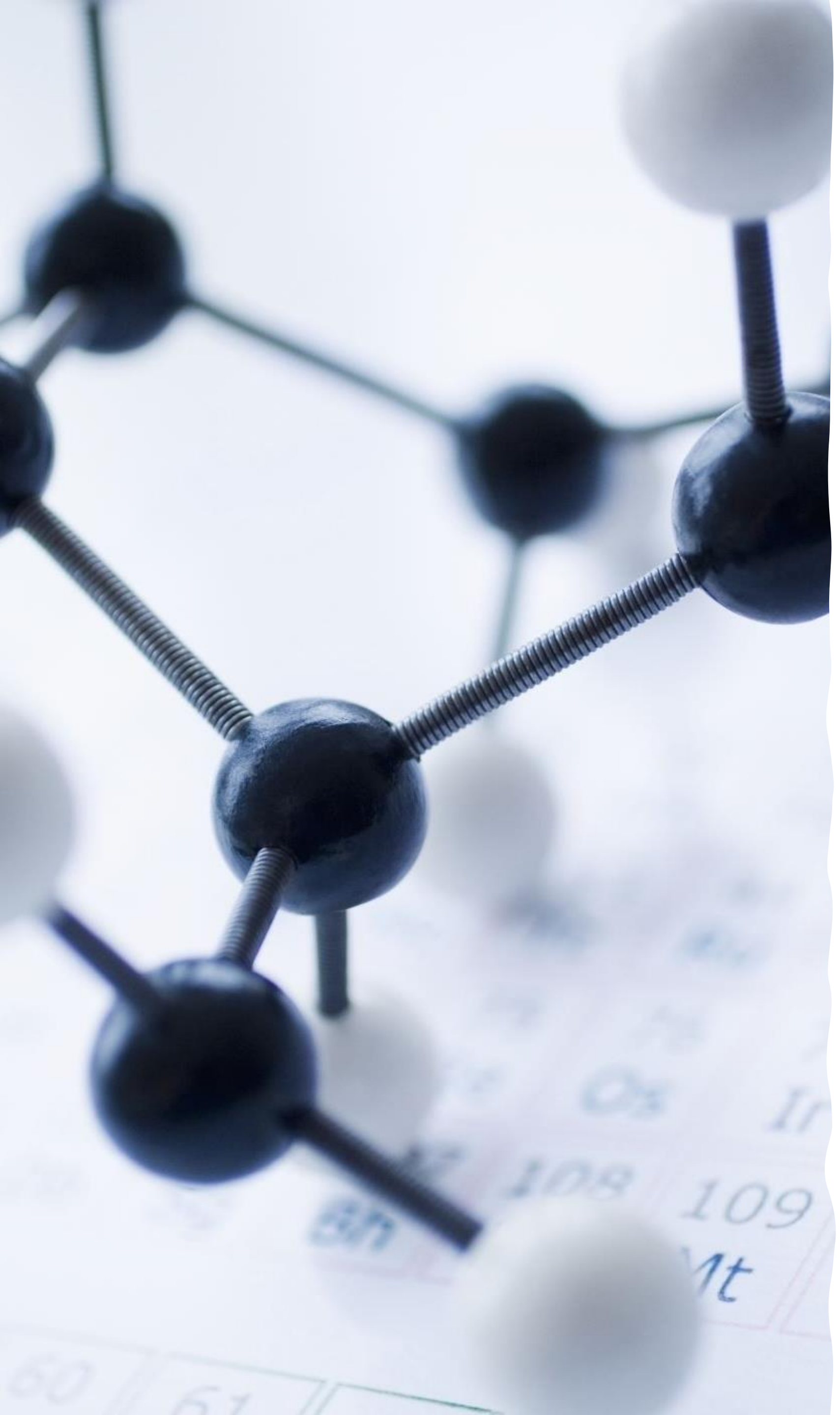
Module 3: Beyond Classical Search
3.1 Local search algorithms and optimization problems

By: Rohini R Rao & Rashmi L Malghan
Dept of Data Science & Computer Applications
January 2024



Local Search Algorithm - Inspired By Evolutionary Biology & Statistic Physics

By: Rohini R Rao & Rashmi L Malghan
Dept of Data Science & Computer Applications
January 2024

Agenda

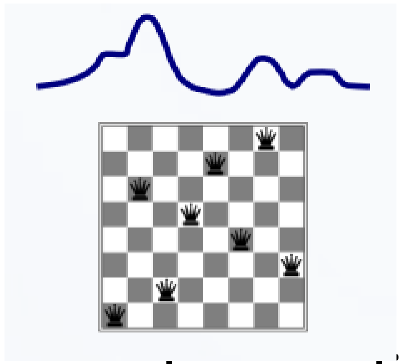
- **Introduction - Hill Climbing**
  - Types of Hill Climbing
  - Example
  - Complexities
  - Applications
- **Simulated Annealing Search**
- **Local Beam Search**
- **Genetic Algorithm**

Iterative Improvement Algorithms

- In the problems we studied so far, the solution is the path.
  - For example, the solution to the 8-puzzle is a series of movements for the "blank tile." The solution to the traveling in Romania problem is a sequence of cities to get to Bucharest.
- In many optimization problems, the path is irrelevant.
  - The goal itself is the solution.
- The state space is set up as a set of "complete" configurations, the optimal configuration is one of them.
- An iterative improvement algorithm keeps a single "current" state and tries to improve it.
- The space complexity is constant

Local Search Algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution.
- Examples:
  - to reduce cost, as in cost functions
  - to reduce conflicts, as in n-queens



- The idea: keep a single "current" state, try to improve it according to an objective function.
- Local search algorithms:
  - Use little memory
  - Find reasonable solutions in large infinite spaces

Local Search Algorithms

- Local search can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions.

- Local search algorithms move from solution to solution in the space of candidate solutions (the search space) until a solution deemed optimal is found or a time bound is elapsed.

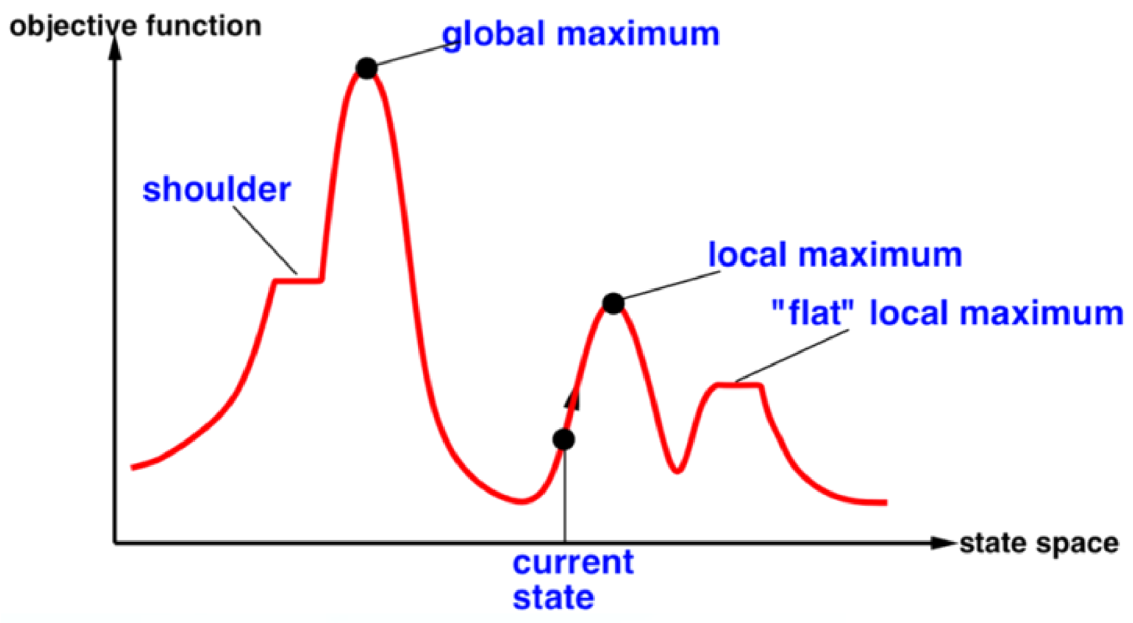
- For example: The travelling salesman problem, in which a solution is a cycle containing all nodes of the graph and the target is to minimize the total length of the cycle. i.e. a solution can be a cycle and the criterion to maximize is a combination of the number of nodes and the length of the cycle.

