Beyond Classical Search

Unit-II

Local Search Algorithms

- The informed and uninformed search expands the nodes systematically in two ways:
  - keeping different paths in the memory and
  - selecting the best suitable path,
- In "local search algorithms", the path cost does not matters, and only focus on solution-state needed to reach to the goal node.
- A local search algorithm completes its task by traversing on a single current node rather than multiple paths and following the neighbors of that node.
- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution.

Local Search Algorithms

- ***Advantages***:
  1. Local search algorithms use a very little or constant amount of memory.
  2. They find a reasonable solution in large or infinite state spaces where the classical or systematic algorithms do not work suitably.
- Local search algorithms can also solve optimization problems, in which the aim is to find the best state according to an objective function.

Local Search Algorithms…

- The local search algorithm is also work for a pure optimized problem, where a pure optimization problem is one where all the nodes can give a solution.
- But the target is to find the best state out of all according to the objective function.
- An objective function is a function whose value is either minimized or maximized in different contexts of the optimization problems.

Local Search Algorithms…

- To understand the local search algorithms, consider the below state-space landscape diagram having:
- Location: It is defined by the state.
- Elevation: It is defined by the value of the objective function or heuristic cost function.

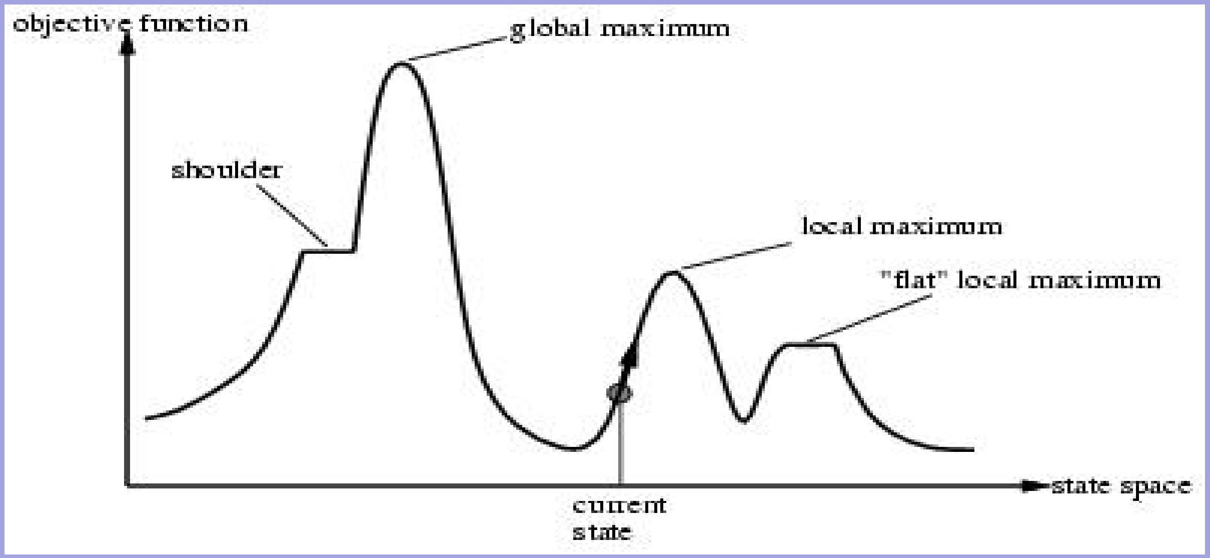
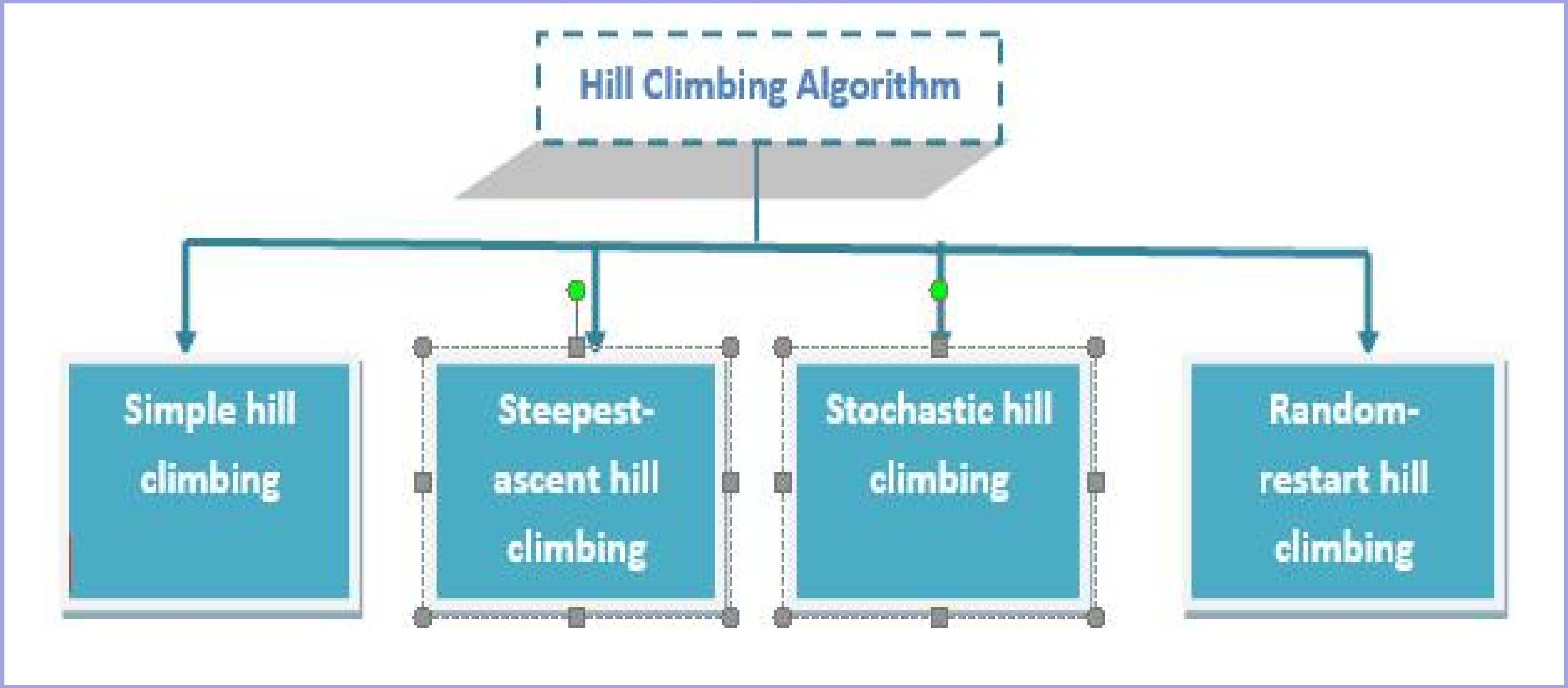


Figure: A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum.

Local Search Algorithms…

- The local search algorithm explores the above landscape by finding the following two points:
- Global Minima: If the elevation corresponds to the cost, then the task is to find the lowest valley, which is known as Global Minimum.
- Global Maxima: If the elevation corresponds to an objective function, then it finds the highest peak which is called as Global Maxima. It is the highest point in the valley.
- Different types of local searches:
  - Hill-climbing Search
  - Simulated Annealing
  - Local Beam Search

Local Search Algorithms…



Hill climbing

- It is often used when a good heuristic function is available for evaluating states.
- But when no other useful knowledge is available. This algorithm is simply a loop that continuously moves in the direction of increasing value i.e., uphill.
- It terminates when it reaches a “peak” where no neighbor has a higher value.
- The algorithm doesn’t maintain a search tree, so the current node data structure only records the state and its objective function value.
- Hill – climbing doesn’t look ahead beyond the immediate neighbors of the current state.

Simple Hill climbing algorithm

1. Evaluate the initial state (IS). If it is the goal state (GS) , then return it and quit. Else consider IS as the current state (CS) and proceed.
2. Loop until a solution is found or there are no new operator (OP) to be applied to the CS.
   a) Select an OP that has not yet been applied to the CS and apply it to produce a new state (NS).
   b) Evaluate the NS:
      - If NS is a GS , then return it and quit.
      - If it is not a GS but better than the CS, then consider it as the current state (CS) and proceed.
      - If NS is not better than CS then continue in the loop by selecting the next appropriate OP for CS.

Steepest – Ascent Hill climbing algorithm

- It considers all the moves from the CS and selects the best one as the next state.
- It is also known as a greedy approach
- It is also called gradient search (or gradient ascent/descent).

