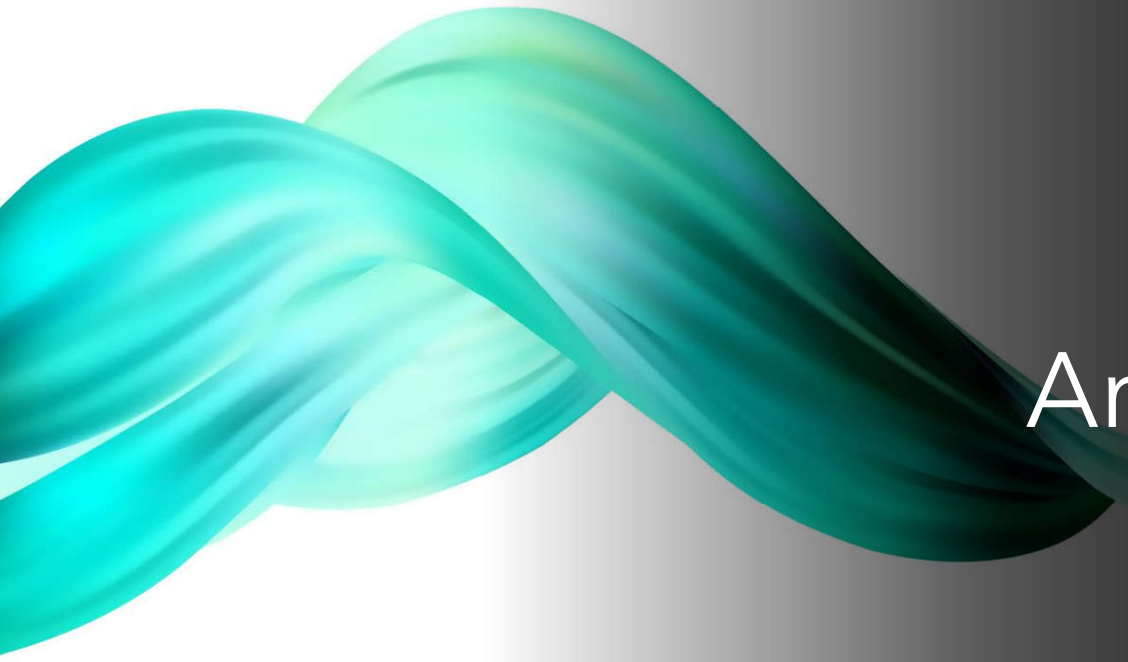


Artificial Intelligence

Dr. Mubashir Ahmad (Ph.D.)

Local search algorithms in AI

- Local search in AI refers to a family of optimization algorithms that are used to find the best possible solution within a given search space. Unlike global search methods that explore the entire solution space, local search algorithms focus on making incremental changes to improve a current solution until they reach a locally optimal or satisfactory solution. This approach is useful in situations where the solution space is vast, making an exhaustive search impractical.

Local search algorithms in AI

- Local search algorithms are a class of techniques used to find optimal solutions in optimization problems. They involve attempts to improve the solution quality by making local changes which may create an improved, but not necessarily globally-optimal result. These algorithms are used for solving complex optimization problems with many variables and constraints.

Working of a Local Search Algorithm

- The basic working principle of a local search algorithm involves the following steps:
- **initialization:** Start with an initial solution, which can be generated randomly or through some heuristic method.
- **Evaluation:** Evaluate the quality of the initial solution using an objective function or a fitness measure. This function quantifies how close the solution is to the desired outcome.
- **Neighbor Generation:** Generate a set of neighboring solutions by making minor changes to the current solution. These changes are typically referred to as "moves."
- **Selection:** Choose one of the neighboring solutions based on a criterion, such as the improvement in the objective function value. This step determines the direction in which the search proceeds.
- **Termination:** Continue the process iteratively, moving to the selected neighboring solution, and repeating steps 2 to 4 until a termination condition is met. This condition could be a maximum number of iterations, reaching a predefined threshold, or finding a satisfactory solution.

