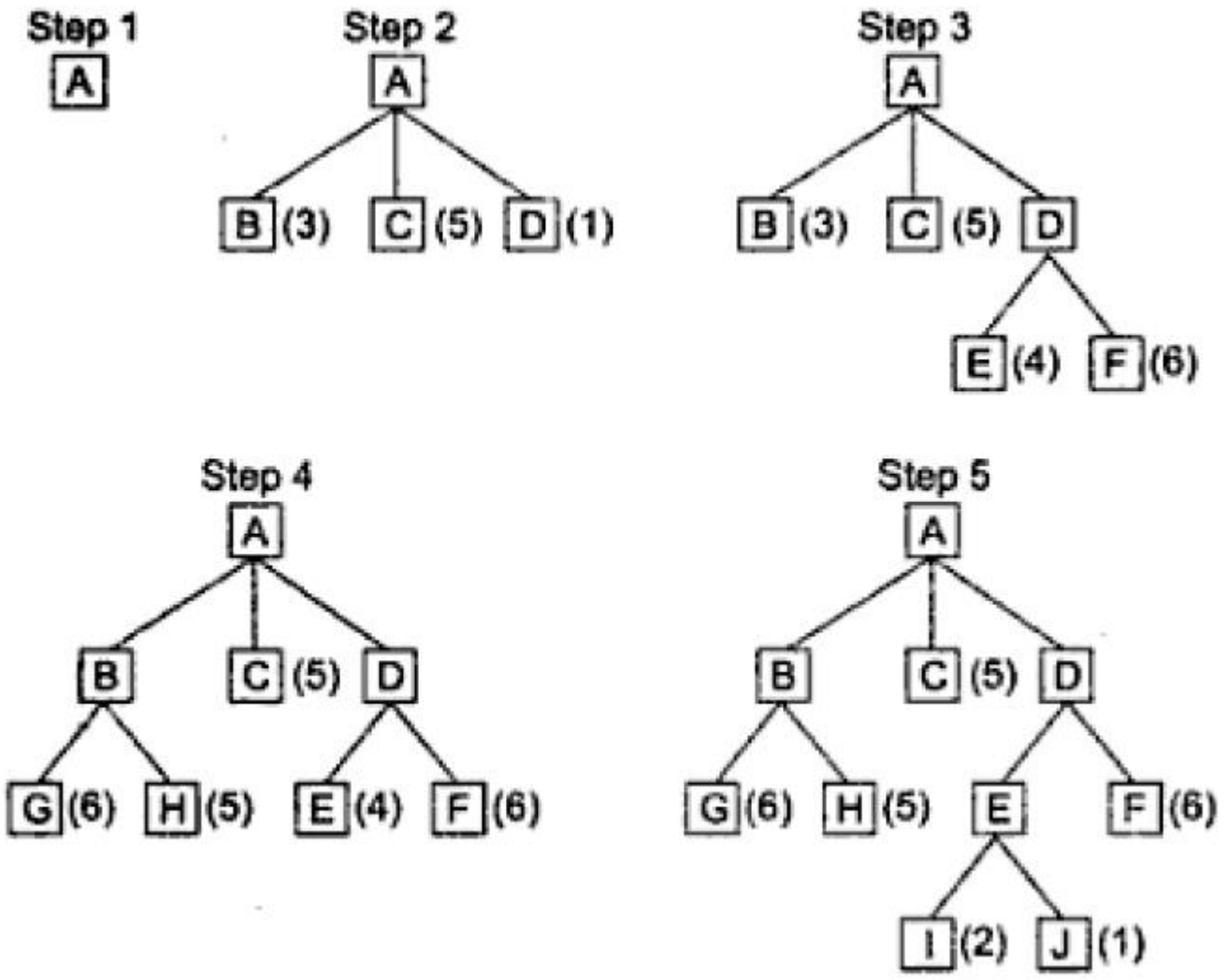
Simulated Annealing



Simulated Annealing



Best First Search – OR Graphs



Best First Search Algorithms

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

- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

OPEN: nodes that have been generated but not yet examined (had their successors generated). This is a priority queue. Elements or nodes with highest priority are those with the most promising values of the heuristic function.