Algorithm:

1. Evaluate the initial state (IS). If it is the goal state (GS) , then return it and quit. Else consider IS as the current state (CS) and proceed.
2. Loop until a solution is found or until a complete iteration produces no change to the CS:

Steepest – Ascent Hill climbing algorithm

a) Let successor (SUC) be a state such that any NS that can be generated from CS is better than SUC. (i.e., setting SUC to a minimum value at the beginning of an iteration or set CS as SUC)

b) For each operator OP that applies to the CS do:

   I. Apply OP to CS and generate a NS.

   II. Evaluate the NS. If it is a GS then return it and quit. If not , compare it with SUC. If NS is better than SUC, then set SUC to NS; else leave SUC unchanged.

c) If the SUC is better than CS, then set CS to SUC (i.e., move to the next best state)

Steepest Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

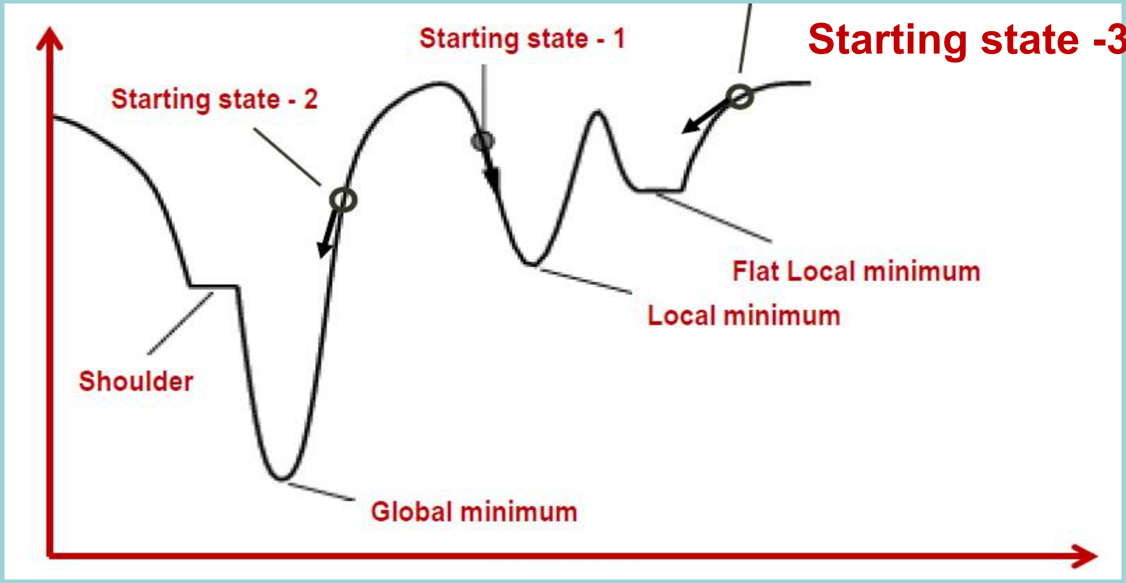
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
       neighbor ← a highest-valued successor of current
       if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
       current ← neighbor
```

Stochastic hill climbing search