- A local search algorithm starts from a candidate solution and then iteratively moves to a neighbor solution

Local Search Algorithms

- Terminate on a time bound or if the situation is not improved after number of steps.
- Local search algorithms are typically incomplete algorithms, as the search may stop even if the best solution found by the algorithm is not optimal

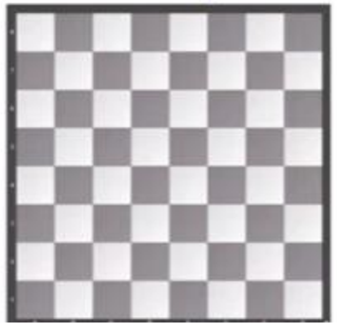
- **Local search** algorithms operate by searching from a start state to neighboring states,
- without keeping track of the paths, nor the set of states that have been reached.
- They are not systematic—
- they might never explore a portion of the search space where a solution actually resides.
- They searches only **the final state**

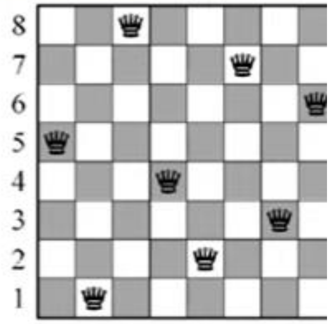


Local Search Algorithm

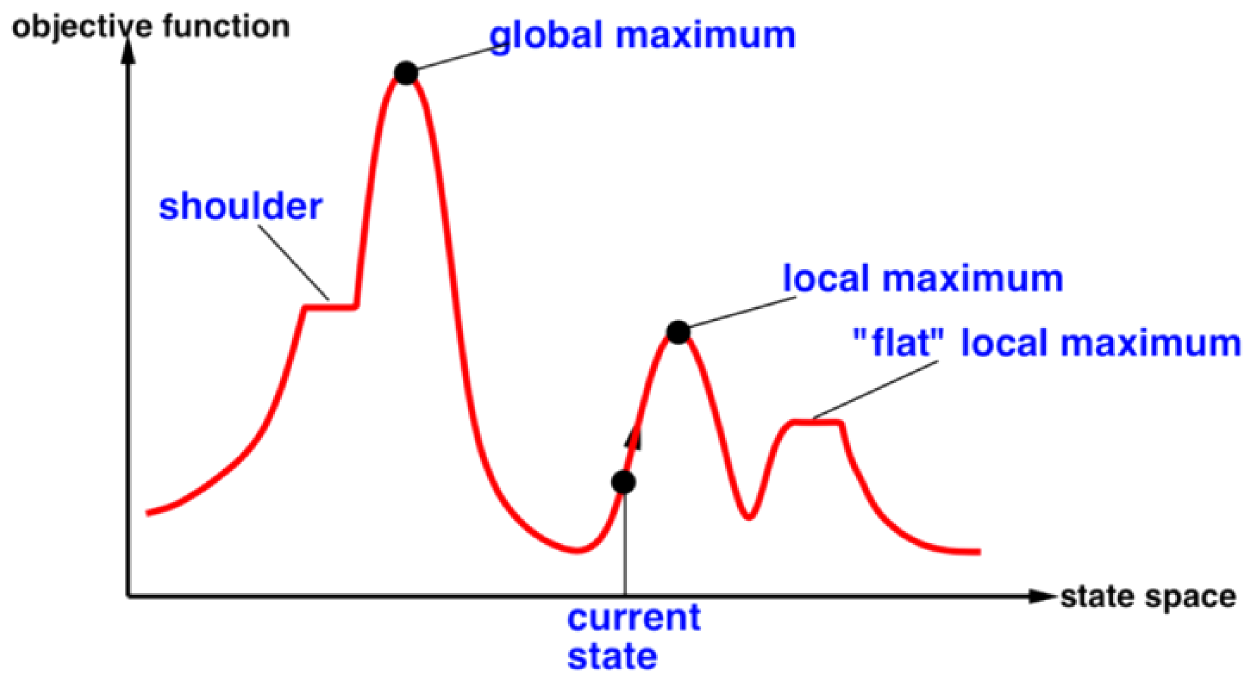
- Its always useful in real time environment.
- Suitable in “continuous environment”

- The Local search algorithm searches only the final state, not the path to get there.
- For example, in the 8-queens problem,
- we care only about finding a valid final configuration of 8 queens (8 queens arranged on chess board, and no queen can attack other queens) and not the path from initial state to final state.
-

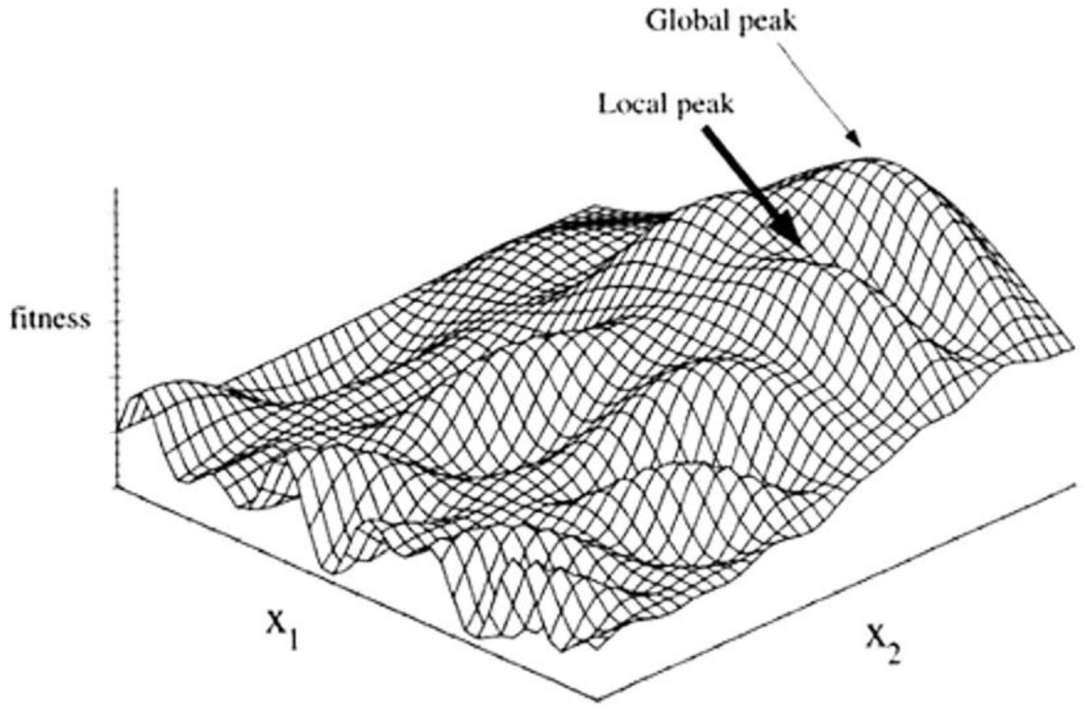




Search Landscape (Two & Three Dimensions)



Two Dimension: A dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow.