CLOSED: nodes that have already been examined.

Best First Search Algorithms

- Step 1: Define the OPEN list with a single node, the starting node.
- Step 2: Check whether or not OPEN is empty. If it is empty, then the algorithm returns failure and exits.
- Step 3: Remove the node with the best score, *n*, from OPEN and place it in CLOSED.
- Step 4: The fourth step “expands” the node *n*, where expansion is the identification of successor nodes of *n*.
- Step 5: check each of the successor nodes to see whether or not one of them is the goal node. If any successor is the goal node, the algorithm returns success and the solution, which consists of a path traced backwards from the goal to the start node. Otherwise, the algorithm proceeds to the sixth step.
- Sep 6: For every successor node, the algorithm applies the evaluation function, *f*, to it, then checks to see if the node has been in either OPEN or CLOSED. If the node has not been in either, it gets added to OPEN.
- Step 7: Finally, the seventh step establishes a looping structure by sending the algorithm back to the second step. This loop will only be broken if the algorithm returns success in step five or failure in step two.

Best First Search Algorithms

- The algorithm is represented here in pseudo-code:
- 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.

A* Algorithm

The goal node is denoted by node_goal and the source node is denoted by node_start

We maintain two lists: OPEN and CLOSE:

OPEN consists on nodes that have been visited but not expanded (meaning that sucessors have not been explored yet). This is the list of pending tasks.

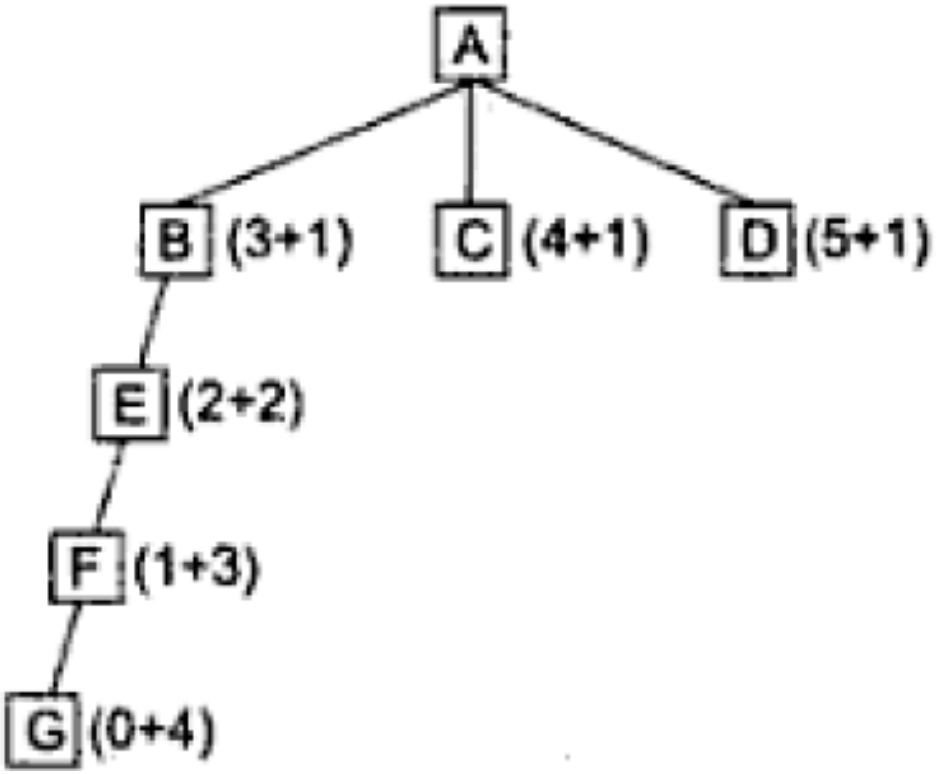
CLOSE consists on nodes that have been visited and expanded (sucessors have been explored already and included in the open list, if this was the case).