```
current position = initial solution;
repeat
    for All neighbours of current position do
        Obtain a random neighbour;
        if cost of neighbour ≤ cost of current position then
            current position = neighbour position;
            break;
        end
    end
until cost of current position ≤ cost of all its neighbours;
```

Random-restart hill climbing search

- 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.



Limitations: Hill-climbing search

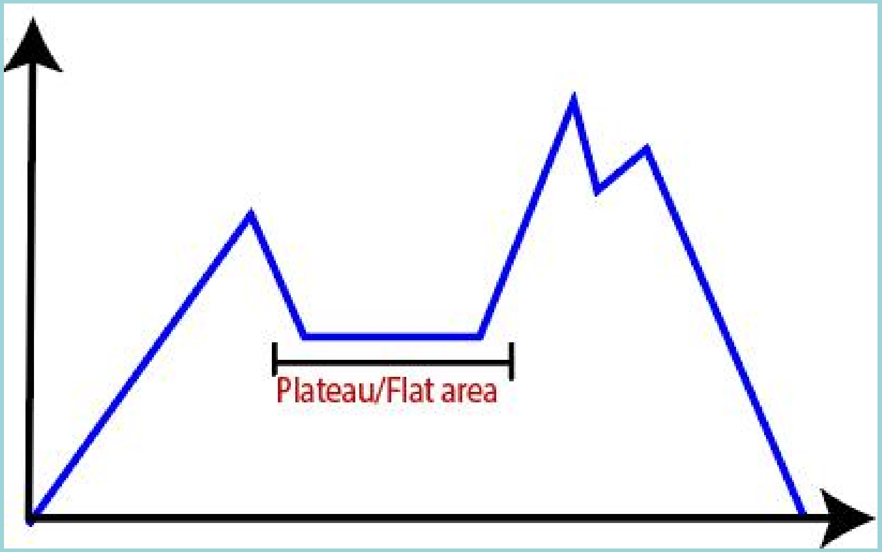
- Most of the hill climbing search algorithms may fail to find a optimum solution.
- Either algorithm may terminate not by finding a goal state but by getting to a state from which no better states can be generated.
- Hill Climbing is not complete; Unless we introduce backtracking
- Hill Climbing is not optimal; Solution found is a local optimum
- This will happen if the program has reached either a local maximum, a plateau or a ridge.

Limitations: Hill-climbing search

- A **local maximum** is a state that is better than all its neighbors but is not better than some other states further away.
- Solution of local maxima:-
  - Move in some arbitrary direction
  - Back track to an ancestor and try some other alternatives.

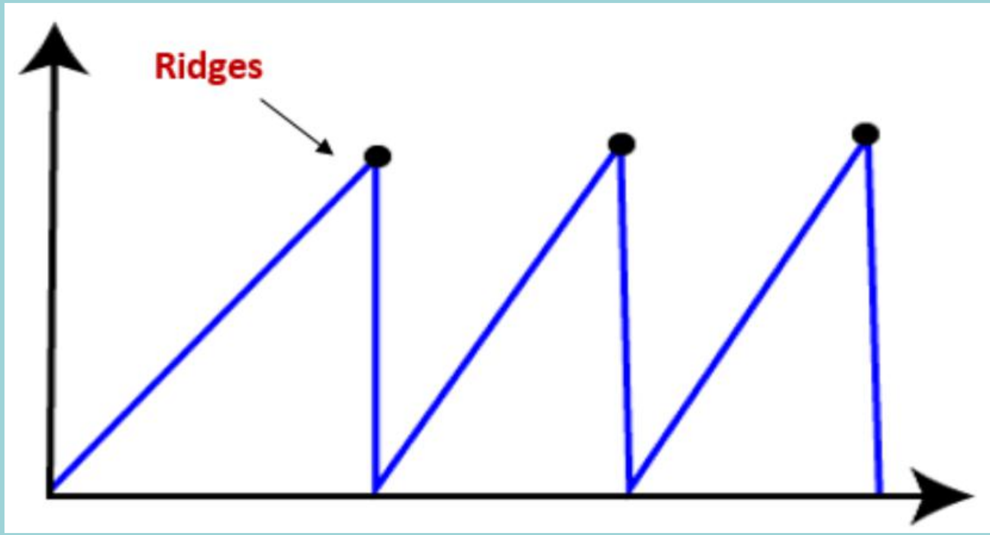
Limitations: Hill-climbing search

- **A plateau** is a flat area in the search space in which all the neighboring states have the same heuristic function value.
- Solution of plateau
  - Expand few generation ahead to move to a different section of the search space.



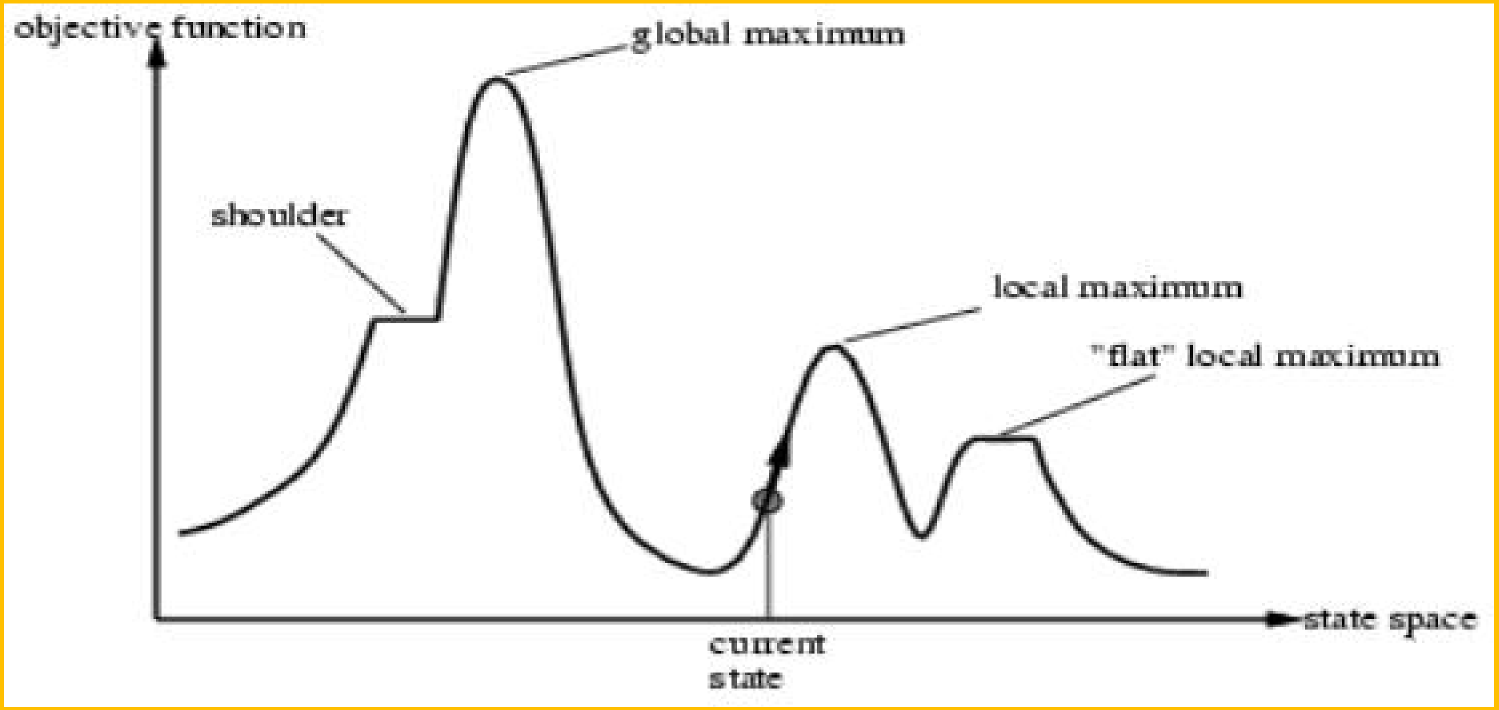
Limitations: Hill-climbing search

- **A ridge** is an area in the search space which is higher than its surroundings but itself has slopes.
- It is not possible to traverse a ridge by a single move i.e., no such operator is available.
- Solution of ridges:-
  - Apply several operators before doing the evaluation



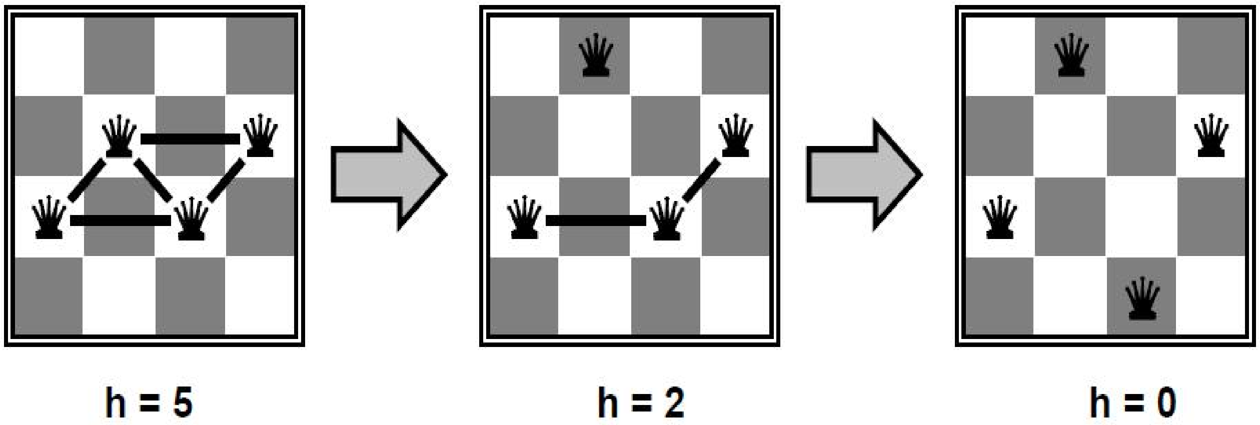
Limitations: Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima