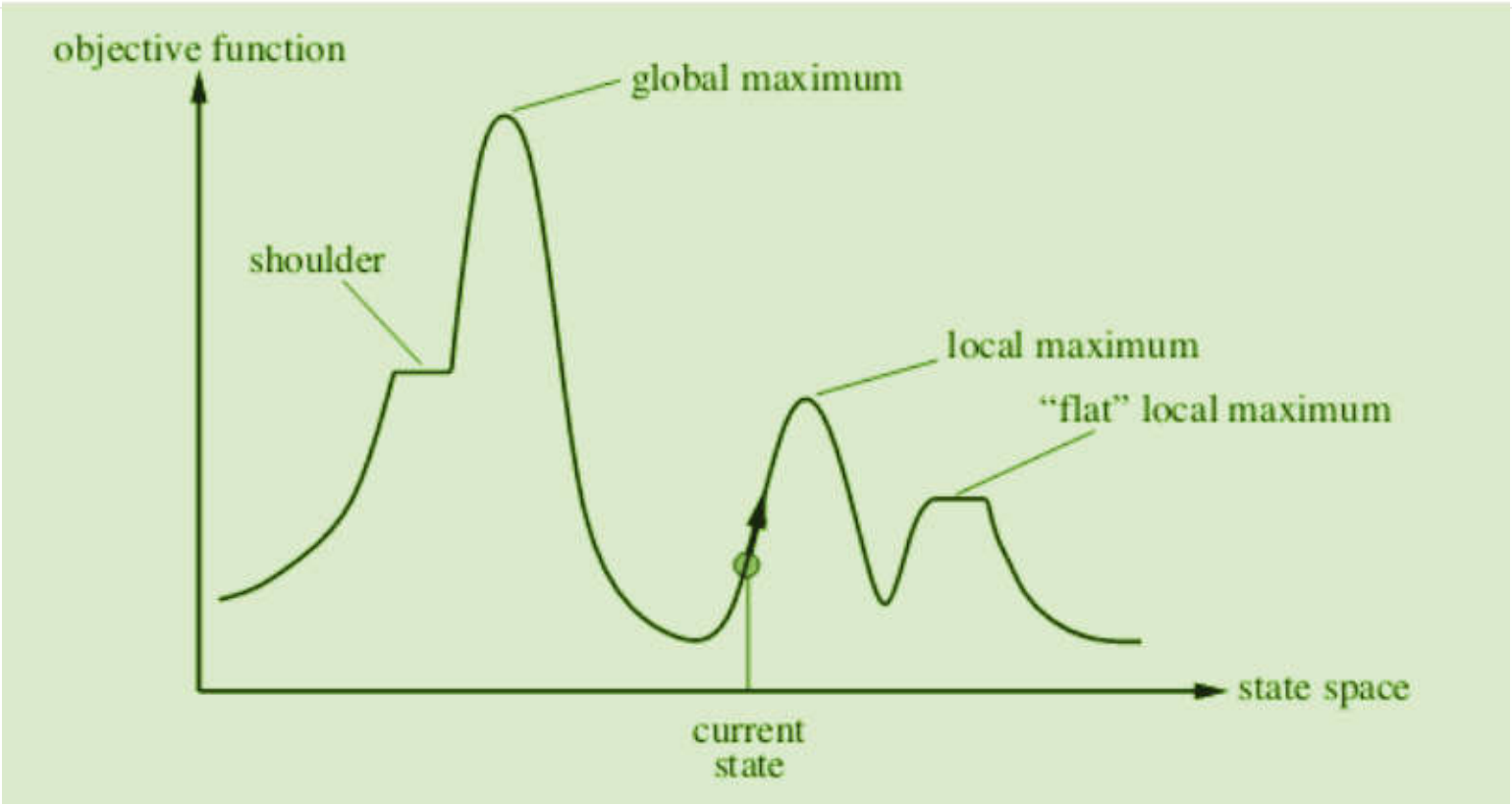
1. Hill Climbing

- In hill climbing the basic idea is to always head towards a state which is better than the current one. (Greedy)
- So, if you are at town A and you can get to town B and town C (and your target is town D) then you should make a move IF town B or C appear nearer to town D than town A does.

1. Hill Climbing

1. Evaluate the initial state. If it is also goal state then return it, otherwise continue with the initial state as the current state.
2. Loop until the solution is found or until there are no new operators 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 new state
   b) Evaluate the new state
      i. If it is a 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 as current state
      iii. If it is not better than the current state, then continue in loop.

Local search algorithms in AI



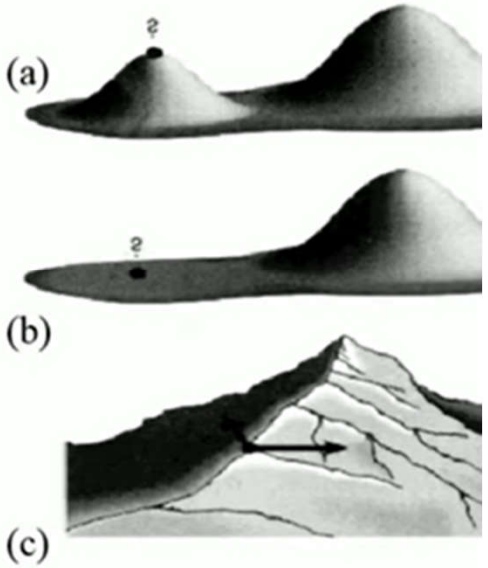
Limitations in Hill Climbing

This simple Hill Climbing policy has three well-known drawbacks:

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

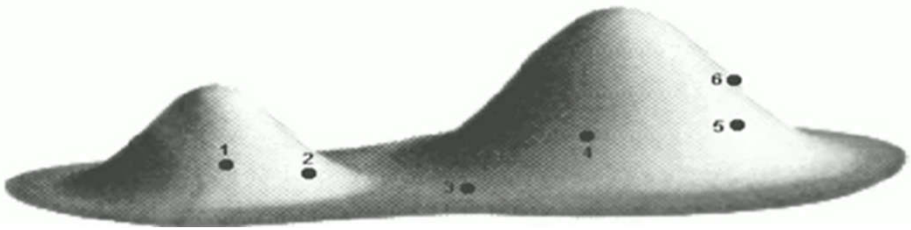
2. **Plateaus**: An area of the search space where evaluation function is flat, thus requiring random walk.

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



Limitations in 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.



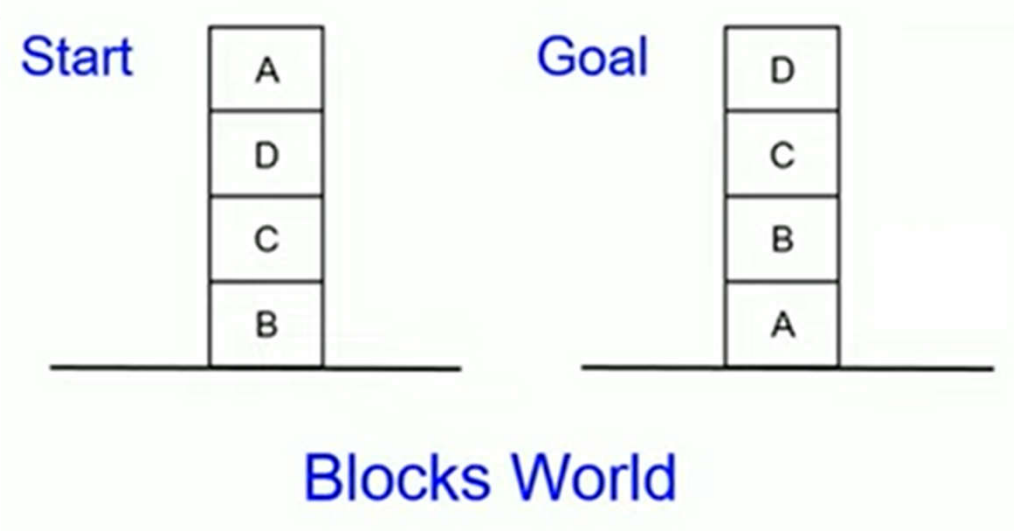
Limitations in Hill Climbing

- Hill climbing using local information :
  Decides what to do next by looking only at the "immediate" consequences of its choices.
- Will terminate when at local optimum.
- The order of application of operators can make a big difference.
- Global information might be encoded in heuristic functions.

Hill climbing with local and global heuristics

- In local heuristics we only consider the immediate consequences. We apply different operators to solve the problem.
- Global heuristics where we will consider the global information where we can say what will happen in the future.