A* Algorithm

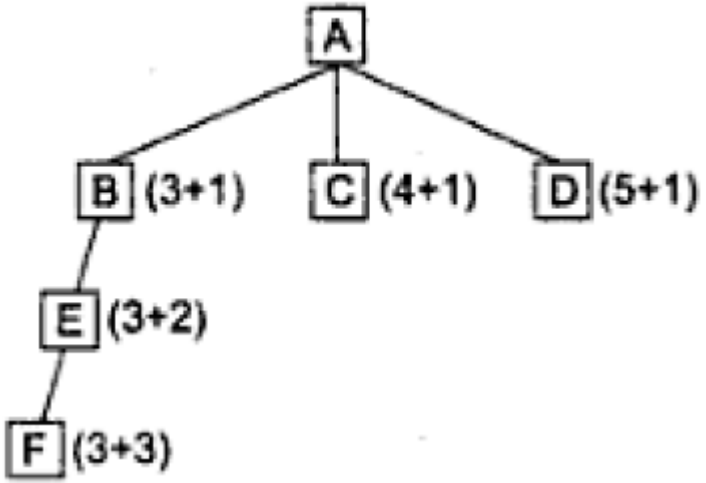
```
Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
while the OPEN list is not empty {
        Take from the open list the node node_current with the lowest
                f(node_current) = g(node_current) + h(node_current)
if node_current is node_goal we have found the solution; break

Generate each state node_successor that come after node_current
for each node_successor of node_current {
        Set successor_current_cost = g(node_current) + w(node_current, node_successor)
        if node_successor is in the OPEN list {
                if g(node_successor) ≤ successor_current_cost continue (to line 20)
        } else if node_successor is in the CLOSED list {
                if g(node_successor) ≤ successor_current_cost continue (to line 20)
                Move node_successor from the CLOSED list to the OPEN list
                } else {
                 Add node_successor to the OPEN list
                Set h(node_successor) to be the heuristic distance to node_goal
                }
                Set g(node_successor) = successor_current_cost
                Set the parent of node_successor to node_current
        }
        Add node_current to the CLOSED list
}
if(node_current != node_goal) exit with error (the OPEN list is empty)
```