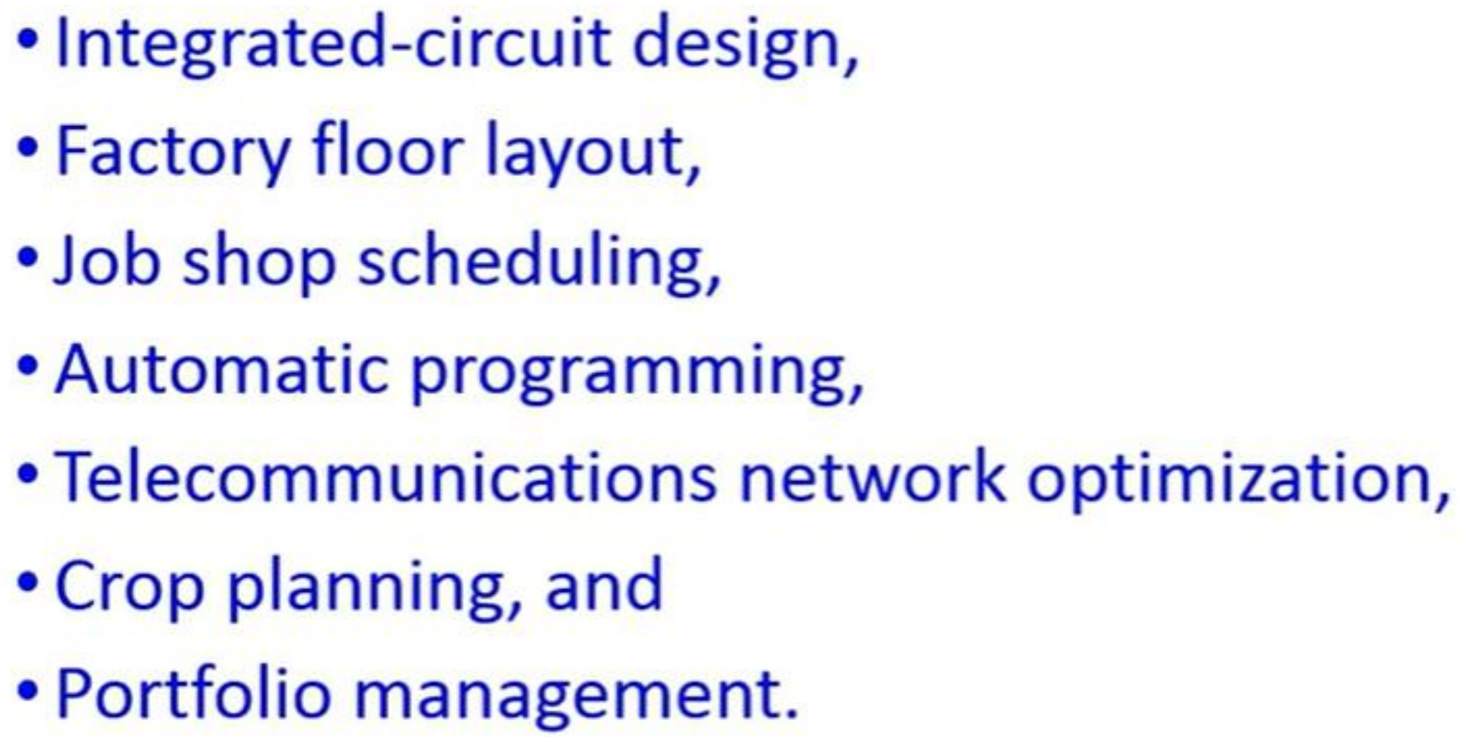


Three Dimension

Summary - Local Search And Optimization

- **Local search**
  - Keep track of single current state – Move only to neighboring states – Ignore paths
- **Advantages:**
  - Use very little memory
  - Can often find reasonable solutions in large or infinite (continuous) state spaces.
- **“Pure optimization” problems**
  - All states have an objective function
  - Goal is to find state with max (or min) objective value
  - Does not quite fit into path-cost/goal-state formulation
  - Local search can do quite well on these problems.

Applications of Local Search Algorithm

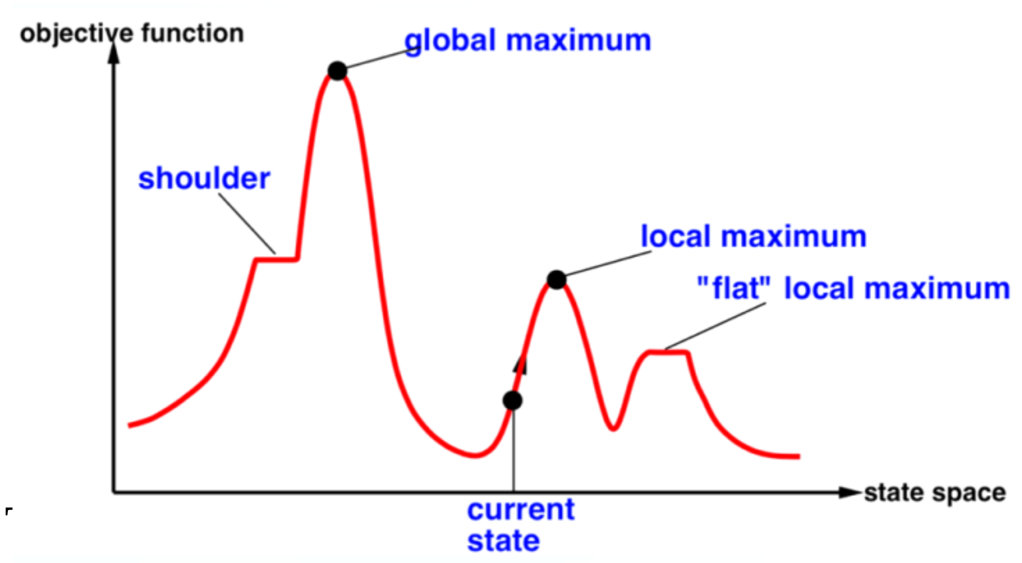


- Integrated-circuit design,
- Factory floor layout,
- Job shop scheduling,
- Automatic programming,
- Telecommunications network optimization,
- Crop planning, and
- Portfolio management.

Local Search Algorithms

- Hill Climbing
- Local Beam Search Algorithm
- Evolutionary Algorithm – Genetic Algorithm
- Simulated Annealing

Hill- Climbing Search (Heuristic Search)



- Continually moves in the direction of increasing value (i.e., uphill).
  of the mountain or best solution to the problem.
- Terminates when it reaches a **“peak”,** no neighbor has a higher value. It keeps track of one current state and on each iteration moves to the neighboring state with highest value (i.e. It heads in the direction of steepest ascent)
- Only records the state and its objective function value.
- Does not look ahead beyond the immediate.
- Sometimes called Greedy Local Search
- **Problem:** Can get stuck in local maxima,
- Its success depends very much on the shape of the state-space land-scape: if there are few local maxima, random-restart hill climbing will find a “good” solution very quickly

Hill- Climbing Search

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

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

The hill-climbing search algorithm, which is the **most basic** local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h.