Hill-climbing search: n-queens problem

- Objective/Goal: Put n queens on an n x n board with no two queens on the same row, column, or diagonal which means no queen attacking another.



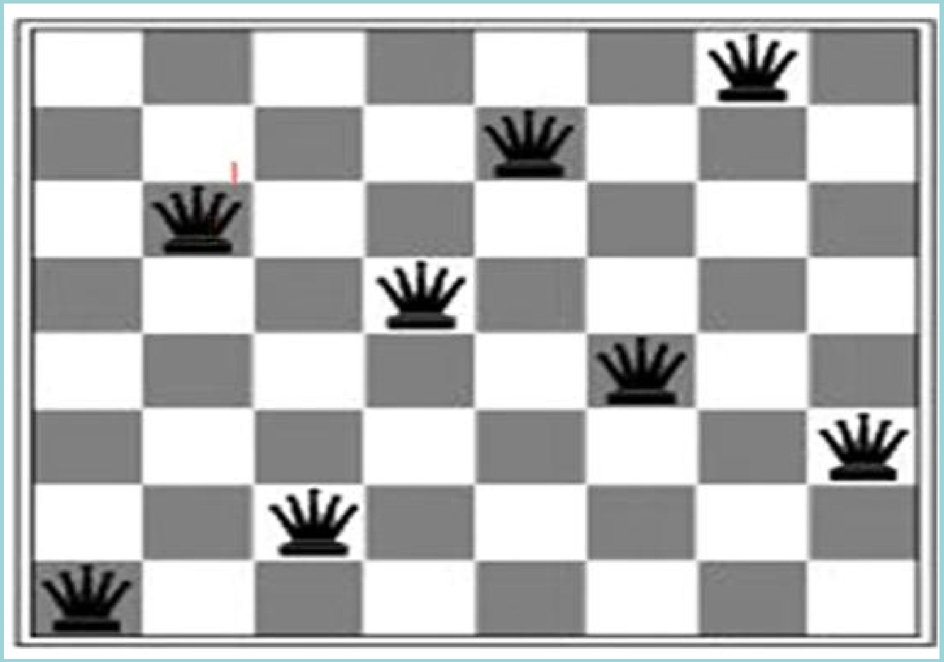
- States: n queens on board, one per column
- Actions: move a queen in its column
- Heuristic value function: number of conflicts
- Move a queen to reduce number of conflicts

Hill-climbing search: 8-queens problem

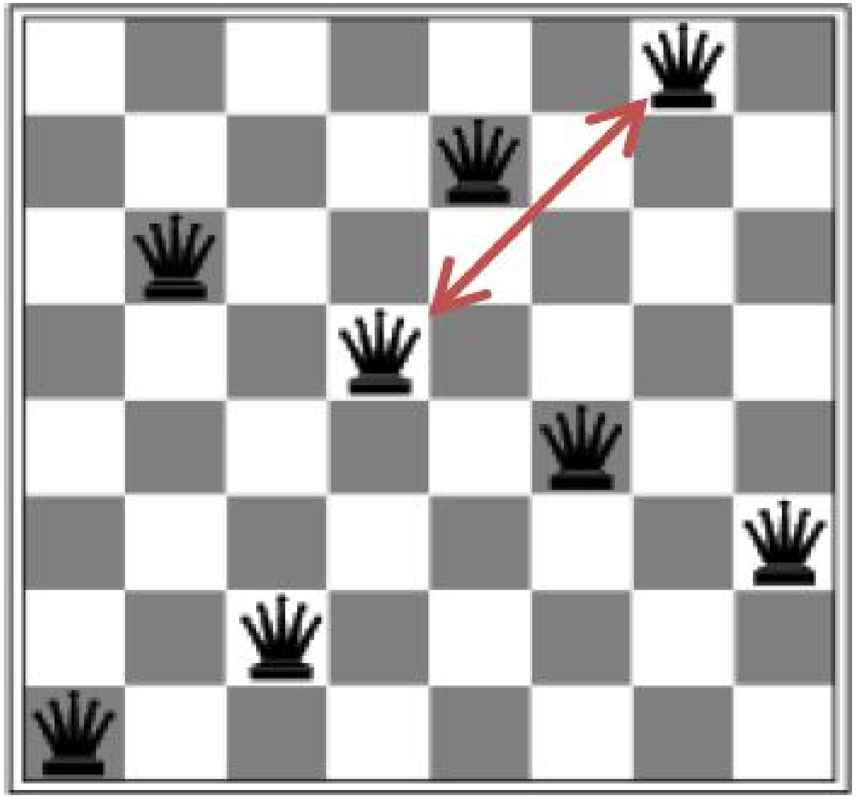
- *h* = number of pairs of queens that are attacking each other, either directly or indirectly.
- *h = 17* for the above state.
- Therefore the best greedy move is to move a queen to a square labeled with 12.

Hill-climbing search: 8-queens problem

- What is h value here? Is it global minimum?

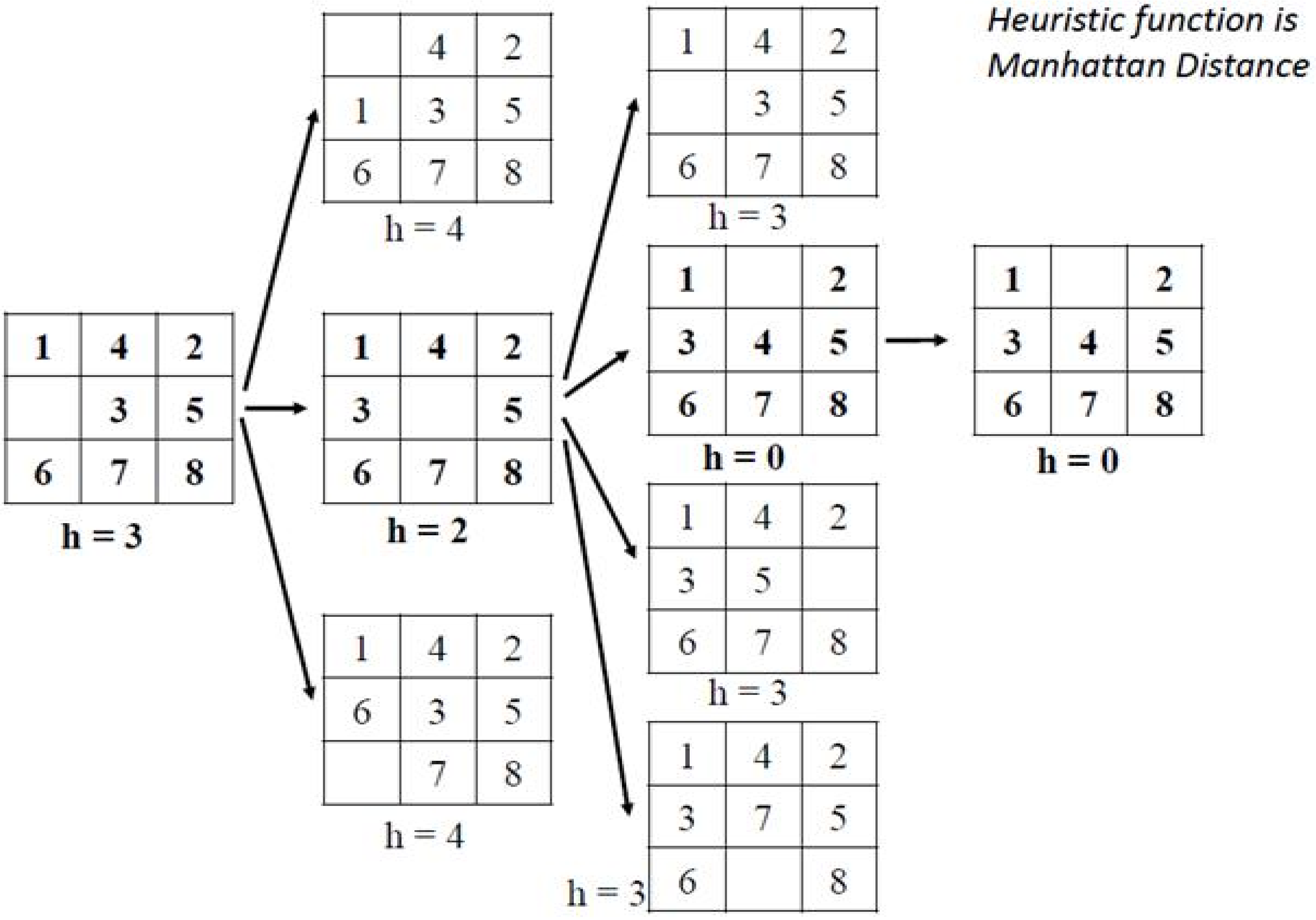


Hill-climbing search: 8-queens problem



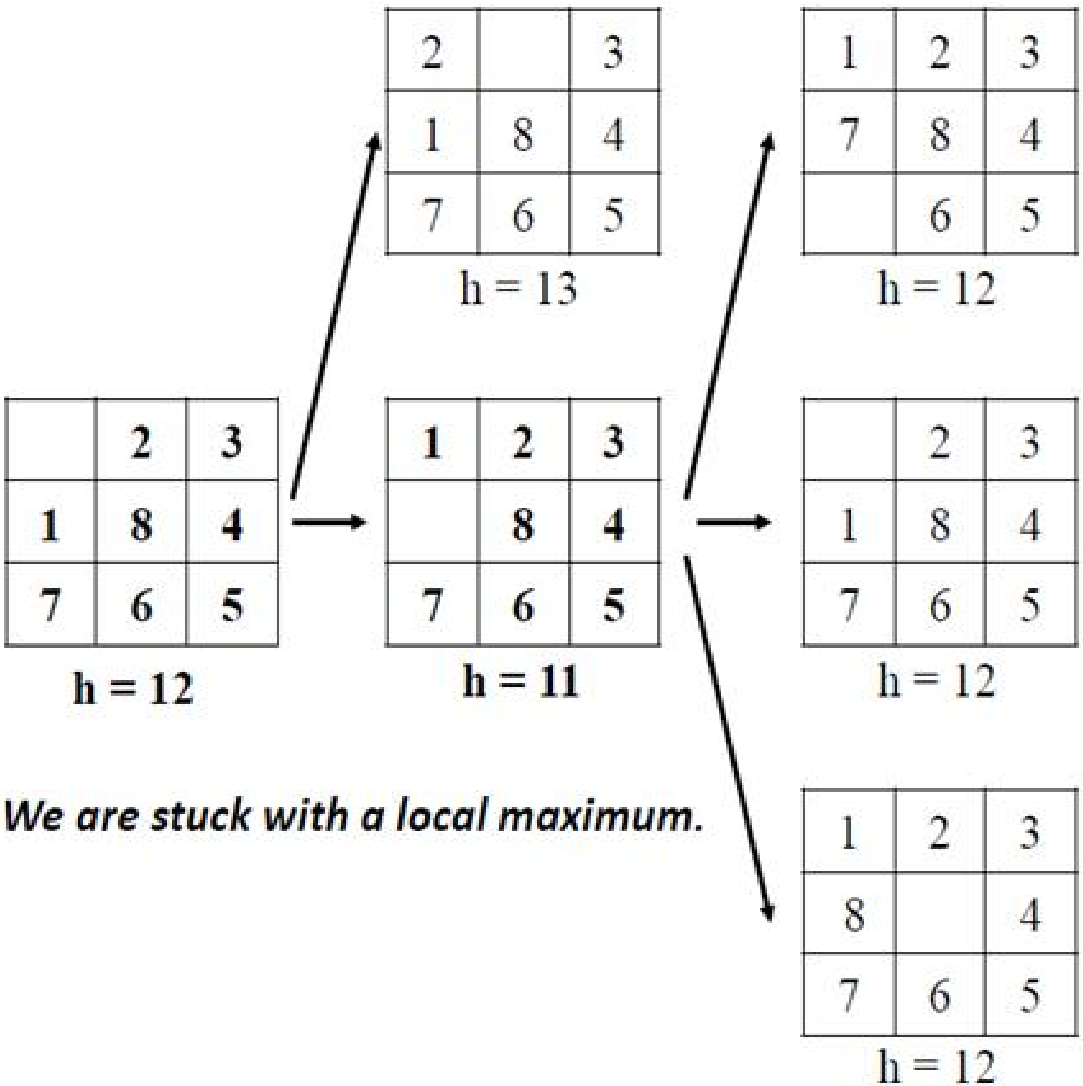
- A local minimum with *h* = *1*

Hill-climbing search: 8-puzzle problem



Hill-climbing example: 8-puzzle problem

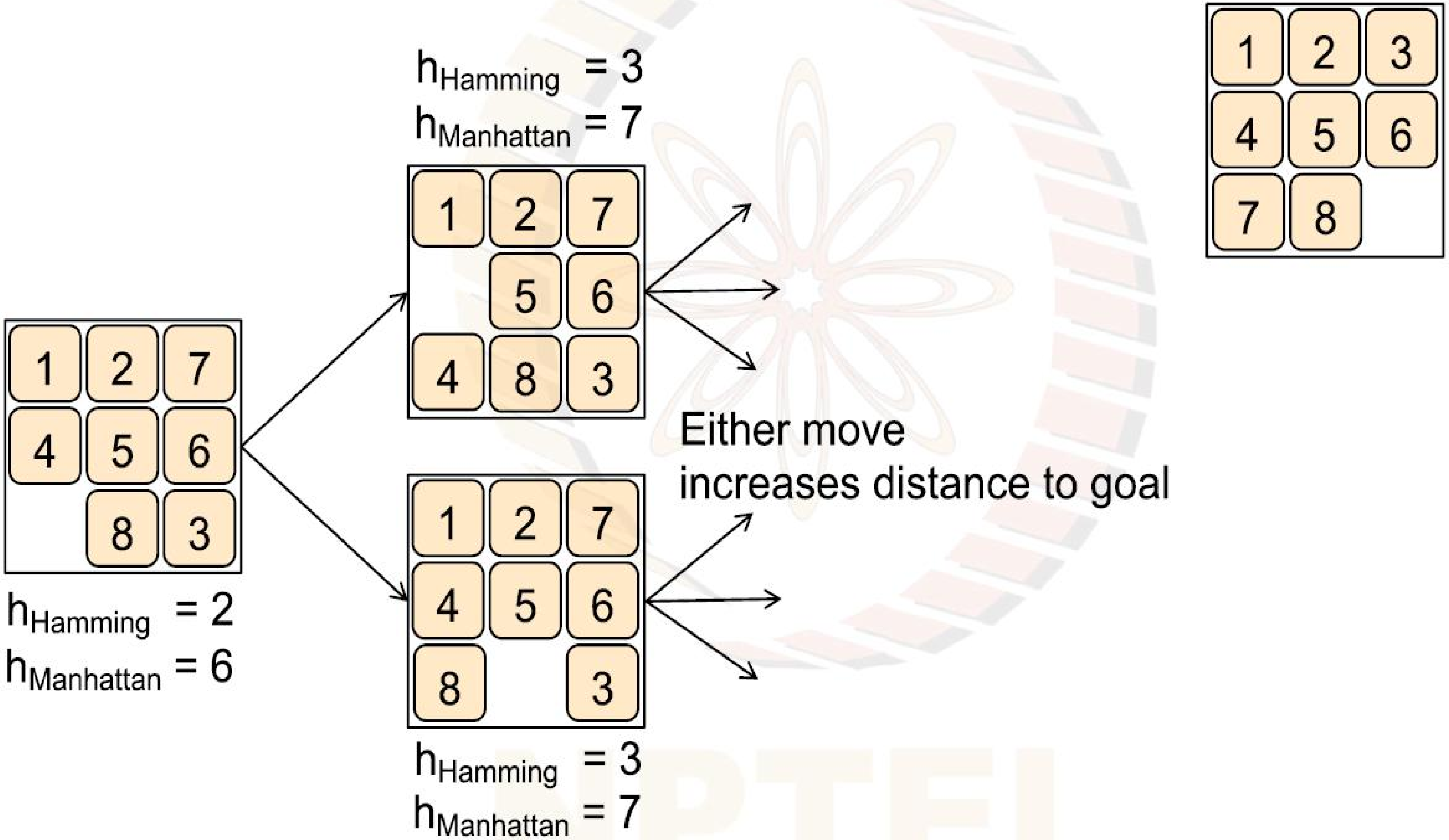
Heuristic function is Manhattan Distance



We are stuck with a local maximum.

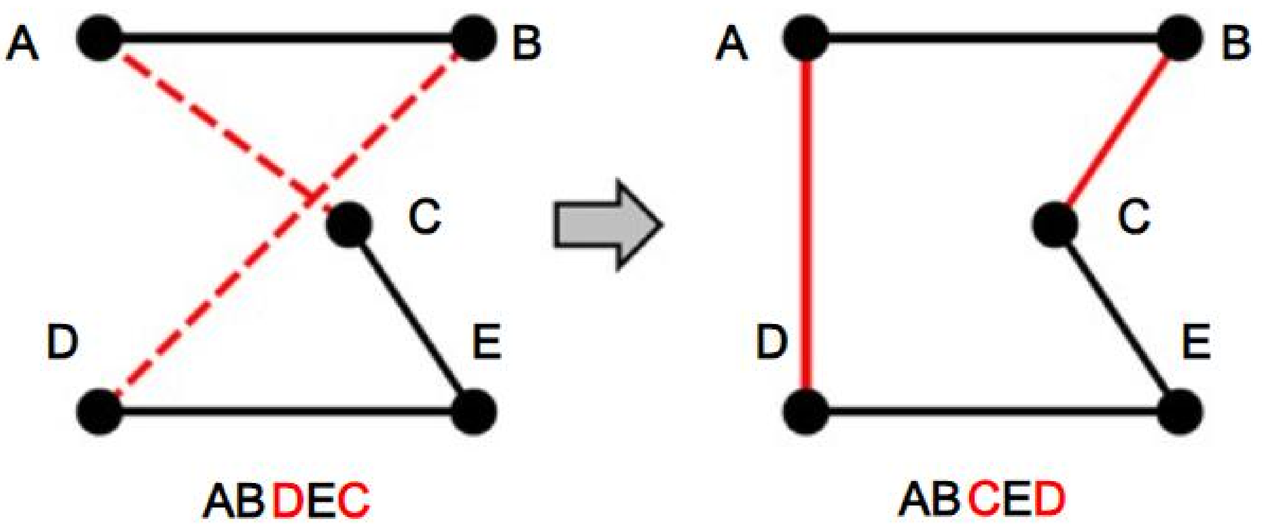
Hill-climbing example: 8-puzzle problem

8-puzzle: A local minimum



Hill-climbing example: Traveling Salesman Problem(TSP)

- Find the shortest tour connecting n cities
- **State space:** all possible tours
- **Objective function:** length of tour
- What's a possible local improvement strategy?
  - Start with any complete tour, perform pairwise exchanges



Simulated Annealing

- A **hill-climbing algorithm** that never makes “downhill” moves toward states with lower value (or higher cost) is guaranteed to be **incomplete**, because ***it can get stuck on a local maximum***.
  - In contrast, a **purely random walk**—that is, moving to a successor chosen uniformly at random from the set of successors—is **complete but extremely inefficient**.
  - Therefore, it seems reasonable to ***combine hill climbing with a random walk in some way that yields both efficiency and completeness***.