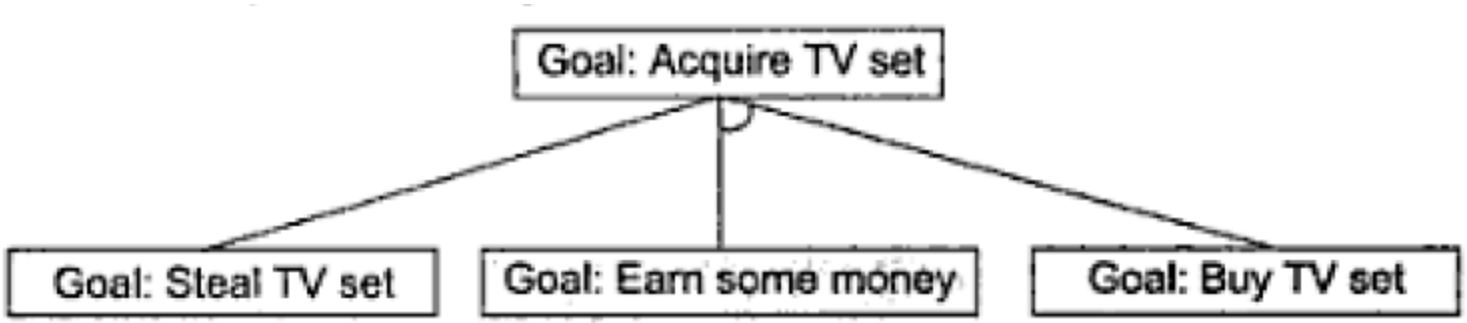
A* Algorithm: Under Estimation



A* Algorithm: Over Estimation

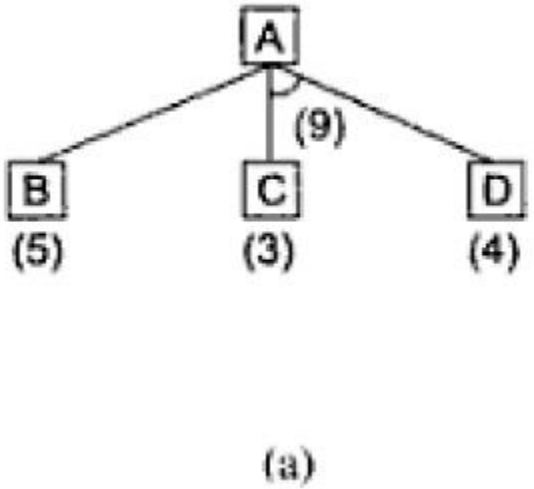


Problem Reduction: AND-OR Tree

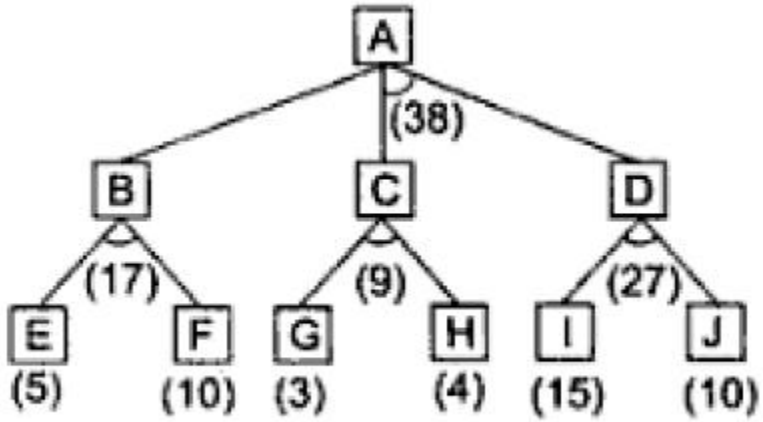


FUTILITY: If estimated cost of a solution > FUTILITY, then abandon the search

Problem Reduction: AND-OR Tree



(a)

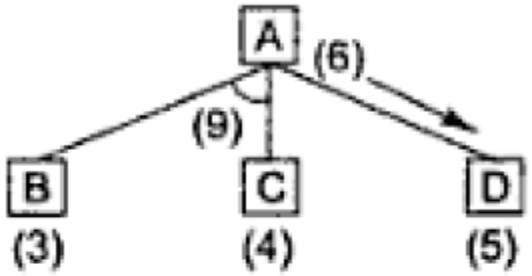


(b)