Example: 8-queens

Each number indicates h if we move a queen in its corresponding column

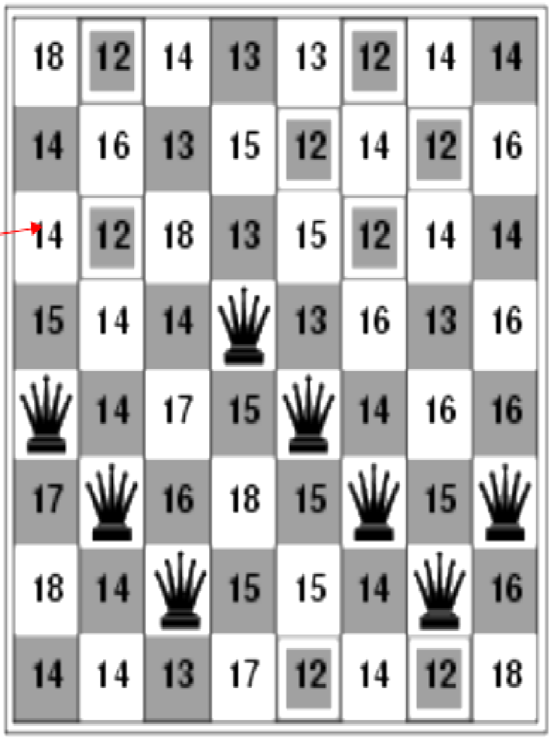


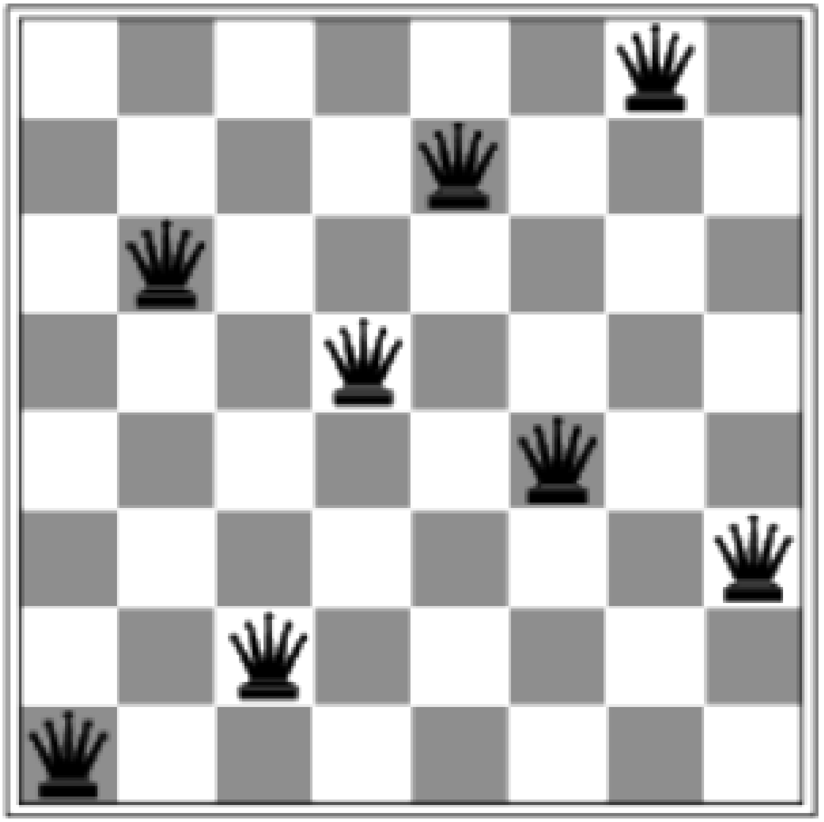
Figure 4.3 (a) An 8-queens state with heuristic cost estimate h = 17, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked.

h = number of pairs of queens that are attacking each other, either directly or indirectly (*h* = *17* for the above state)

- To illustrate hill climbing, we will use the 8-queens problem. Local search algorithms typically use a complete-state formulation, where each state has 8 queens on the board, one per column.
- The successors of a state are all possible states generated by moving a single queen to another square in the **same column** (so each state has 8 × 7 = 56 successors).
- The heuristic cost function **"h"** is the number of pairs of queens that are attacking each other, either directly or indirectly.
- The global minimum of this function is zero, which occurs only at perfect solutions.
- Figure 4.3(a) shows a state with h = 17. The figure also shows the values of all its successors, with the best successors having h = 12. Hill-climbing algorithms typically choose randomly among the set of best successors if there is more than one.

Example: n-queens

- **Figure 4.3 (b)** A local minimum in the 8-queens state space;
- The state has h = 1 but every successor has a higher cost

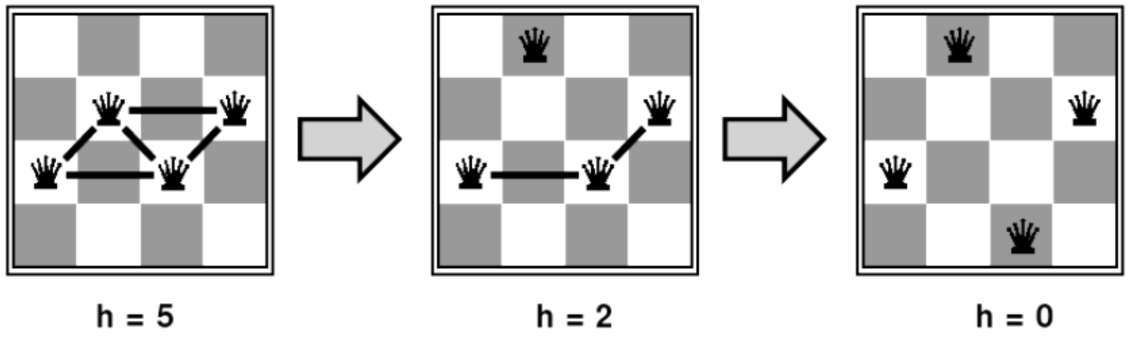


A local minimum with *h* = *1*

Example: n-queens

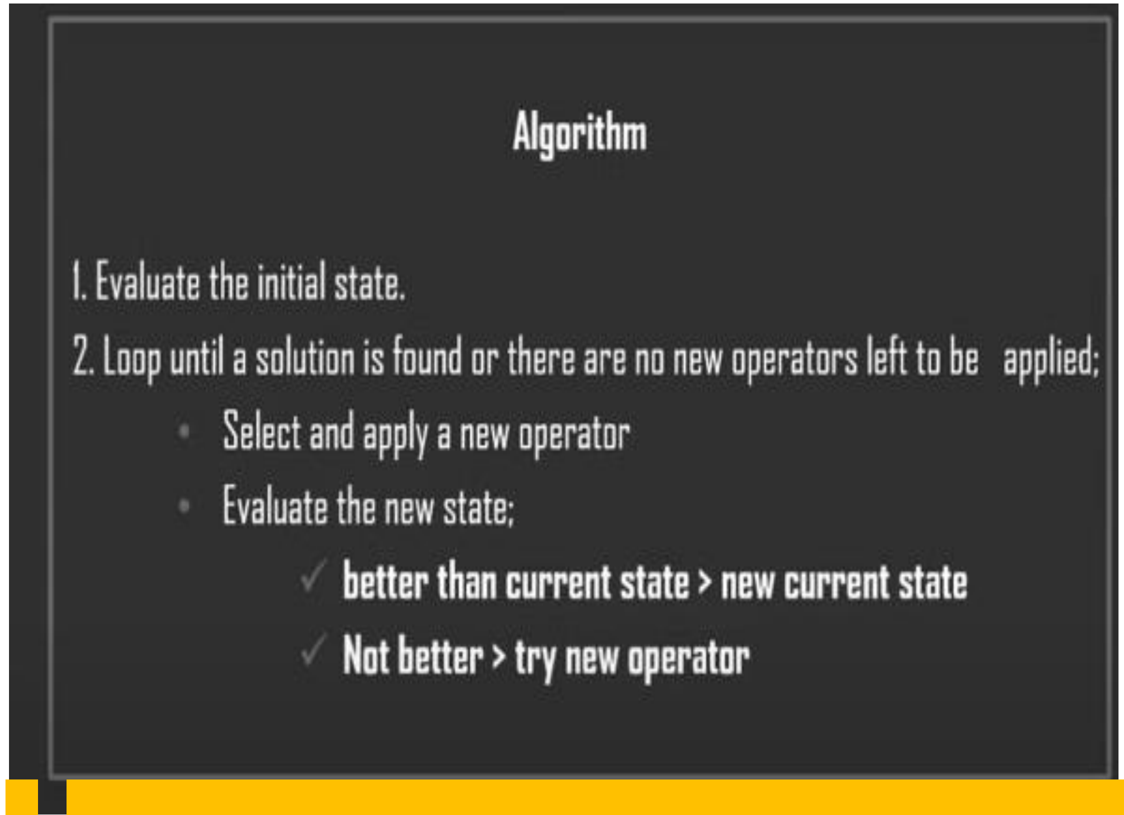
Put *n* queens on an *n*×*n* board with no two queens on the same row, column, or diagonal.