Hill Climbing Local Heuristic Function

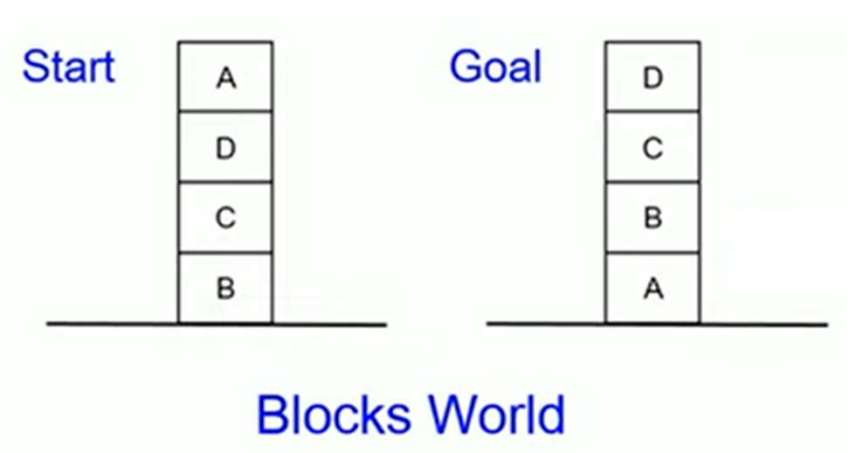


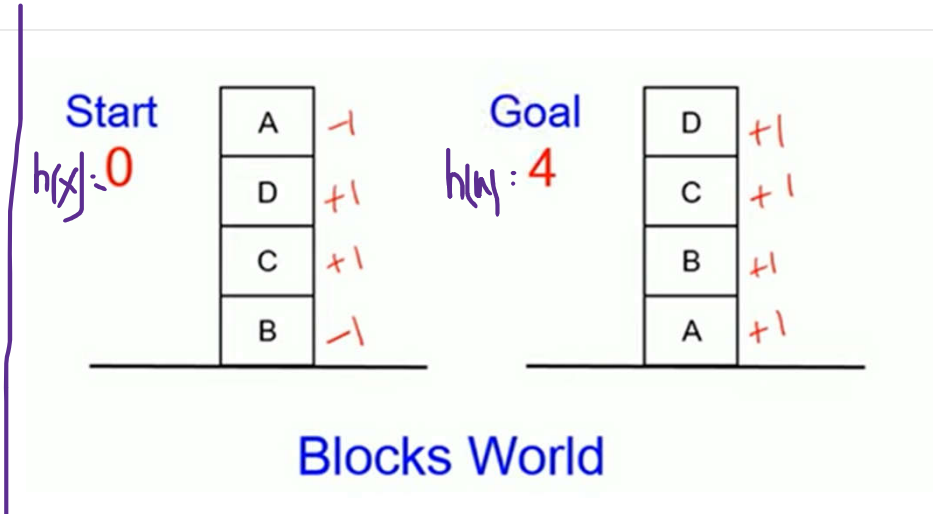
Local heuristic:

+1 for each block that is resting on the thing it is supposed to be resting on.

–1 for each block that is resting on a wrong thing.

Hill Climbing: Local Heuristic Function



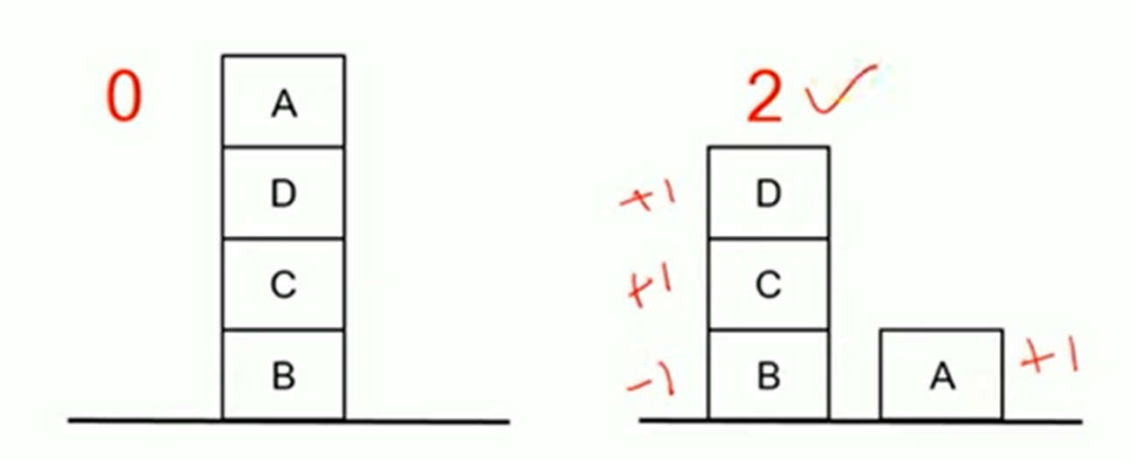


Local heuristic:

+1 for each block that is resting on the thing it is supposed to be resting on.

–1 for each block that is resting on a wrong thing.

Hill Climbing: Local Heuristic Function

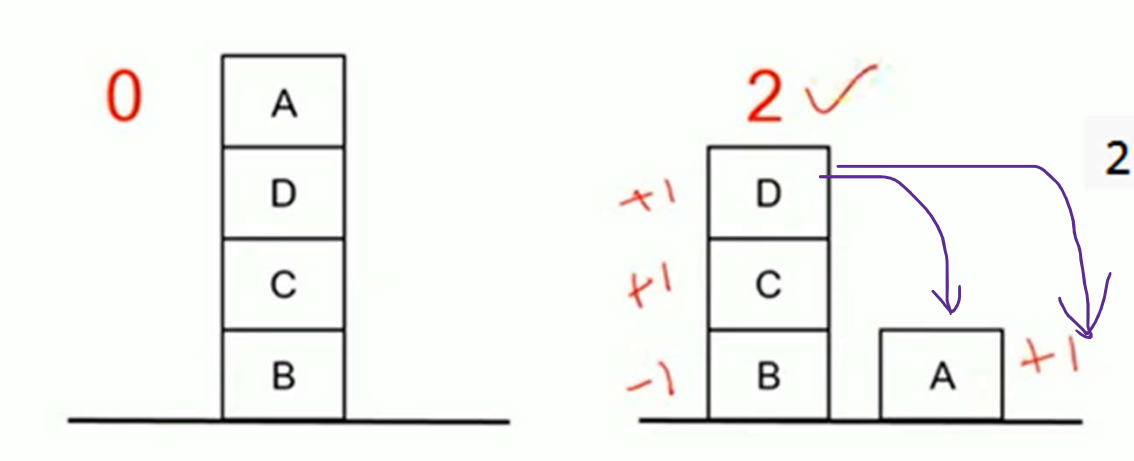


Local heuristic:

+1 for each block that is resting on the thing it is supposed to be resting on.

–1 for each block that is resting on a wrong thing.