- Simulated Annealing (SA) is applied to solve optimization problems.
- SA is a stochastic algorithm.
- Simulated Annealing is allow moves to inferior neighbors with a probability that is regulated over time.
- Escape local maxima by allowing some "bad" moves but gradually decrease their probability.

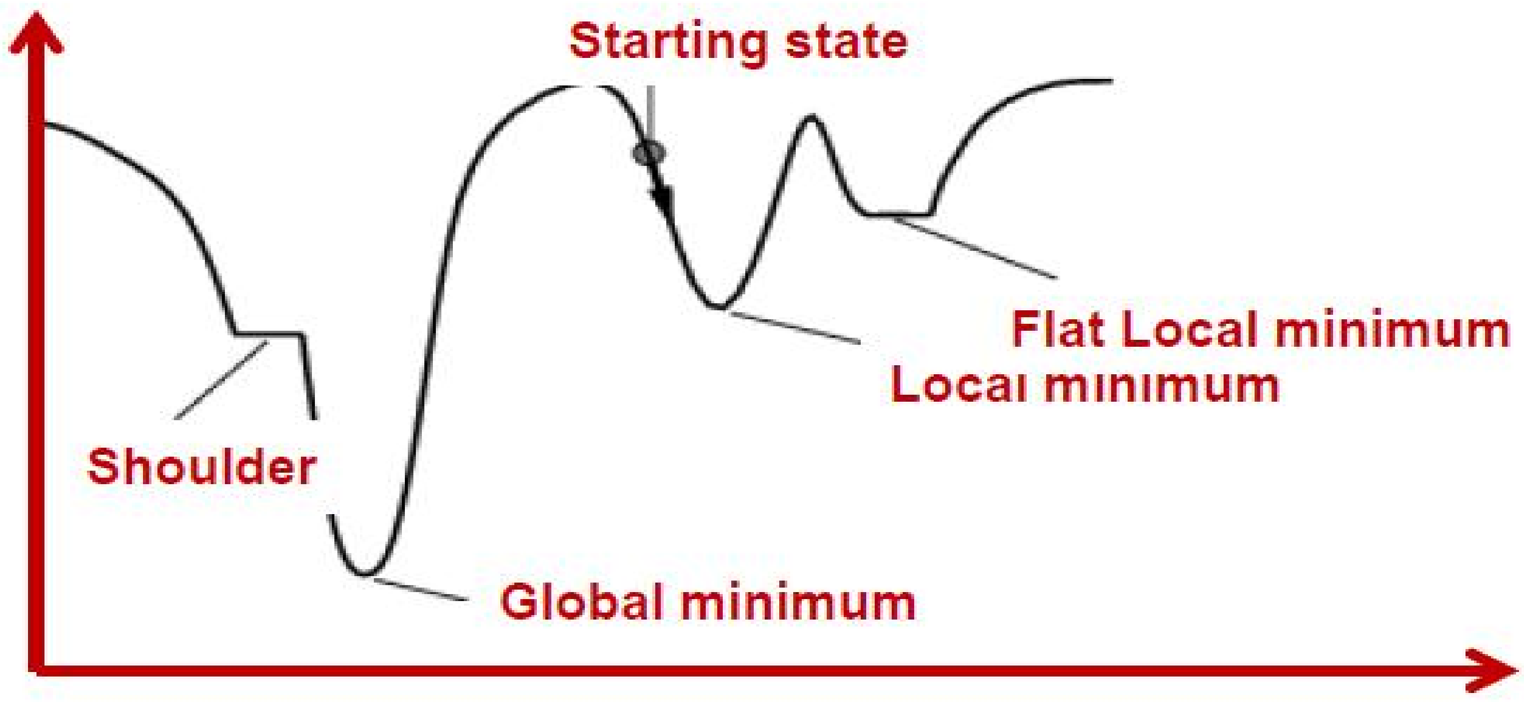
Simulated Annealing

- The probability is controlled by a parameter called temperature
- Higher temperatures allow more bad moves than lower temperatures
- Annealing: Lowering the temperature gradually
- Quenching: Lowering the temperature rapidly .
- It can be proven that: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1.
- SA is motivated by the physical annealing process.
- Method proposed in 1983 by IBM researchers for solving VLSI layout problems (Kirkpatrick et al, *Science*, 220:671-680, 1983). It will always find the global optimum
- Other applications: Traveling salesman, Graph partitioning, Graph coloring, Scheduling, Facility Layout, Image Processing, useful for some problems, but can be very slow.

Simulated Annealing

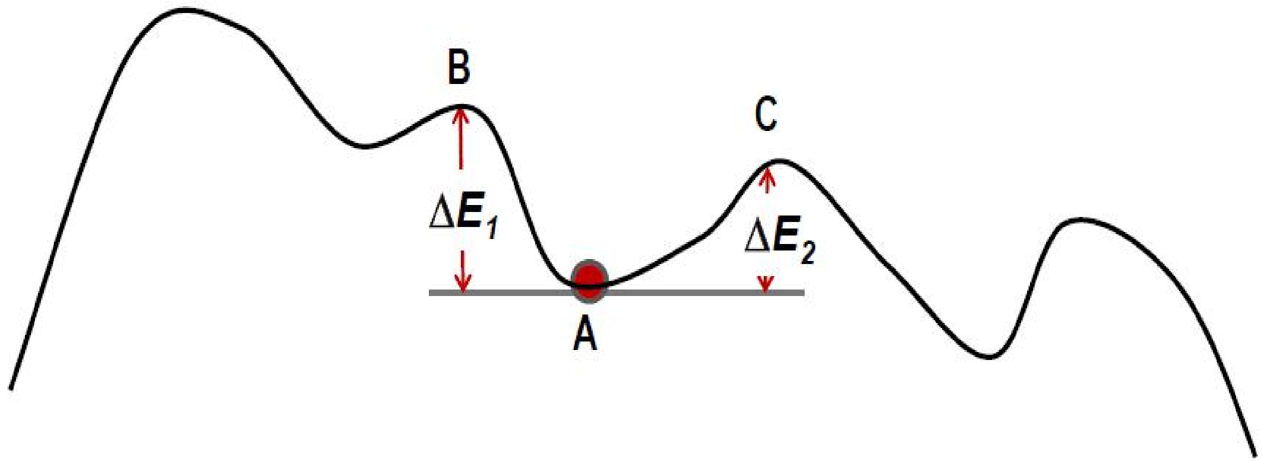
```
function SIMULATED-ANNEALING( problem, schedule) return a solution state
    input: problem, a problem
           schedule, a mapping from time to temperature
    local variables: current, a node.
                     next, a node.
                     T, a "temperature" controlling the prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] - VALUE[current]
        if ΔE<0 then current ← next
        else current ← next only with probability e^(-ΔE/T)
```

Simulated Annealing



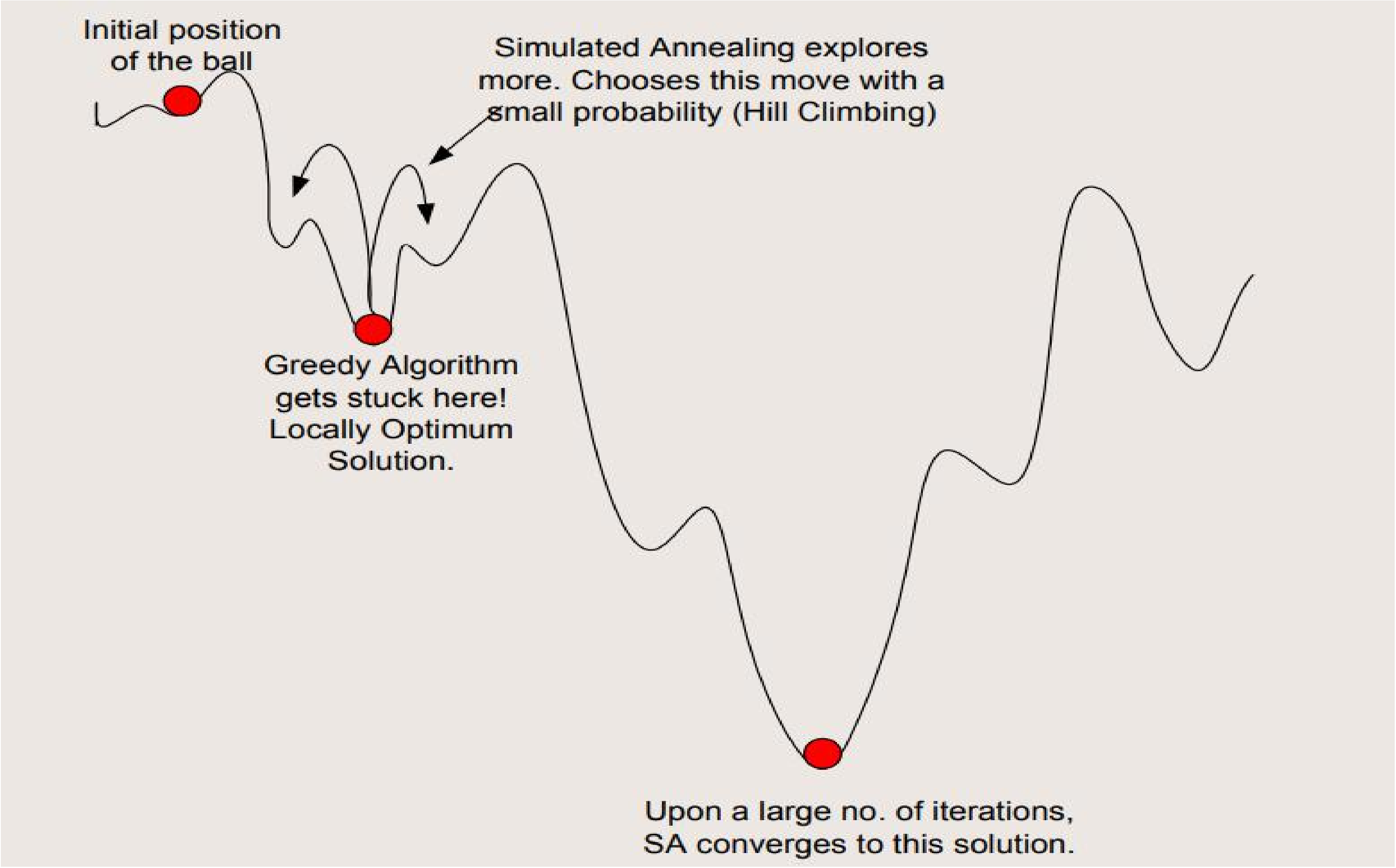
Simulated Annealing

Probability of making a bad move = $e^{-\Delta E / T} = \frac{1}{e^{\Delta E / T}}$



Since $\Delta E_1 > \Delta E_2$ moving from A to C is exponentially more probable than moving from A to B

Simulated Annealing

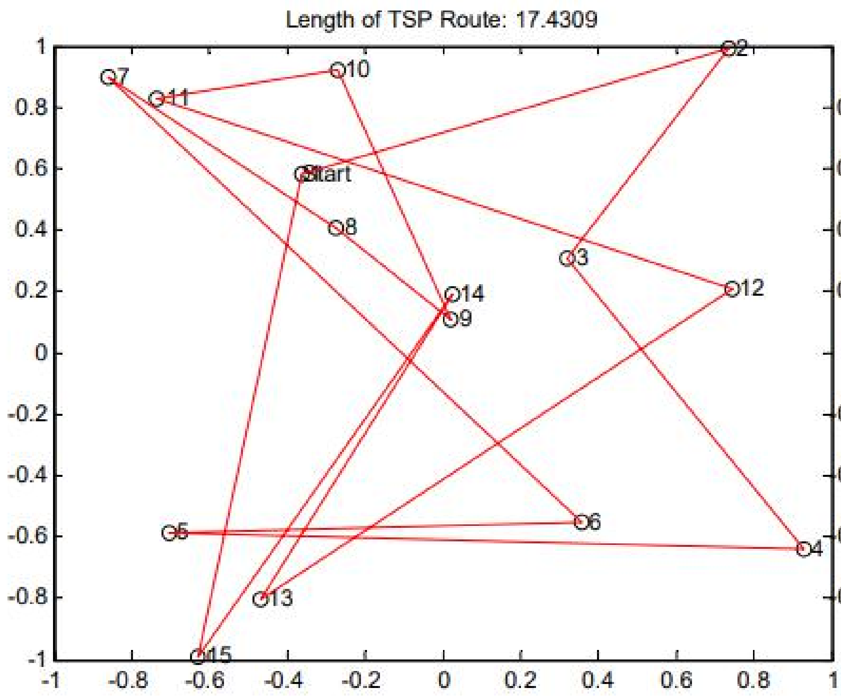


Simulated Annealing

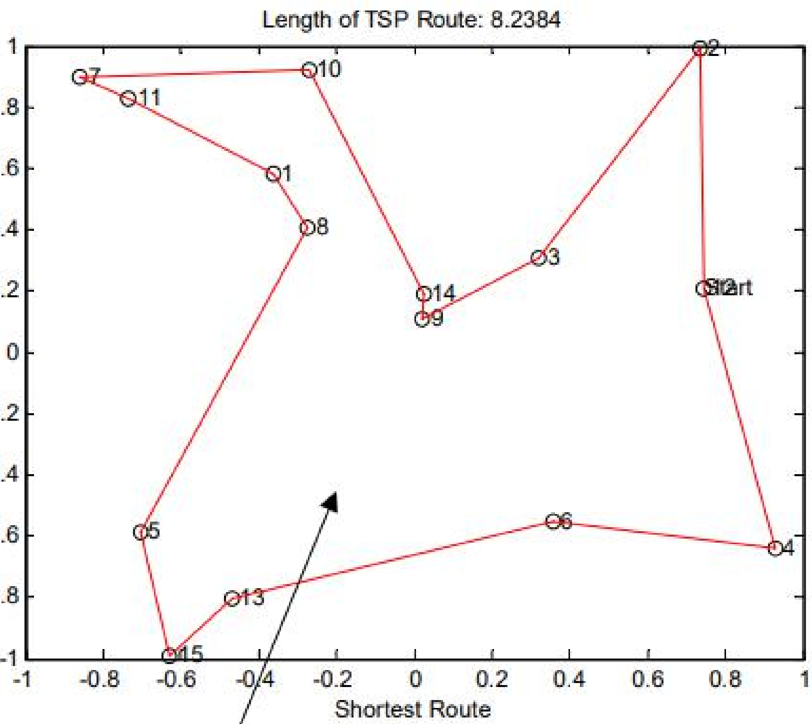