Move a queen to reduce number of conflicts.



- Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next.
- Although greed is considered one of the seven deadly sins, it turns out that greedy algorithms often perform quite well.
- Hill climbing often makes rapid progress toward a solution because it is usually quite easy to improve a bad state.
- For example, from the state in Figure 4.3(a), it takes just five steps to reach the state in Figure 4.3(b), which has h = 1 and is very nearly a solution.

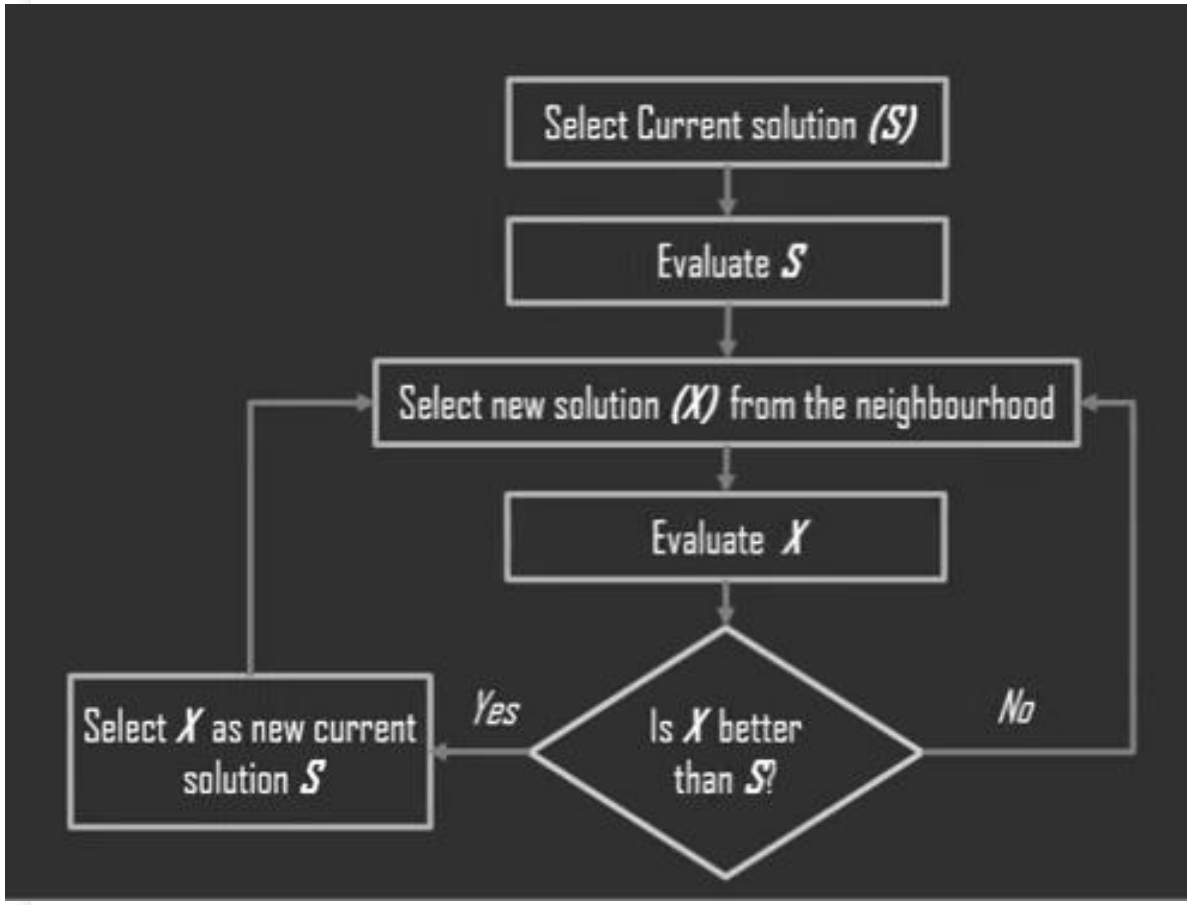
Flowchart: Hill Climbing



Algorithm

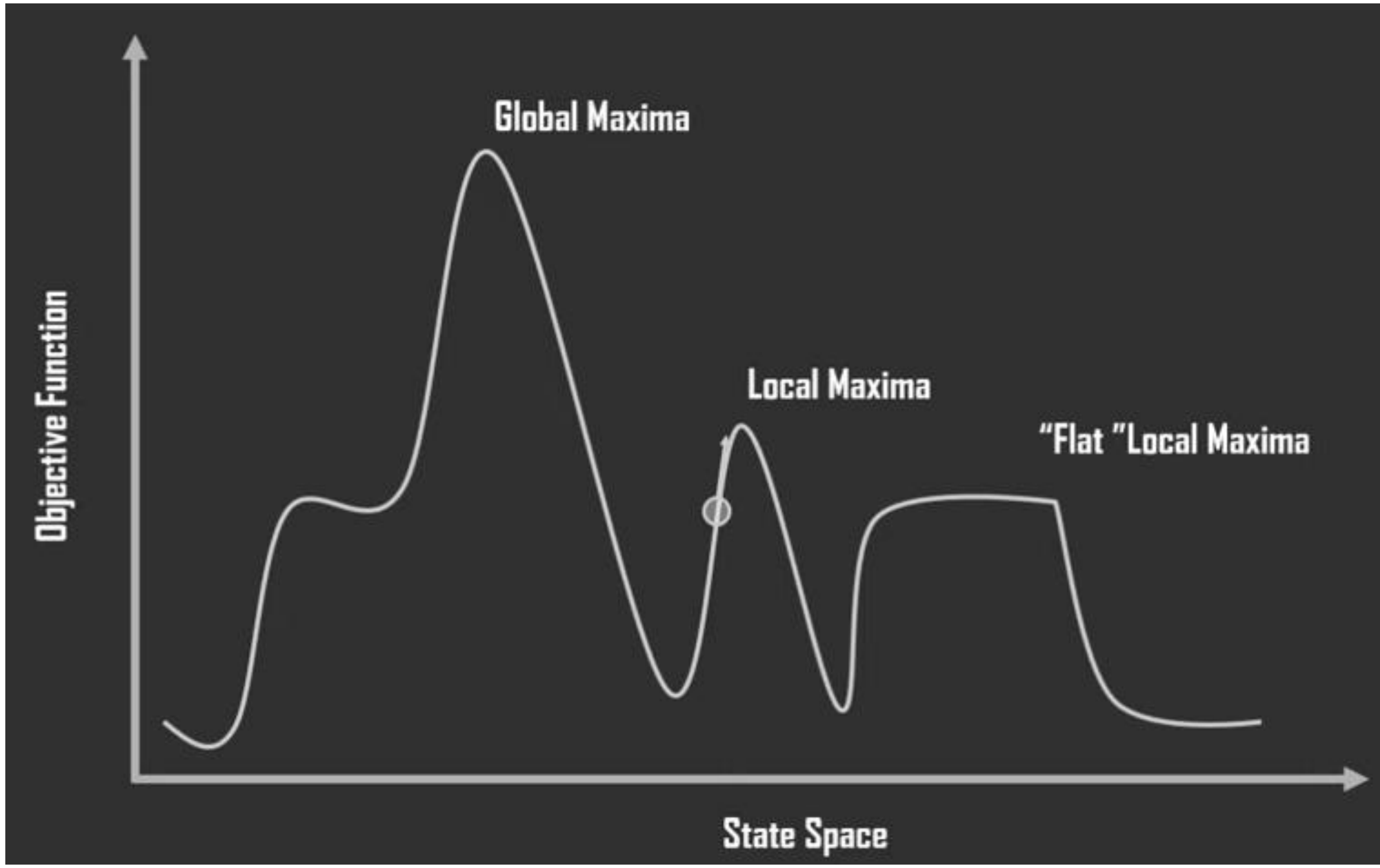
1. Evaluate the initial state.
2. Loop until a solution is found or there are no new operators left to be applied;
   - Select and apply a new operator
   - Evaluate the new state;
     - ✓ better than current state > new current state
     - ✓ Not better > try new operator