Hill Climbing: Local Heuristic Function

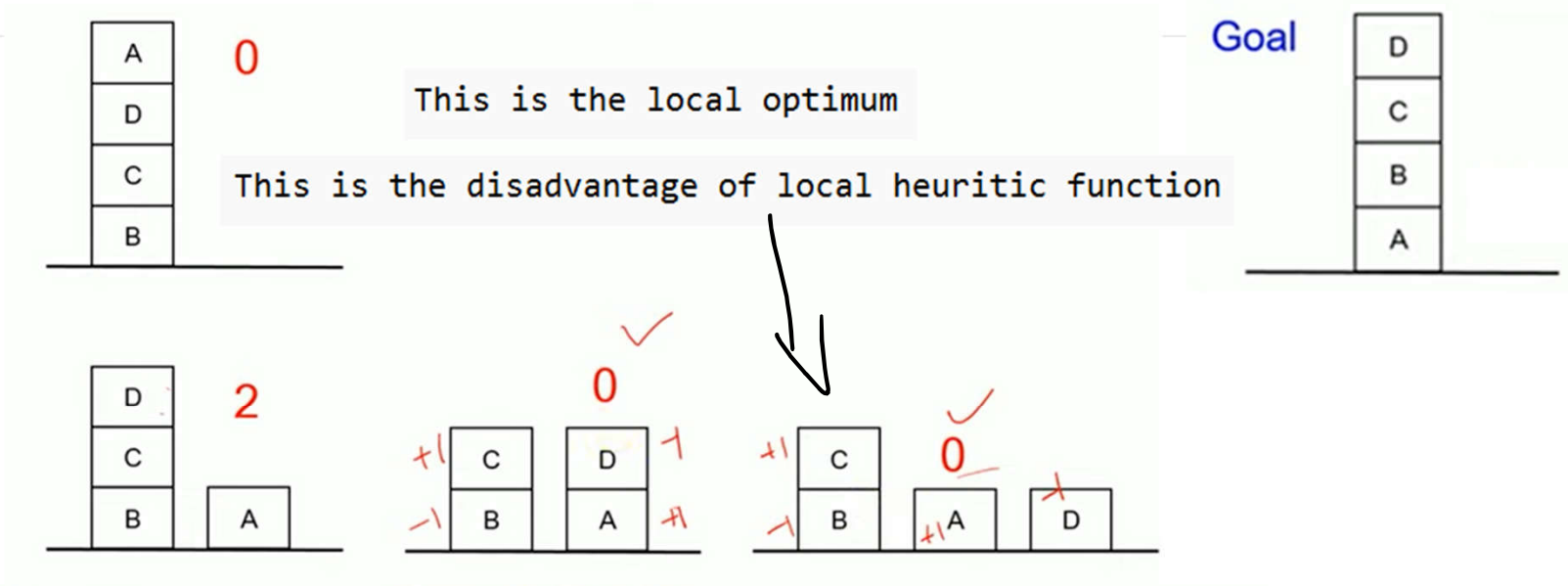


Local heuristic:

+1 for each block that is resting on the thing it is supposed to be resting on.

–1 for each block that is resting on a wrong thing.

Hill Climbing: Local Heuristic Function

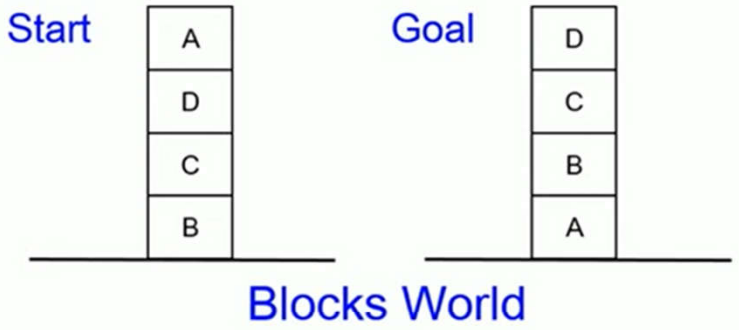


Local heuristic:

+1 for each block that is resting on the thing it is supposed to be resting on.

–1 for each block that is resting on a wrong thing.

Hill Climbing: Global Heuristic Function



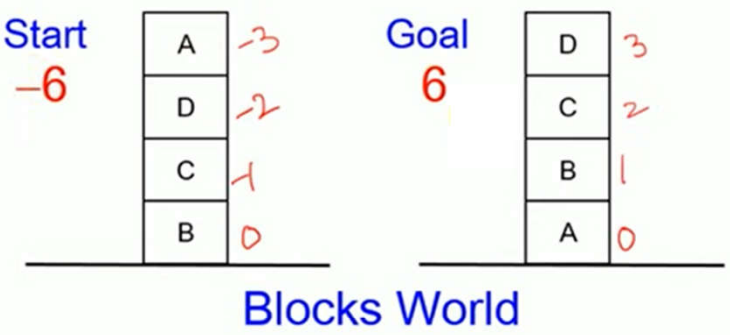
Global heuristic:

For each block that has the correct support structure: +1 to every block in the support structure.

For each block that has a wrong support structure: –1 to every block in the support structure.

Hill Climbing: Global Heuristic Function

```
Below B nothing so we will give 0 reward
Below C, B is present but its not correctly placed, reward -1
and so on
```

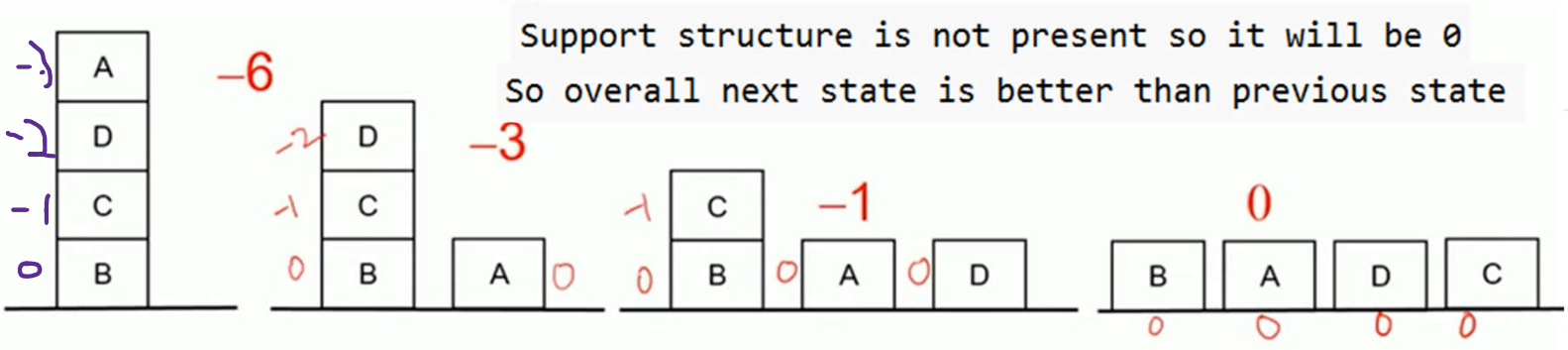


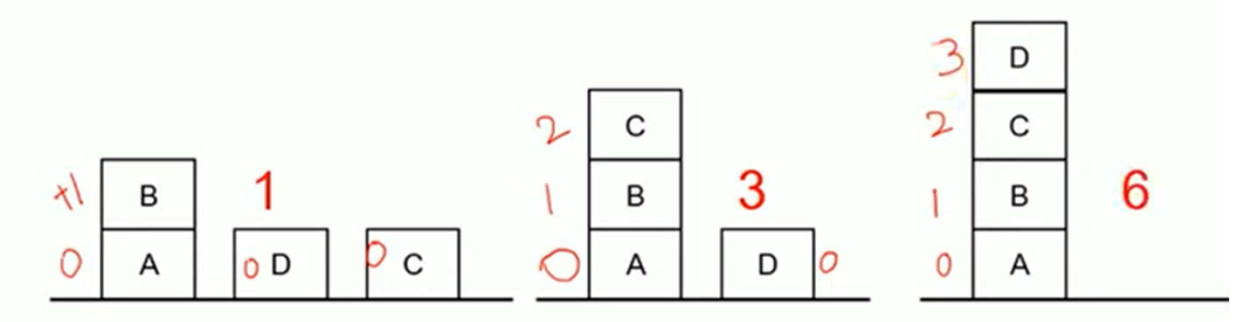
Global heuristic:

For each block that has the correct support structure: +1 to every block in the support structure.