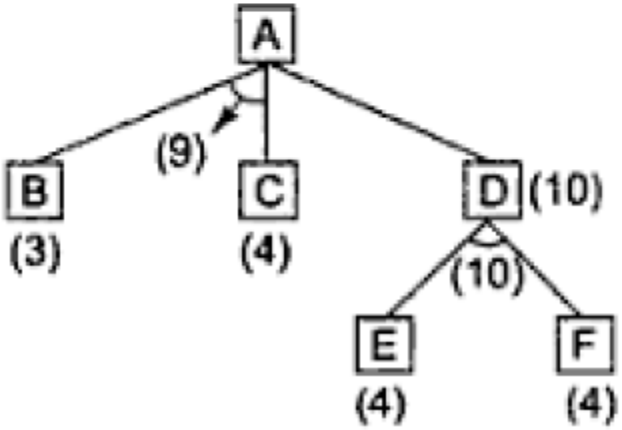
Problem Reduction: AND-OR Tree

Before step 1

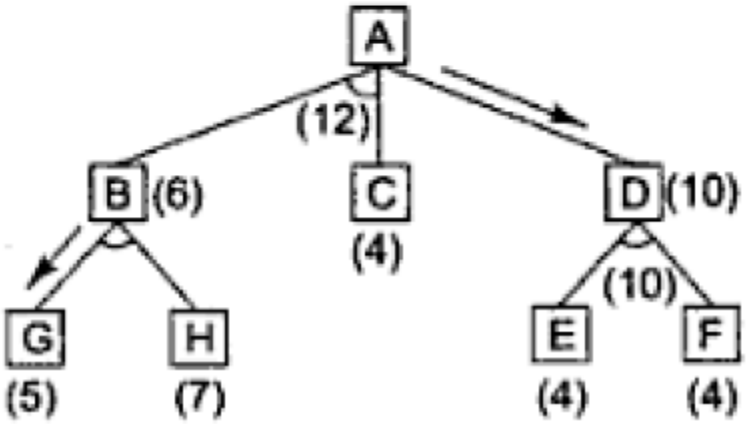
Before step 2



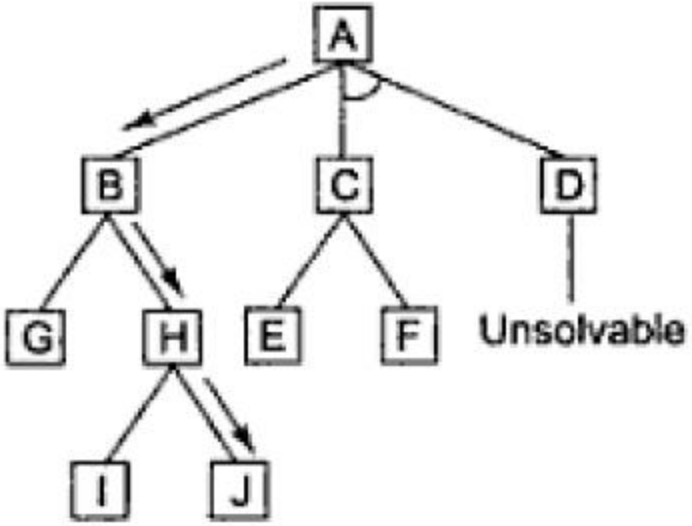
Before step 3

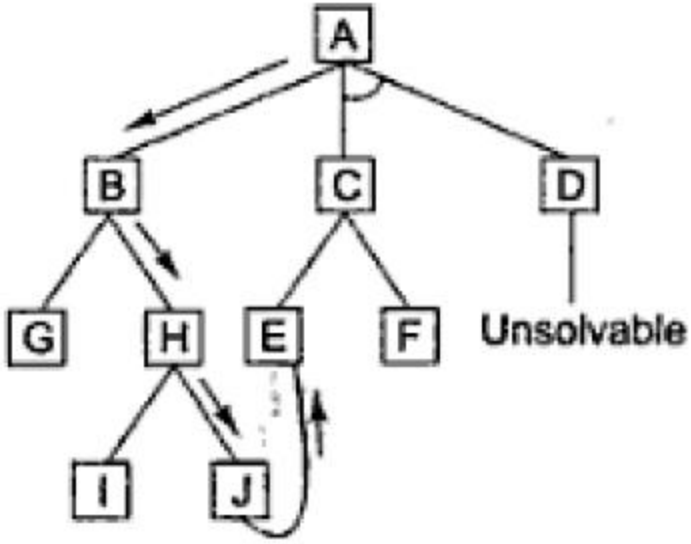


Before step 4

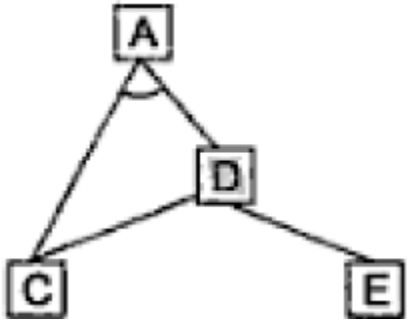


Problem Reduction: Longer path is better





Problem Reduction: Interacting Subgoals



AO* ALGORITHM

The AND/OR Graph Search Problem

- Problem Definition:
  - Given: [G , s , T] where
    - G: implicitly specified AND/OR graph
    - S: start node of the AND/OR graph
    - T: set of terminal nodes
    - H(n): heuristic function estimating the cost of solving the sub- problem at n
  - To find:
    - A minimum cost solution tree

AO* ALGORITHM

Algorithm

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved. If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable, then return failure and exit.