- The SA algorithm is based on the annealing process used in metallurgy, where a metal is heated to a high temperature quickly and then gradually cooled.
- At high temperatures, the atoms move fast, and when the temperature is reduced, their kinetic energy decreases as well.
- At the end of the annealing 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).

Simulated Annealing

Initial (Random) Route Length: 17.43



Final (Optimized) Route Length: 8.24



Result with SA

Local beam Search

- Keeping just one node in memory might seem an extreme reaction to the problem of memory limitation, but this is what we do in local searches and in simulated annealing.
- Local Beam Search is a heuristic search algorithm used for optimization problems.
- Beam search is an optimization of **best-first search** that reduces its memory requirements.
- It is used to explore a large search space efficiently by focusing on a subset of the search space.
- The main advantage of local beam search is that it can explore a large search space efficiently by focusing on the best subset of solutions.
- This can help to avoid getting stuck in local optima and improve the quality of the solutions found

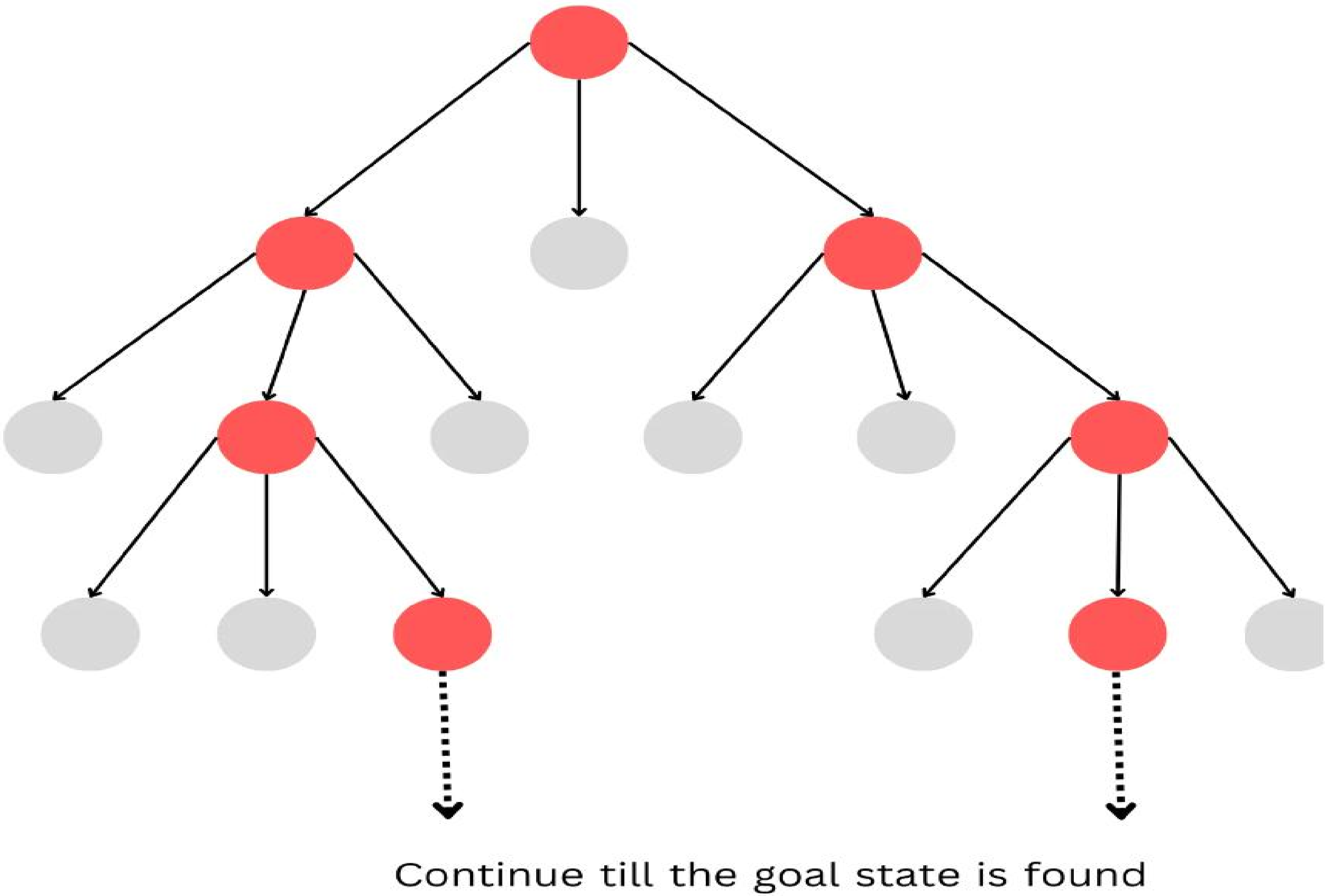
Local beam Search: Algorithm

function BEAM-SEARCH(*problem*, k) **returns** a solution state
  start with k randomly generated states
  loop
    generate all successors of all k states
    if any of them is a solution **then** return it
    else select the k best successors

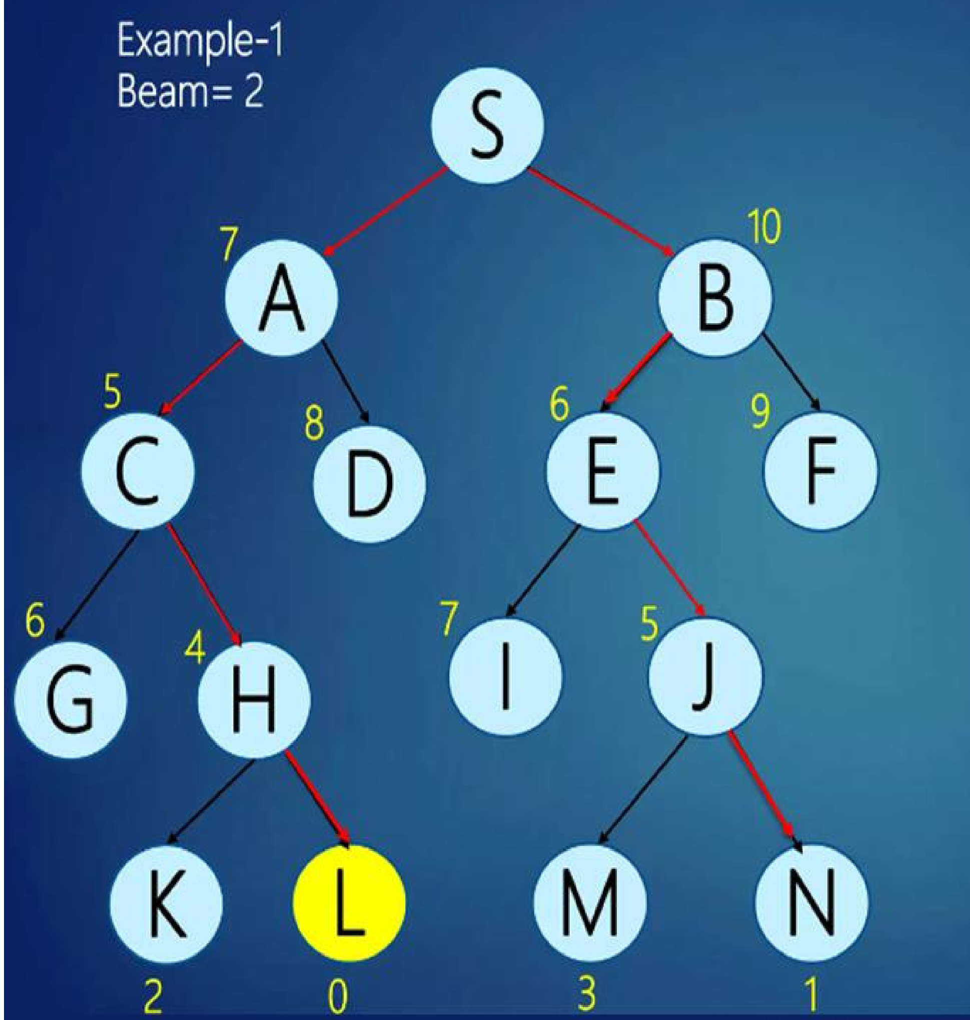
Local beam Search…

- Idea: Keeping only one node in memory is an extreme reaction to memory problems.
- Keep track of *k* states instead of one
- Initially: *k* randomly selected states
- Next: determine all successors of *k* states
- If any of successors is goal->finished
- Else select *k* best from successors and repeat.
- Example: https://www.codecademy.com/resources/docs/ai/search-algorithms/beam-search

Local beam Search



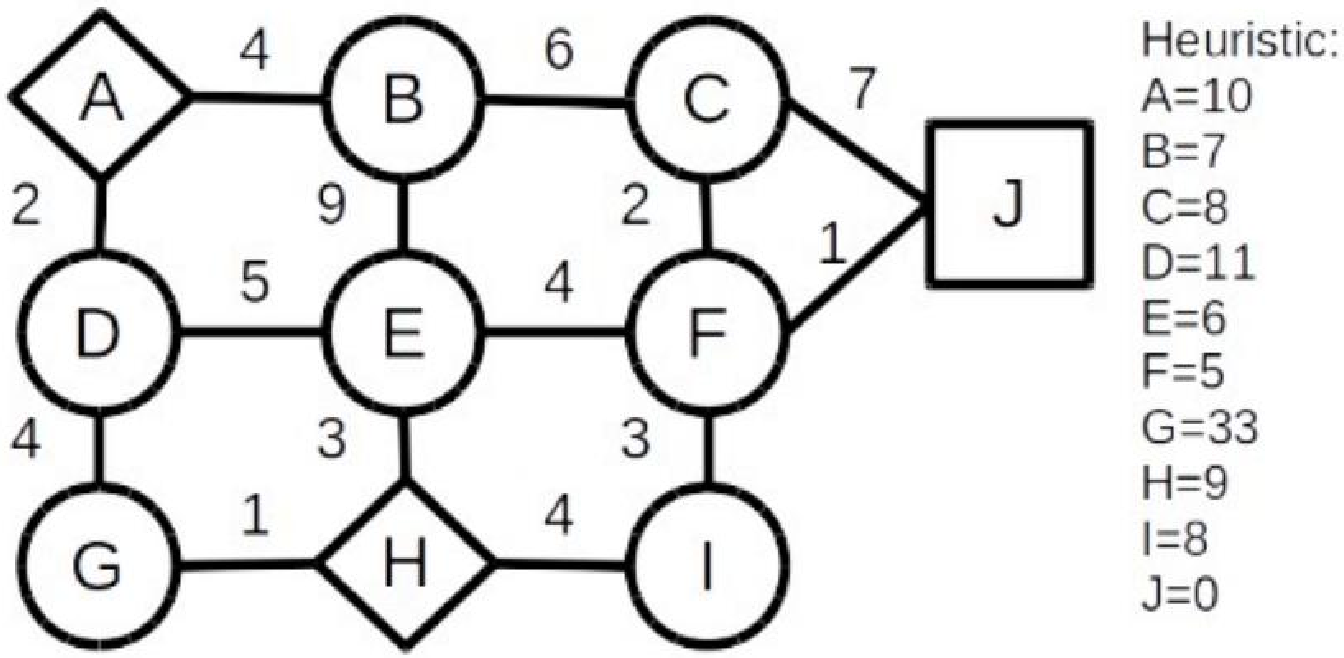