- It is variant of "Generate & Test Algorithm"
- With addition of – Greedy Approach

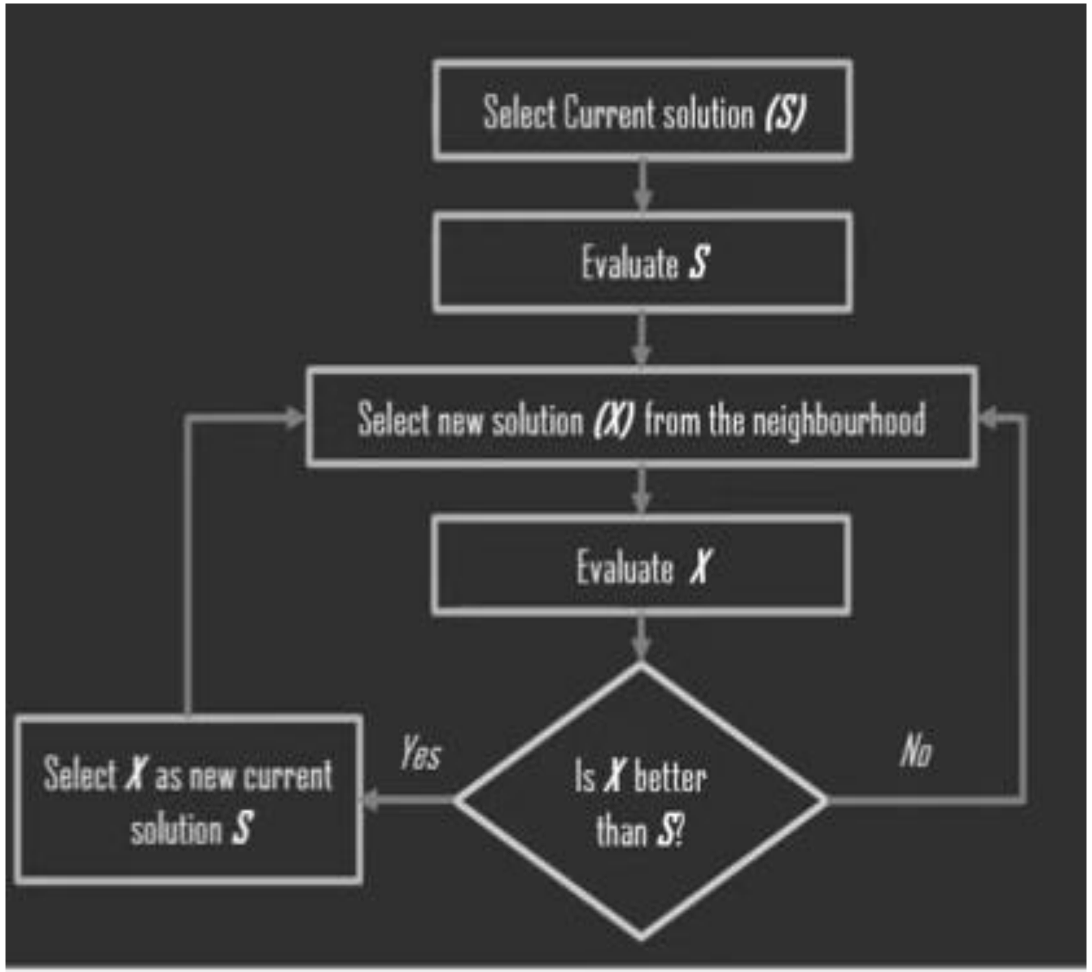

Select Current solution (S)
Evaluate S
Select new solution (X) from the neighbourhood
Evaluate X
Is X better than S?
Yes
No
Select X as new current solution S

Search Space - Diagram

- Search Space Diagram is having so many regions.
- It is graphical representation of set of states that our search algorithm can reach V/S the value of objective function.
- The objective function - wish to maximize or minimize.
- X- Axis = State/Configuration of algorithm
- Y-Axis = Values of objective function corresponding to specific state
- Solution : The state which has maximum objective function value/Global maximum

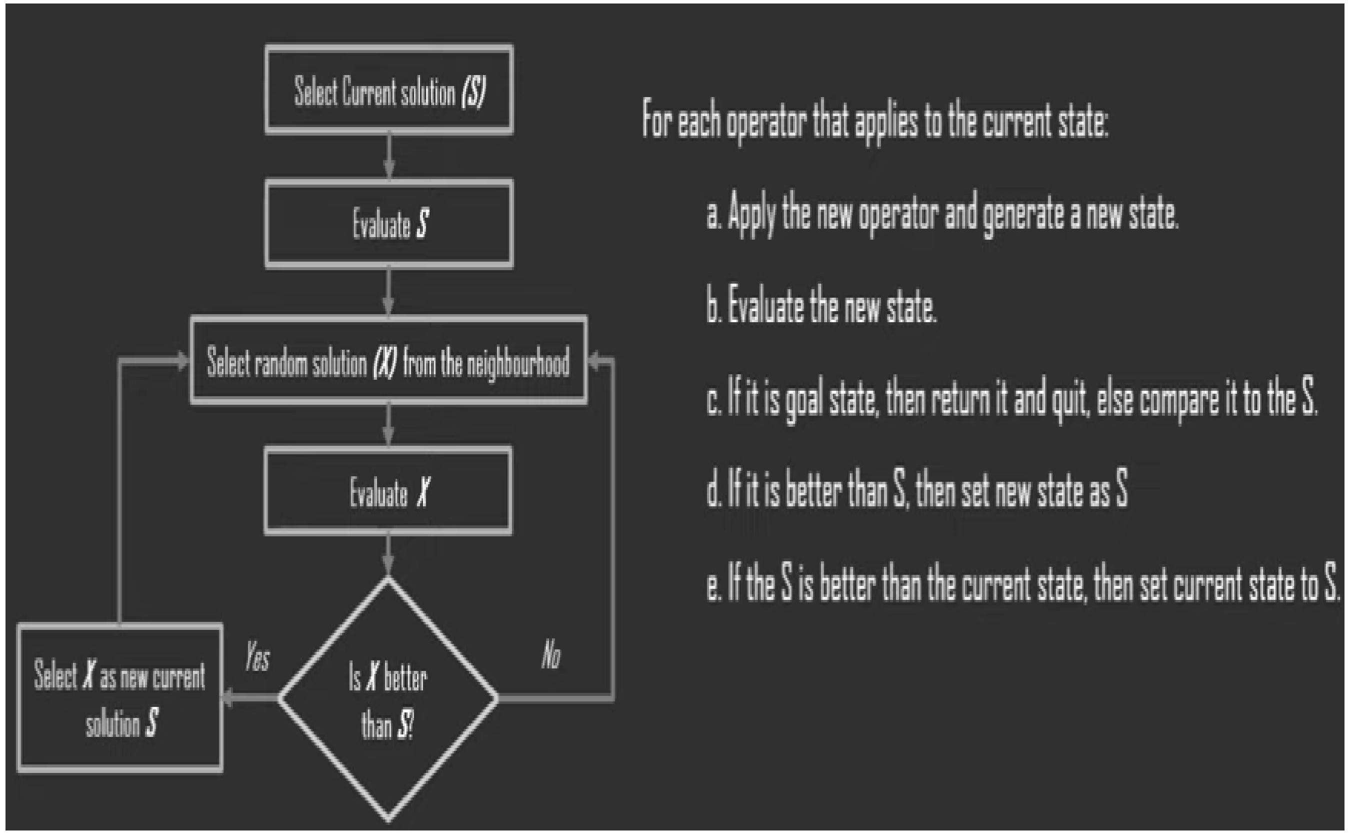


Types of Hill Climbing: 1)Simple Hill Climbing


Select Current solution (S)
Evaluate S
Select new solution (X) from the neighbourhood
Evaluate X
Is X better than S?
Yes
No
Select X as new current solution S