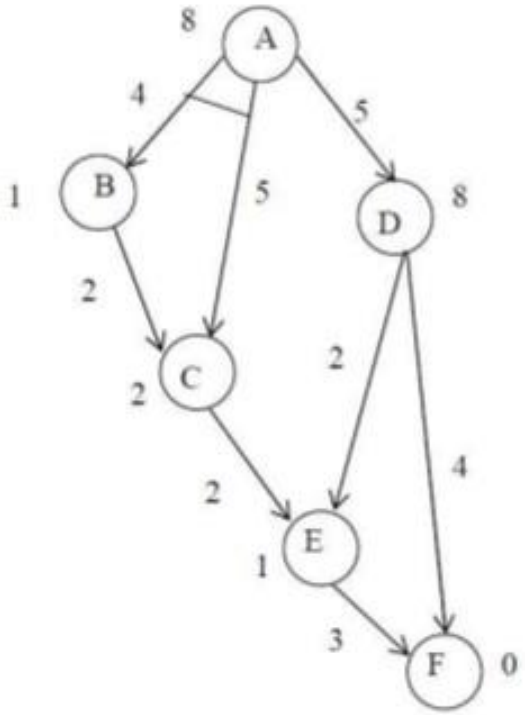
Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

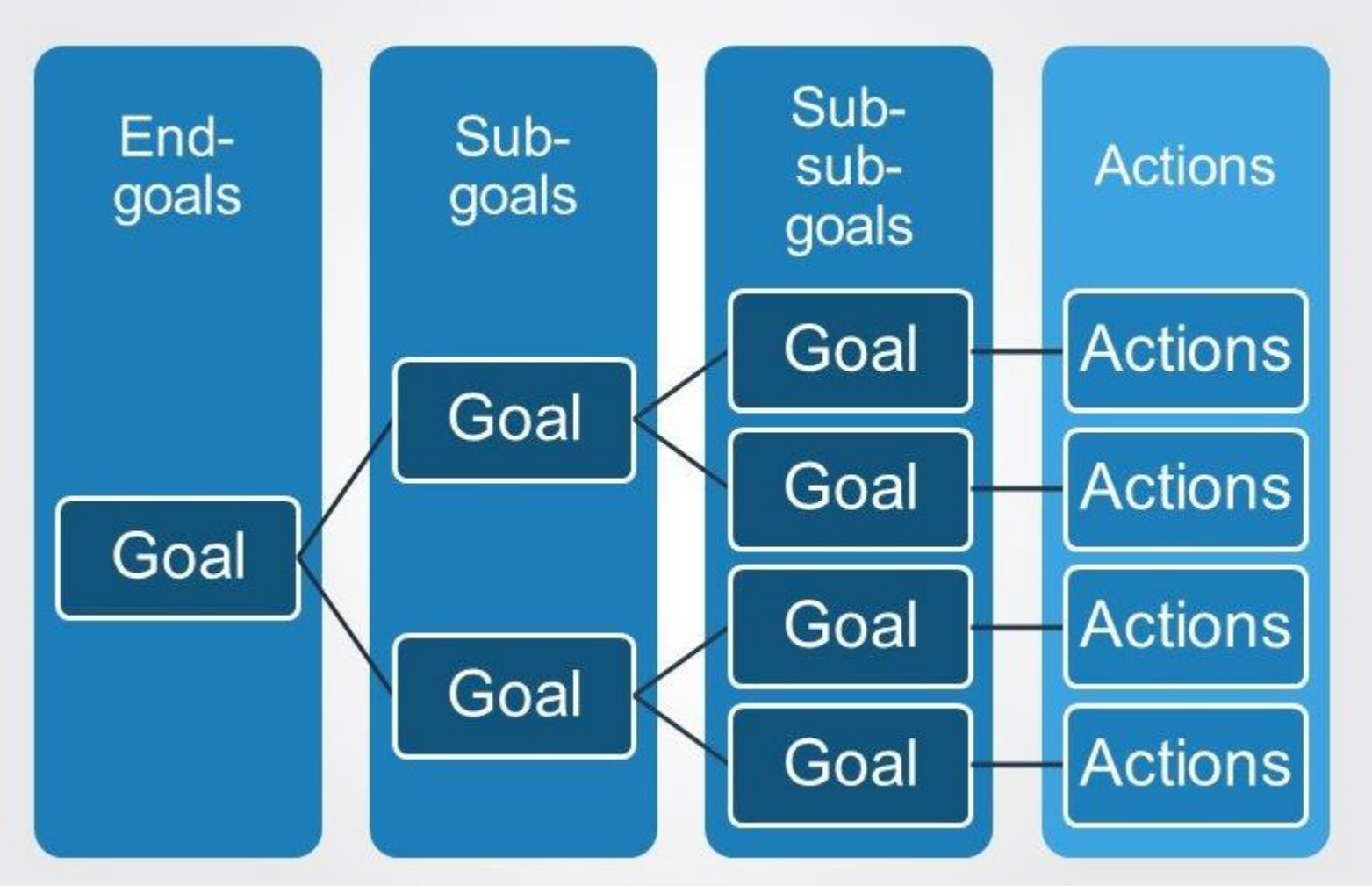
Step 8: Exit. in CLOSE

AO* ALGORITHM

Example



MEANS ENDS ANALYSIS



MEANS ENDS ANALYSIS

Algorithm: Means-Ends Analysis

1. Compare CURRENT to GOAL. If there if no difference between them then return.
2. Otherwise, select the most important difference and reduce it by doing the following until success of failure is signaled:
   a. Select an as yet untried operator O that is applicable to the current difference. If there are no such operators, then signal failure.
   b. Attempt to apply O to CURRENT. Generate descriptions of two states: O-START, a state in which O's preconditions are satisfied and O-RESULT, the state that would result if O were applied in O-START.
   c. If

      (FIRST-PART <- MEA(CURRENT, O-START))

      and

      (LAST-PART <- MEA(O-RESULT, GOAL))

      are successful, then signal success and return the result of concatenating FIRST-PART, O, and LAST-PART.

Agenda Driven Search

Agendas

- An Agenda is a list of tasks a system could perform.
- Associated with each task there are usually two things:
  - A list of reasons why the task is being proposed (justification)
  - Rating representing the overall weight of evidence suggesting that the task would be useful.

Agenda Driven Search

Algorithm: Agenda driven Search

1. Do until a goal state is reached or the agenda is empty:
   a. Choose the most promising task from the agenda.
   b. Execute the task by devoting to it the number of resources determined by its importance. The important resources to consider are time and space. Executing the task will probably generate additional tasks (successor nodes). For each of them do the followings:
      i. See if it is already on the agenda. If so, then see if this same reason for doing it is already on its list of justifications. If so, ignore this current evidence. If this justification was not already present, add it to the list. If the task was not on the agenda, insert it.
      ii. Compute the new task's rating, combining the evidence from all its justifications. Not all justifications need have equal weight. It is often useful to associate with each justification a measure of how strong the reason it is. These measures are then combined at this step to produce an overall rating for the task.