For each block that has a wrong support structure: –1 to every block in the support structure.

Hill Climbing: Global Heuristic Function

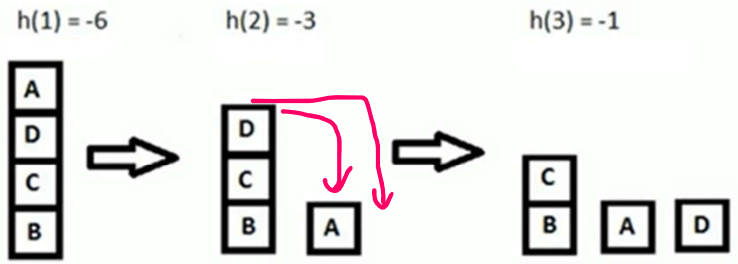




Achieved goal

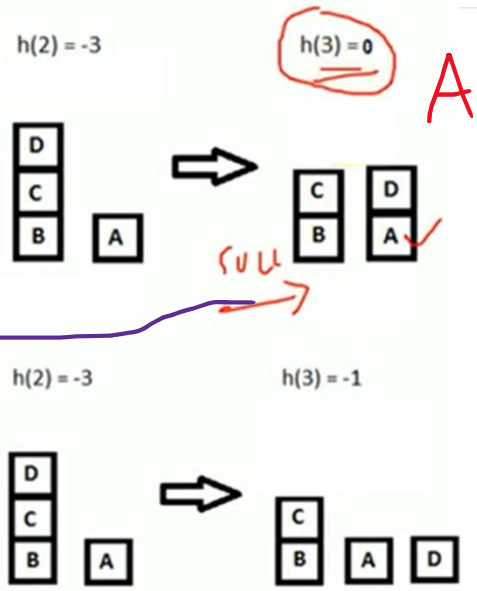
Difference between **Steepest-Ascent** HC and HC Algorithm

- Basic hill climbing first applies one operator and gets new state.
- If it is better that becomes current state
- Whereas steepest hill climbing
  - Considers all the moves from the current state.
  - Selects the best one as the next state.



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
   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 it 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 current state, then set current state to SUCC.



Stochastic hill climbing:

- It does not examine all the neighboring nodes before deciding which node to select. It just selects a neighboring node at random and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

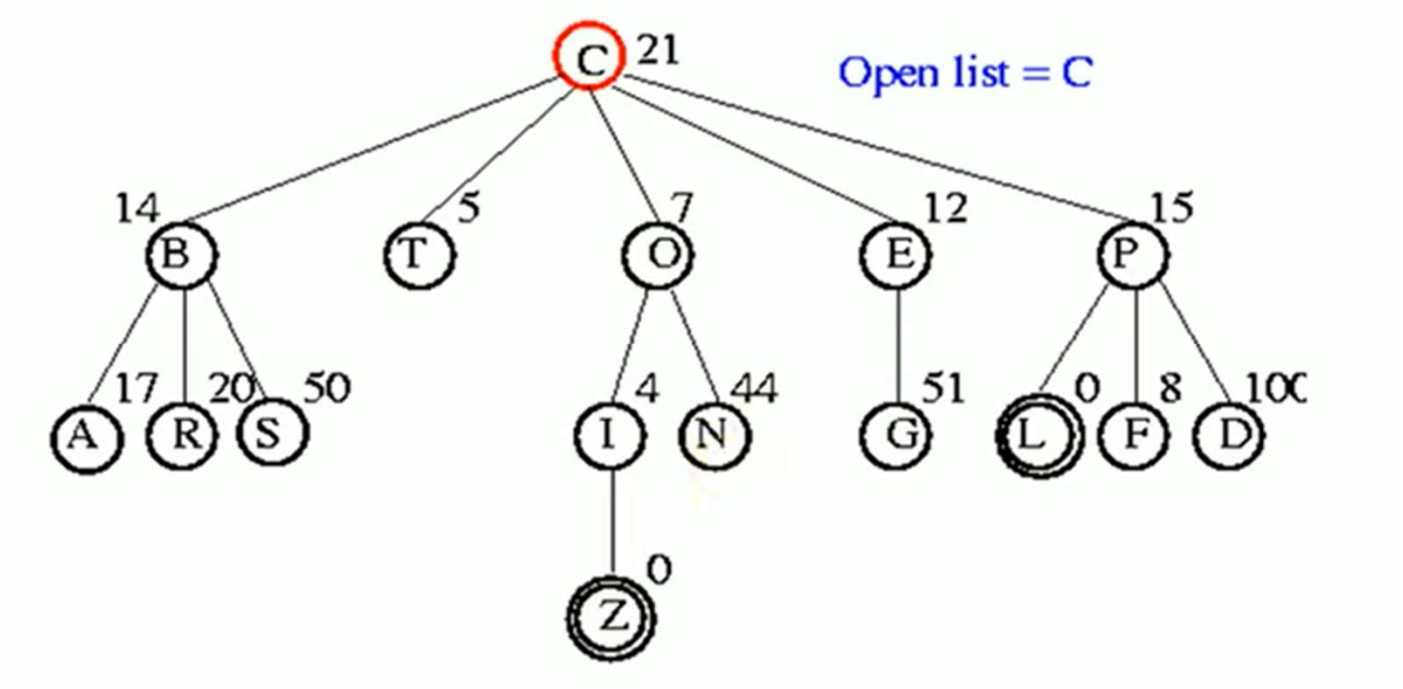
Stochastic hill climbing:

- Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make the initial state the current state.
- Repeat these steps until a solution is found or the current state does not change.
  - Select a state that has not been yet applied to the current state.
  - Apply the successor function to the current state and generate all the neighbor states.
  - Among the generated neighbor states which are better than the current state choose a state randomly (or based on some probability function).
  - If the chosen state is the goal state, then return success, else make it the current state and repeat step 2 of the second point.
- Exit from the function.