- It is Simplest way to implement hill climbing.
- It only evaluates the neighbor node state at a time.
- Select the first one which optimizes current cost & set it as a current state.
- It checks only 1 successor state, if successor state is better then current state then it moves to next state else remains in same state.
- It is less time consuming
- It gives less optimum solution.
- Solution is not guaranteed

Types of Hill Climbing: 2)Steepest Ascent Hill Climbing


Select Current solution (S)
Evaluate S
Select random solution (X) from the neighbourhood
Evaluate X
Is X better than S?
Yes
No
Select X as new current solution S

For each operator that applies to the current state:

a. Apply the new operator and generate a new state.

b. Evaluate the new state.

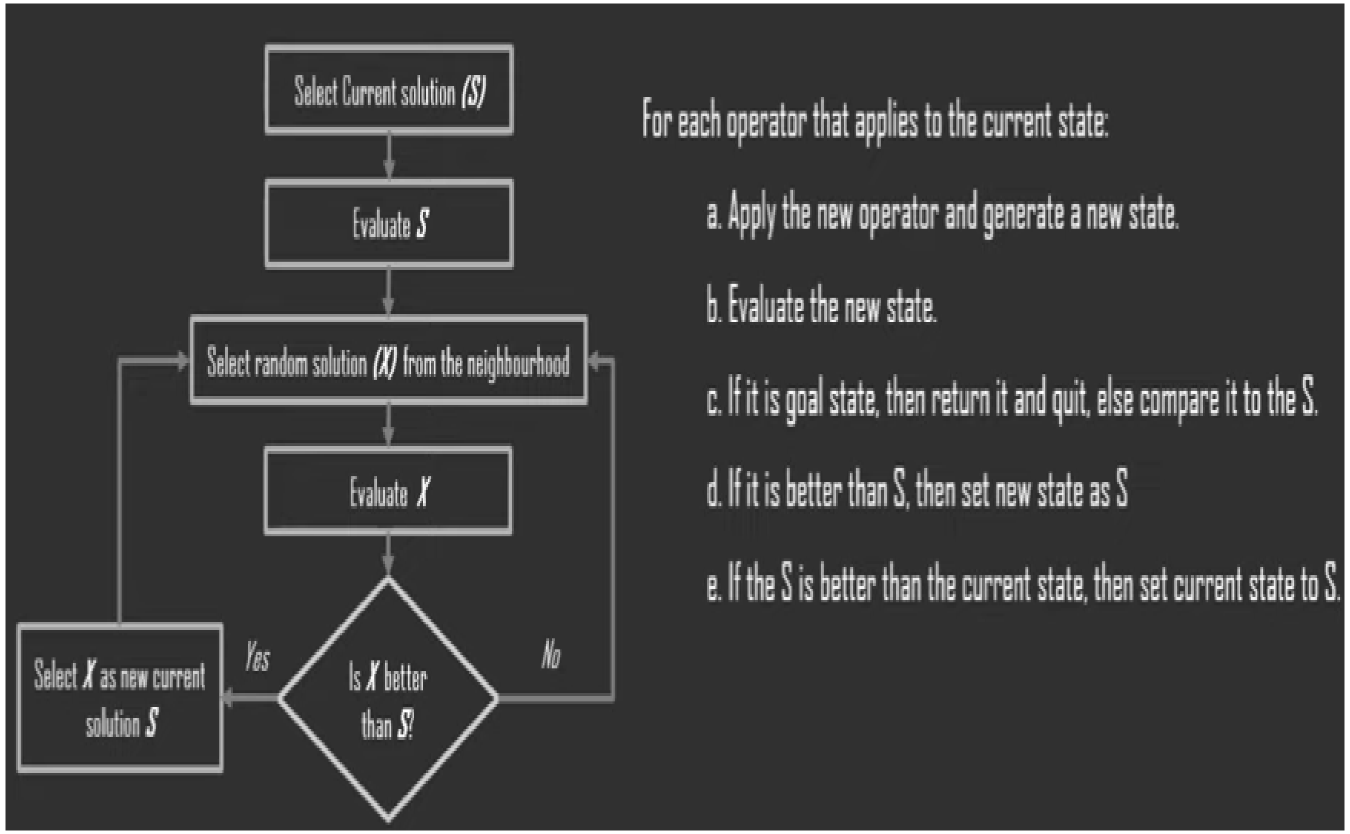
c. If it is goal state, then return it and quit, else compare it to the S.

d. If it is better than S, then set new state as S

e. If the S is better than the current state, then set current state to S.

- It is “variation” of simple hill climbing algorithm.
- It examines all the neighboring nodes of current state.
- Selects 1 neighbor node which is closest to the goal state.
- It consumes more time as it searches for “multiple neighbors”
- It gives less optimum solution.
- Solution is not guaranteed

Types of Hill Climbing: 2) Stochastic Hill Climbing


Select Current solution (S)
Evaluate S
Select random solution (X) from the neighbourhood
Evaluate X
Is X better than S?
Yes
No
Select X as new current solution S

For each operator that applies to the current state:

a. Apply the new operator and generate a new state.

b. Evaluate the new state.

c. If it is goal state, then return it and quit, else compare it to the S.

d. If it is better than S, then set new state as S

e. If the S is better than the current state, then set current state to S.