Agenda Driven Search

Chatbot

Person: I don't want to read any more about china. Give me something else.

Computer: OK. What else are you interested in?

Person: How about Italy? I think I'd find Italy interesting.

Computer : What things about Italy are you interested in reading about?

Person: I think I'd like to start with its history.

Computer: why don't you want to read any more about China?

Agenda Driven Search

Example for Agenda: AM

- Mathematics discovery program developed by Lenat (77, 82)
- AM was given small set of starting facts about number theory and a set of operators it could use to develop new ideas.
- These operators included such things as " Find examples of a concept you already know".
- AM's goal was to generate new "interesting" Mathematical concepts.
- It succeeded in discovering such things as prime numbers and Goldbach's conjecture.
- AM used task agenda.

Constraint Satisfaction

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.

Constraint Satisfaction

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.

Constraint Satisfaction

Constraint Satisfaction: Example

- Cryptarithmetic Problem:

```
  SEND
+MORE
----------
MONEY
```

Initial State:

- No two letters have the same value
- The sums of the digits must be as shown in the problem

Goal State:

- All letters have been assigned a digit in such a way that all the initial constraints are satisfied.

Constraint Satisfaction

Cryptasithmetic Problem: Constraint Satisfaction

- The solution process proceeds in cycles. At each cycle, two significant things are done:

1. Constraints are propagated by using rules that correspond to the properties of arithmetic.
2. A value is guessed for some letter whose value is not yet determined.

A few Heuristics can help to select the best guess to try first:

- If there is a letter that has only two possible values and other with six possible values, there is a better chance of guessing right on the first than on the second.
- Another useful Heuristic is that if there is a letter that participates in many constraints then it is a good idea to prefer it to a letter that participates in a few.

Constraint Satisfaction