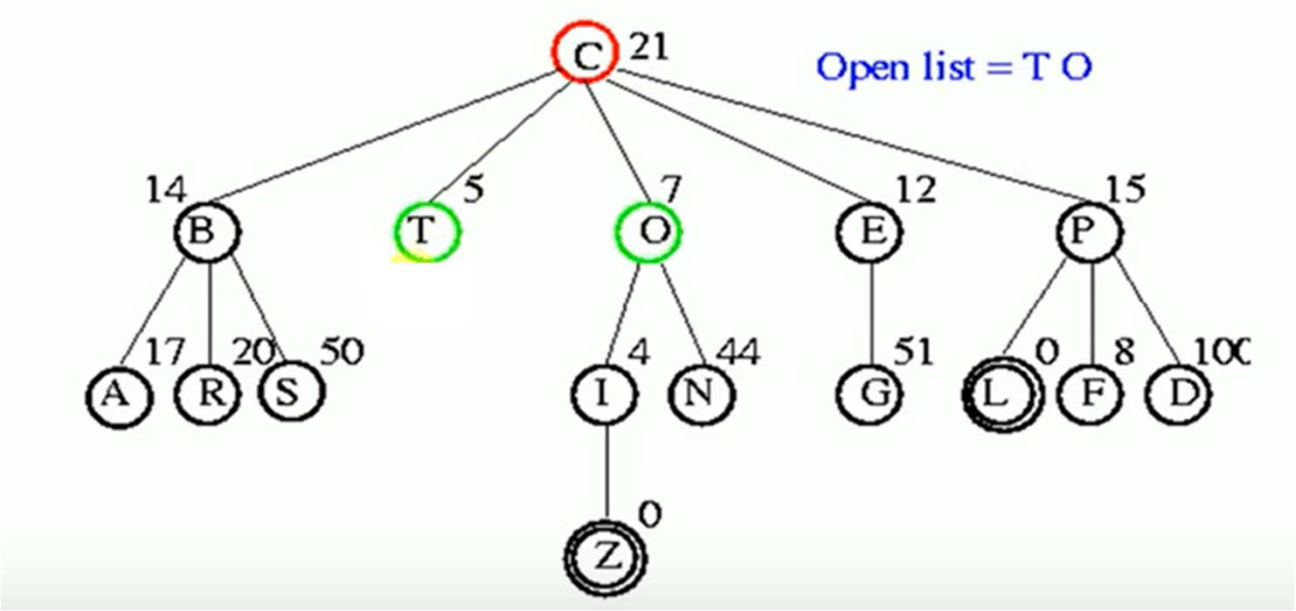
Beam Search Algorithm

- Beam Search only keeps the best (lowest-h) n nodes on open list
- n is the “beam width”
  - n = 1, Hill climbing
  - n = infinity, Best first search