Local beam Search:Example



Frontier List	Explored List
AB	S
CE	SAB
HJ	SABCE
LN	SABCEHJ
	SABCEHJLN

Local beam Search…

- **Question:** Perform Local Beam Search With 2 Beams On The Graph With Edge Costs And Heuristic Values Below. Initial Nodes: A, H Goal Node: J



Stochastic Local beam Search

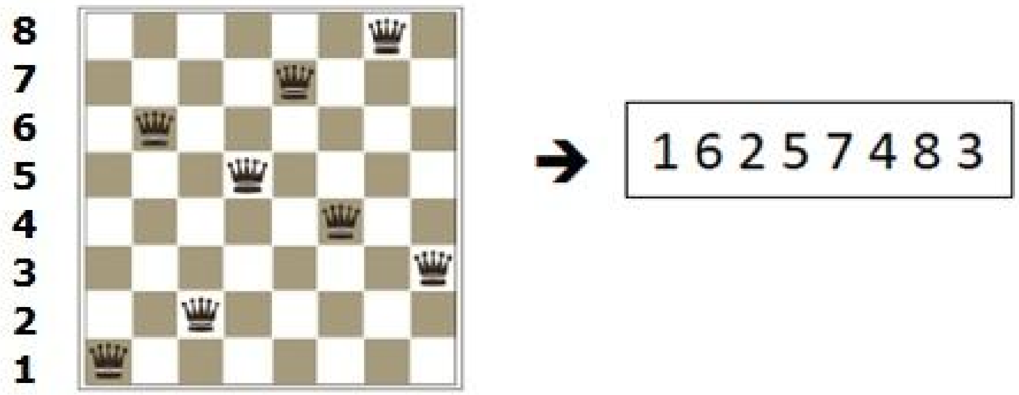
- Local beam search is not the same as *k random-start searches run in parallel!*
- Searches that find good states recruit other searches to join them
- Problem: quite often, all *k states end up on same local hill*
- Idea: Stochastic beam search
- Choose *k successors randomly, biased towards good ones.*
- Observe the close analogy to natural selection!
- Instead of choosing the best k from the pool of candidate successors, **stochastic beam search** chooses k successors at random, with the probability of choosing a given successor being an increasing function of its value.
- **Stochastic beam search** bears some resemblance to the process of natural selection, whereby the "successors" (offspring) of a "state" (organism) populate the next generation according to its "value" (fitness).

Genetic algorithm

- A genetic algorithm (GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state.
- Start with *k* randomly generated states (population)
- A state or individual, is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Each state is rated by an objective function, or (in GA terminology) the fitness function/evaluation function (fitness function). Higher values=Better results
- Pairs of individuals are randomly selected for reproduction.
- Produce the next generation of states by Selection, Crossover, and Mutation.

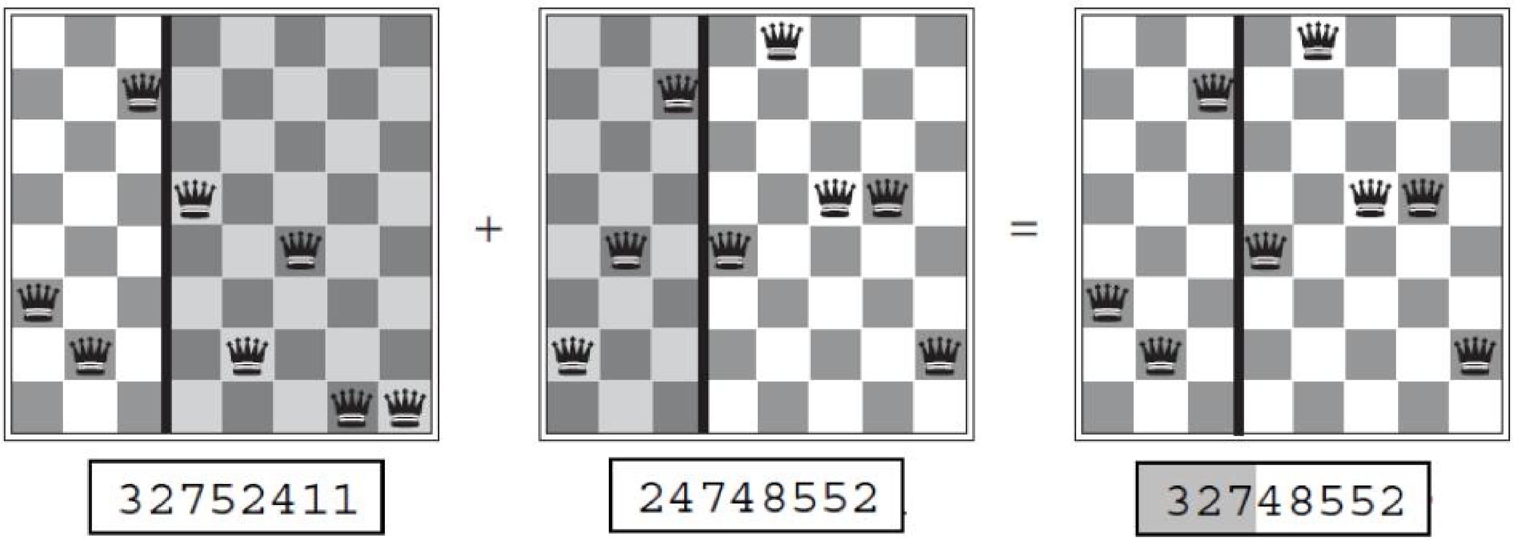
Genetic algorithm: Example

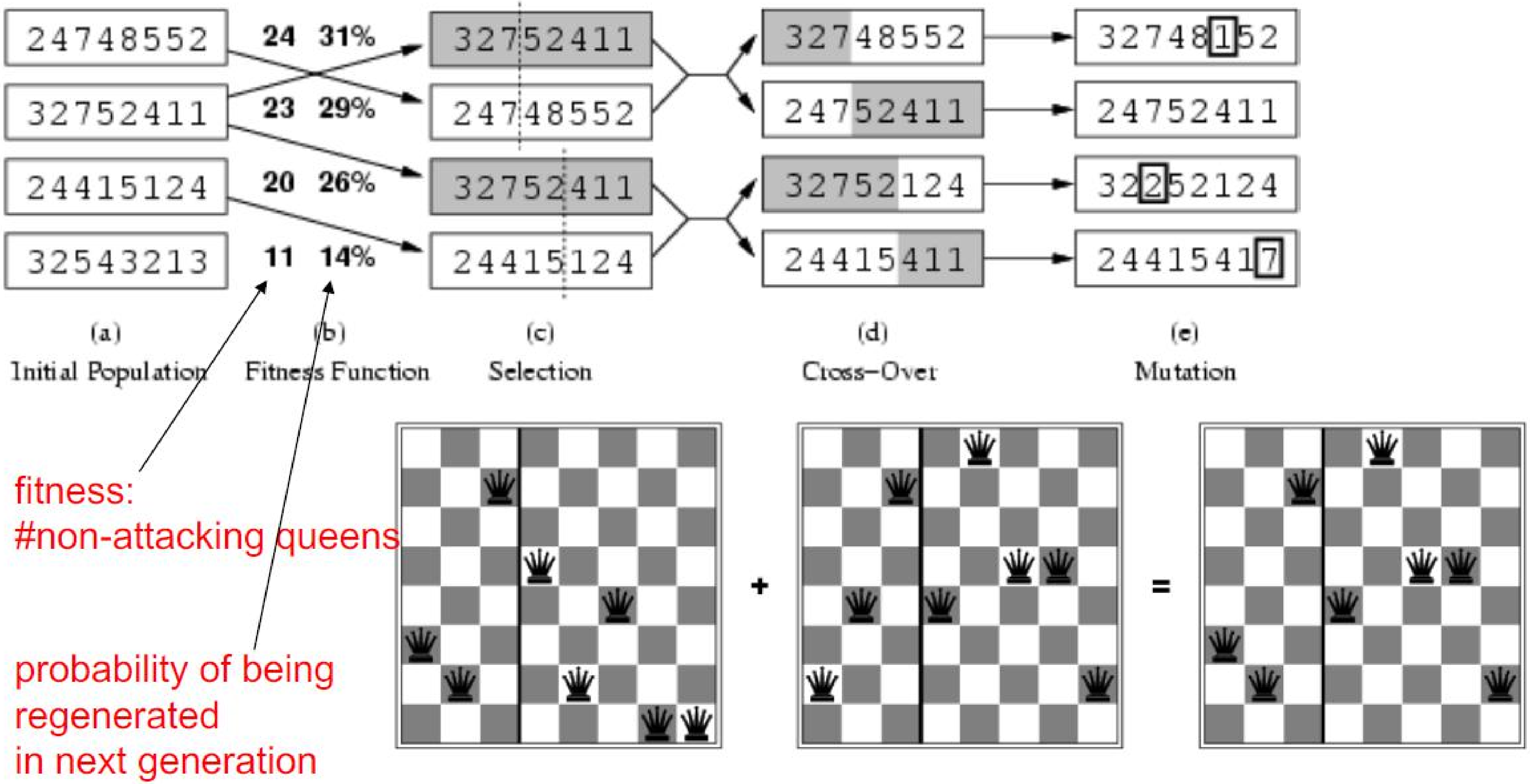
- An 8-queens state must specify the positions of 8 queens, each in a column of 8 squares, and so it requires $8 \times \log_2 8 = 24$ bits.
- Alternatively, the state could be represented as 8 digits, each in the range from 1 to 8.
- A state can be represented using a 8 digit string.
- Each digit in the range from 1 to 8 to indicate the position of the queen in that column.



Genetic algorithm: Example

- Crossover helps if substrings are meaningful components.
- The shaded columns are lost in the crossover step and the unshaded columns are retained.





- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithm: Example

- In above example, the **initial population** has 4 states.
- A **fitness function** should return higher values for better states, so, for the 8-queens problem we use the number of non-attacking pairs of queens, which has a value of 28 for a solution.
  - The values of the four states are 24, 23, 20, and 11.
- The probability of being chosen for reproducing is directly proportional to the fitness score.
- Two pairs are selected at random for reproduction, in accordance with the probabilities.
  - Notice that one individual is selected twice and one not at all.

Genetic algorithm: Example

- The **crossover points** are after third digit in first pair and after fifth digit in second pair.
- The first child of the first pair gets the first three digits from the first parent and the remaining digits from the second parent,
- whereas the second child gets the first three digits from the second parent and the rest from the first parent.
- One digit was **mutated** in the first, third, and fourth offspring.
- In the 8-queensproblem, this corresponds to choosing a queen at random and moving it to a random square in its column.

Genetic algorithm: Example

- Positive points:
- Random exploration can find solutions that local search can’t (via crossover primarily)
- Appealing connection to human evolution
- “neural” networks, and “genetic” algorithms are **metaphors**!
- Negative points:
- Large number of “tunable” parameters
- Difficult to replicate performance from one problem to another
- Lack of good empirical studies comparing to simpler methods
- Useful on some (small?) set of problems but no convincing evidence that GAs are better than hill-