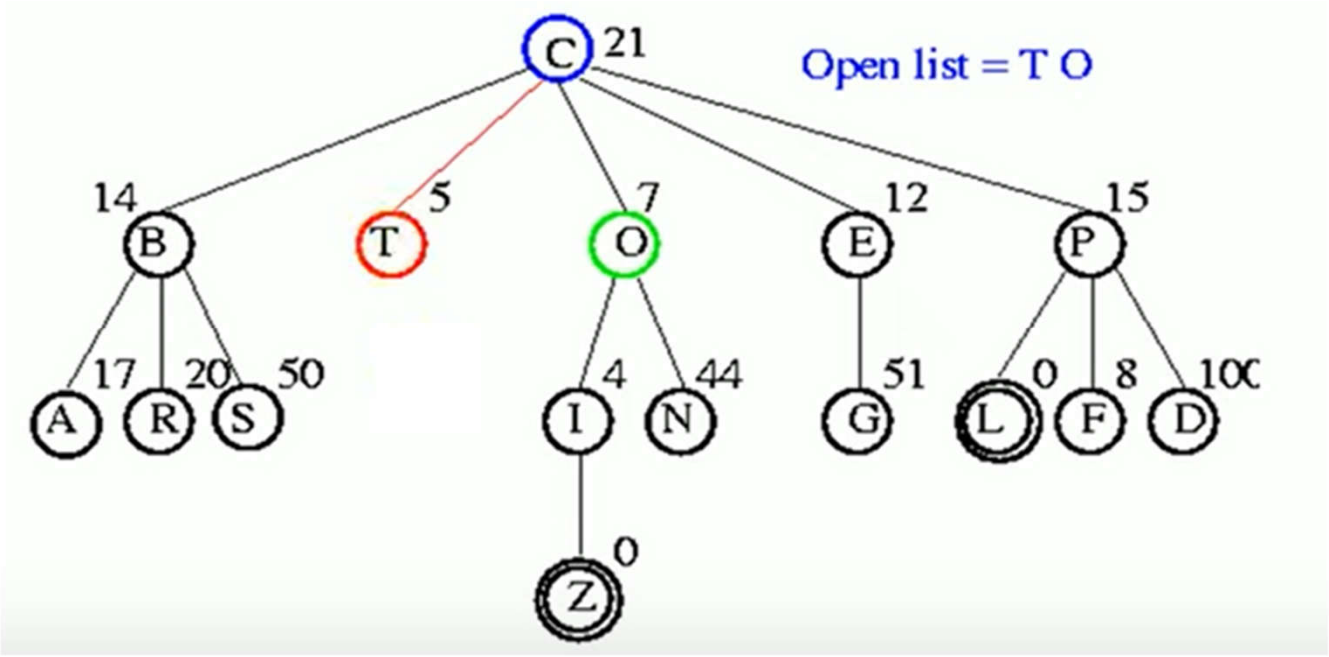
Beam Search Algorithm



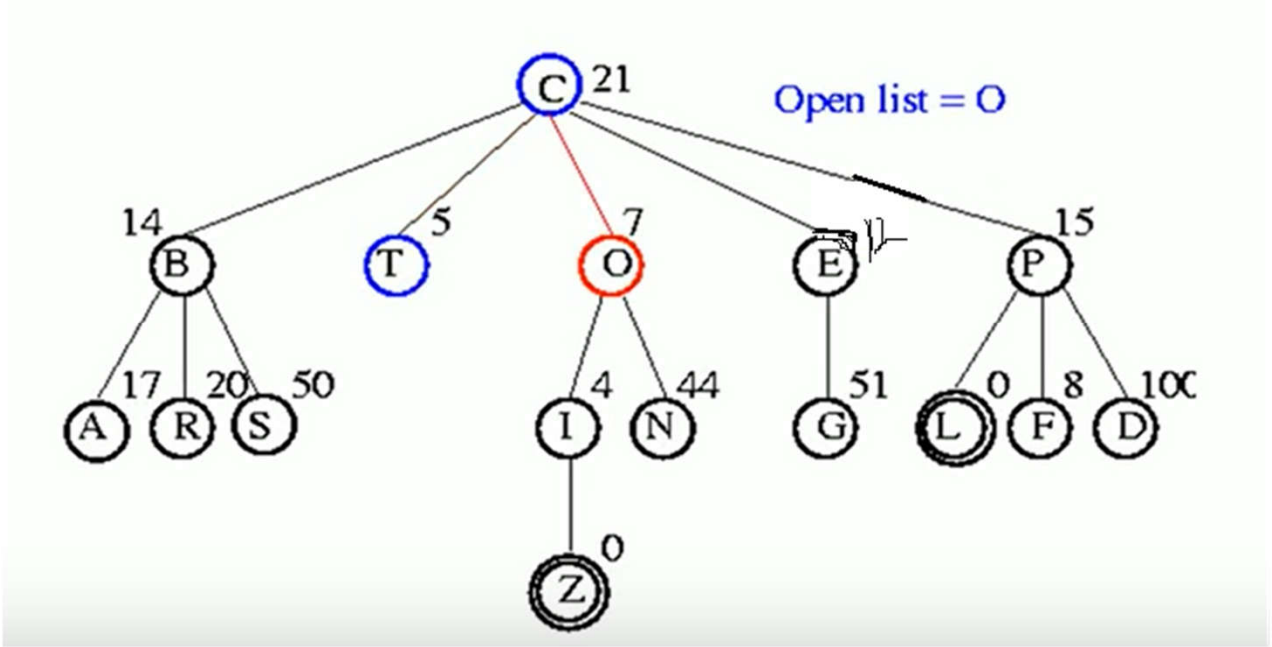
Beam Search Algorithm



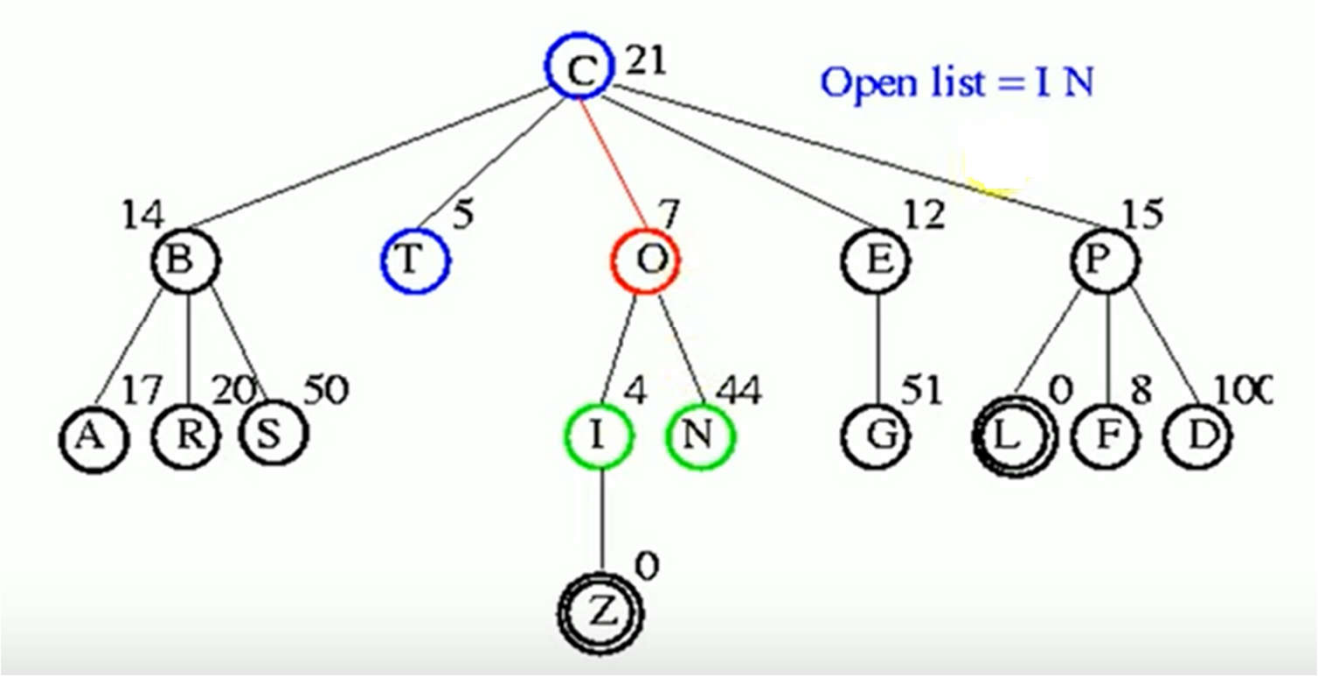
Beam Search Algorithm



Beam Search Algorithm

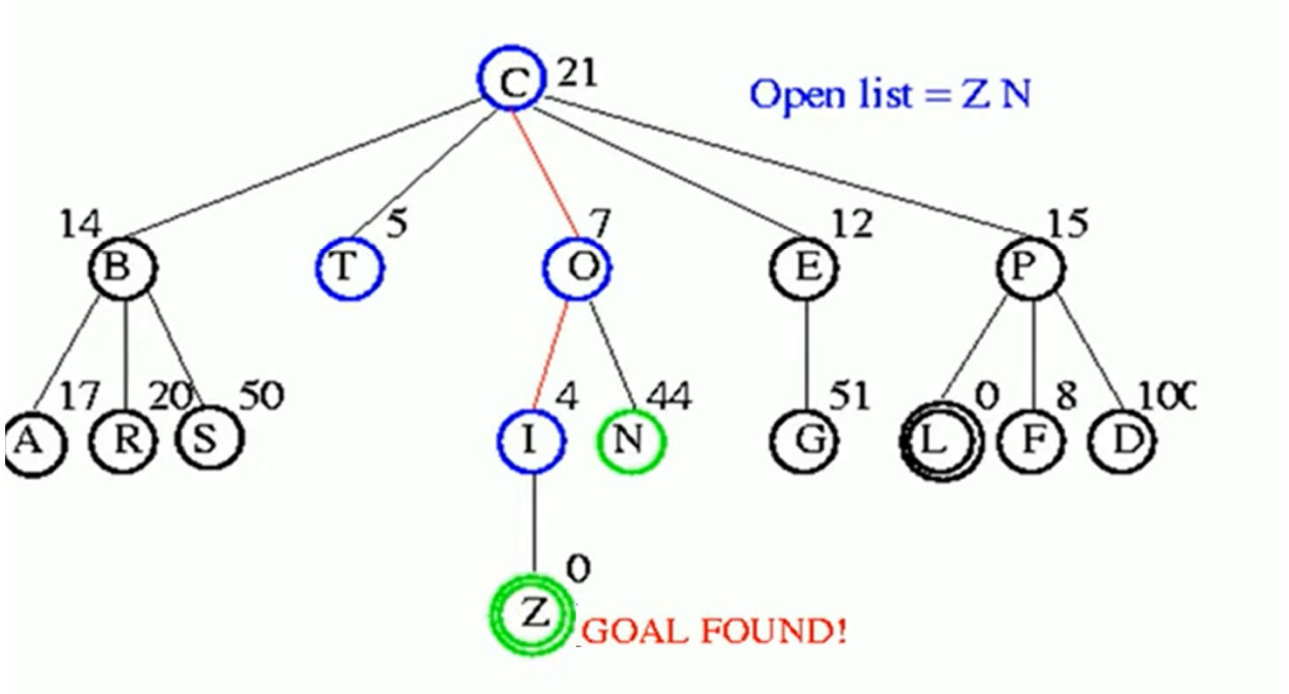


Beam Search Algorithm



Beam Search Algorithm

```
Disadvantage of BFS and Hill Climbing
1. BFS keep all the successors in open list.
2. Hill climbing will see the best node that is T, and T have
no successors
```



Simulated Annealing

- SA is a metaheuristic optimization technique introduced by Kirkpatrick et al. in 1983 to solve the Travelling Salesman Problem (TSP).
- The SA algorithm is based on the annealing process used in metallurgy, where a metal is heated to a high temperature quickly and then gradually cooled. At high temperatures, the atoms move fast, and when the temperature is reduced, their kinetic energy decreases as well. At the end of the annealing process, the atoms fall into a more ordered state, and the material is more ductile and easier to work with.
- Similarly, in SA, a search process starts with a high-energy state (an initial solution) and gradually lowers the temperature (a control parameter) until it reaches a state of minimum energy (the optimal solution).
- SA has been successfully applied to a wide range of optimization problems, such as TSP, protein folding, graph partitioning, and job-shop scheduling. The main advantage of SA is its ability to escape from local minima and converge to a global minimum.

Simulated Annealing

Algorithm 1: Acceptance Function

Data: T, ΔE - the temperature and the energy variation between the new candidate solution and the current one.

Result: Boolean value that indicates if the new solution is accepted or rejected.

```
if (ΔE < 0) then
    return True;
else
    r ← generate a random value in the range [0, 1)
    if (r < exp(−ΔE/T)) then
        return True
    else
        return False
    end
end
```

Simulated Annealing

A candidate solution with lower energy is always accepted. Conversely, a candidate solution with higher energy is accepted randomly with probability exp (Delta E / T)

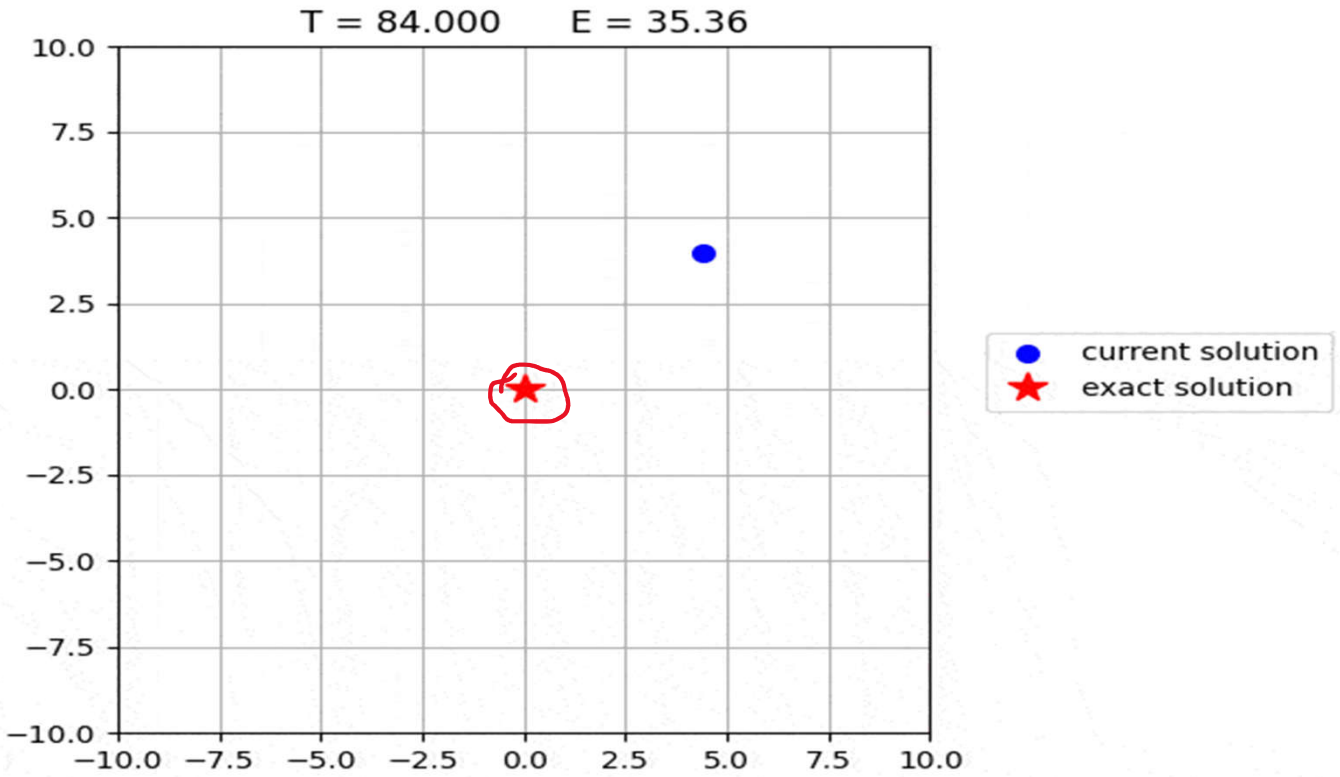
Simulated Annealing

Algorithm 2: Simulated Annealing Optimizer

$T \leftarrow T_{max}$
$\mathbf{x} \leftarrow$ generate the initial candidate solution
$E \leftarrow E(\mathbf{x})$ compute the energy of the initial solution
while $(T > T_{min})$ **and** $(E > E_{th})$ **do**
  $\mathbf{x}_{\text{new}} \leftarrow$ generate a new candidate solution
  $E_{\text{new}} \leftarrow$ compute the energy of the new candidate $\mathbf{x}_{new}$
  $\Delta E \leftarrow E_{new} - E$
  if Accept $(\Delta E, T)$ **then**
    $\mathbf{x} \leftarrow \mathbf{x}_{new}$
    $E \leftarrow E_{new}$
  end
  $T \leftarrow \frac{T}{\alpha}$ cool the temperature
end
return x

fixed threshold E_{th}

To better understand the algorithm, we use SA to illustrate the minimization of the function $f(x, y) = x^2 + y^2$. We used as search space a grid of size 101 × 101 placed in the square area defined by $(x, y) \in [-10, 10] \times [-10, 10]$. We set the cooling rate $\alpha = 0.84$ and the initial solution $(x, y) = (4, 4)$. At each step, a new solution is generated by randomly shifting the current solution by ± 0.2 in x and y direction.



Simulated Annealing