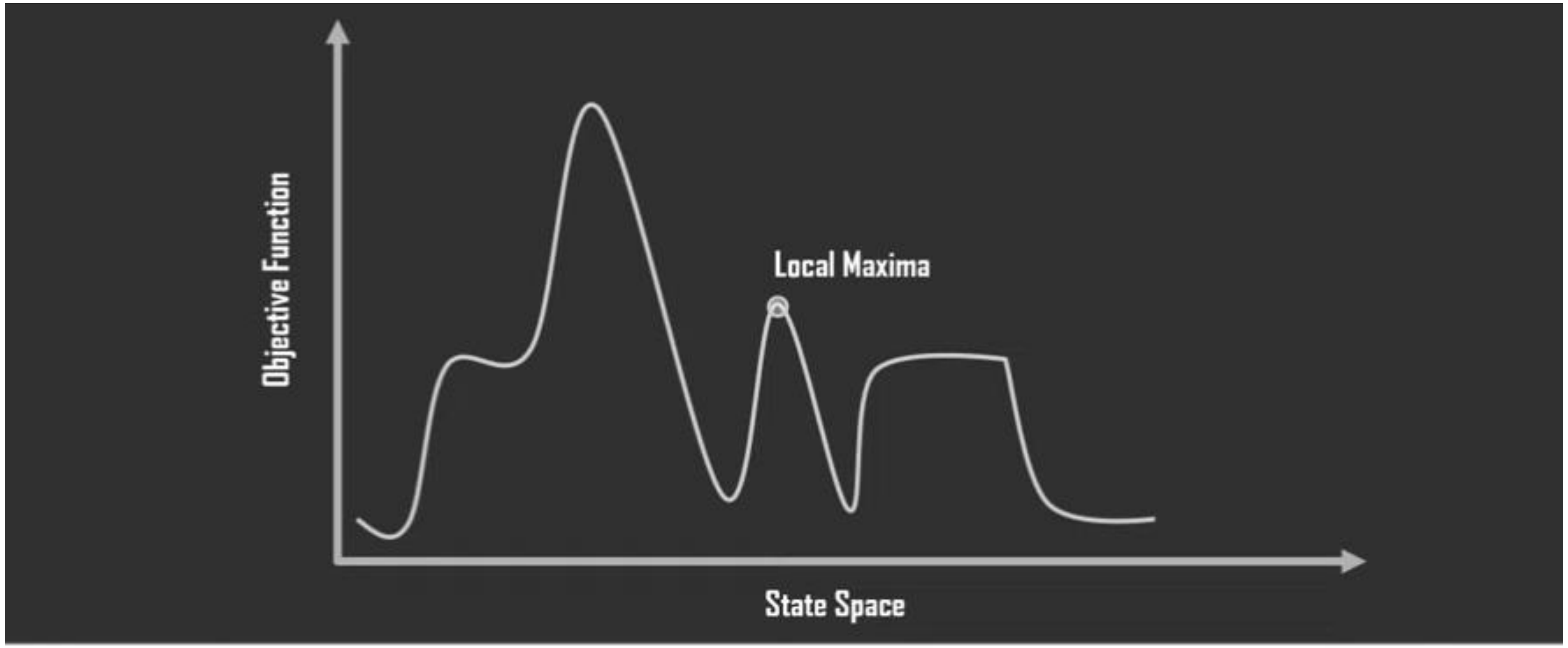
- It doesn’t examine all its neighbor before moving.
- It search algorithm - selects one neighbor node – at random.
- Based on that it decides to choose it as current state or examine another state.
- It consumes more time
- Better Solution is guaranteed

Understanding- Hill Climbing

- It gets rids of “Population” & “Crossover”
- It focuses on ease of implementation
- It has faster iterations compared to traditional genetic algorithm
- It is less thorough.

Complexities: 1) Local Maxima

Local maxima: a local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go

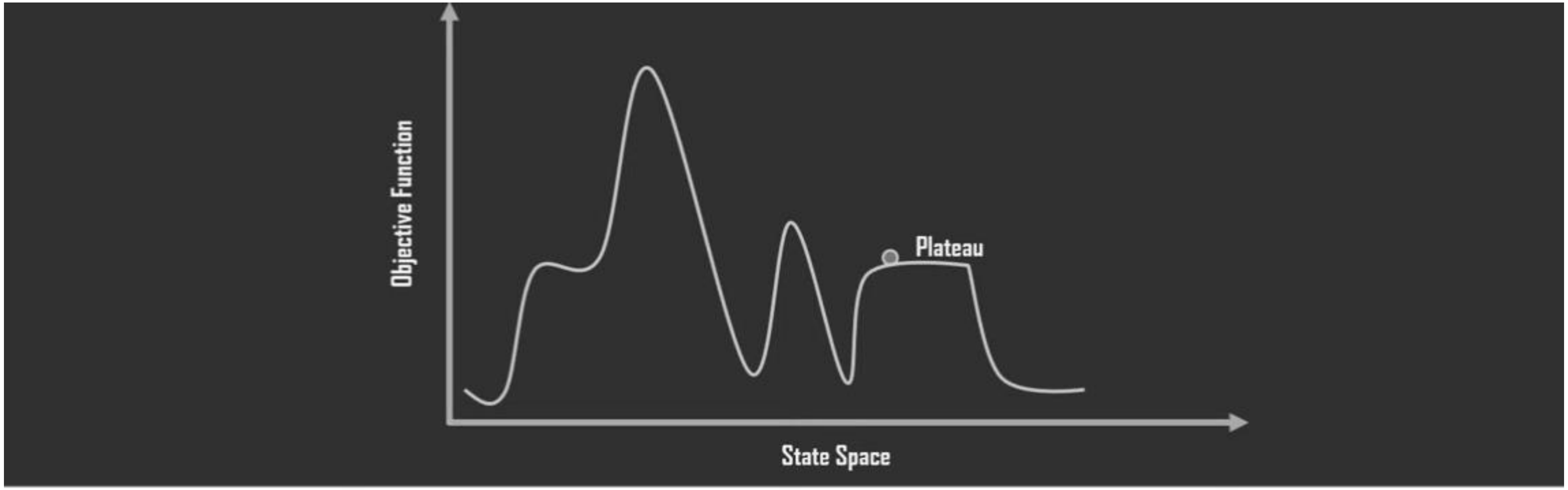


At a local maxima, the process will end even though a better solution may exist.

Utilize backtracking technique to deal with this situation.

2)Reaching Plateau Region

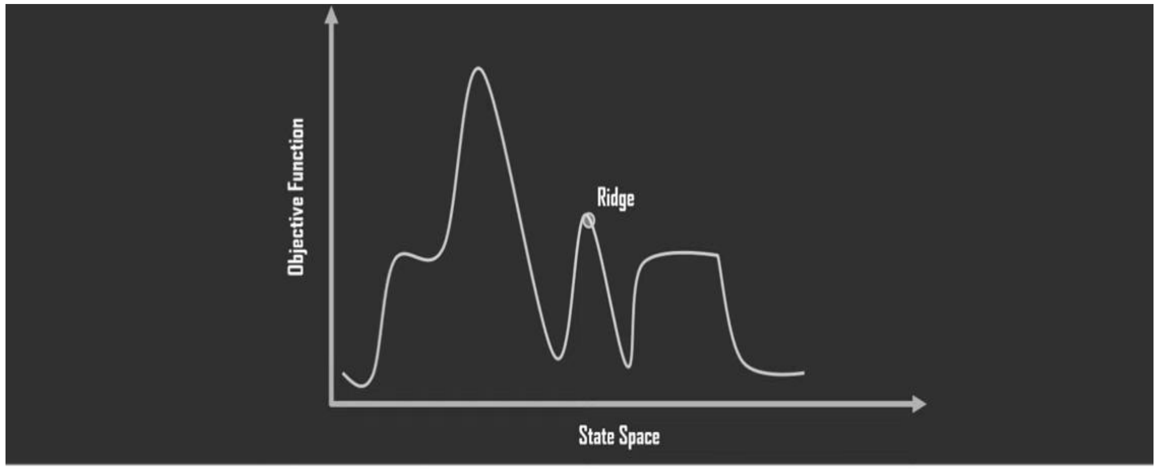
Plateaux: A plateau is a flat area of the state-space landscape. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible.



On plateau all neighbors have same value . Hence, it is not possible to select the best direction.

So, Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region

3) Ridge

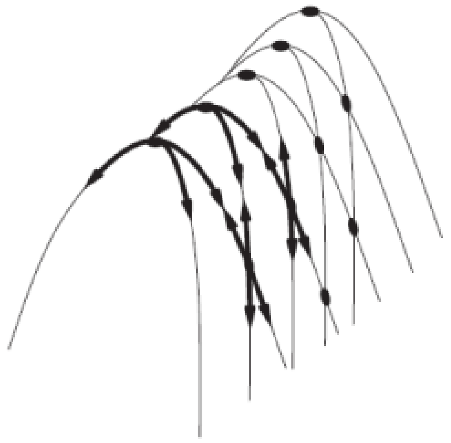


A ridge can look like a peak; hence, algorithm can end.

In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

Difficulties with ridges

The "ridge" creates a sequence of local maxima that are not directly connected to each other. From each local maximum, all the available actions point downhill.

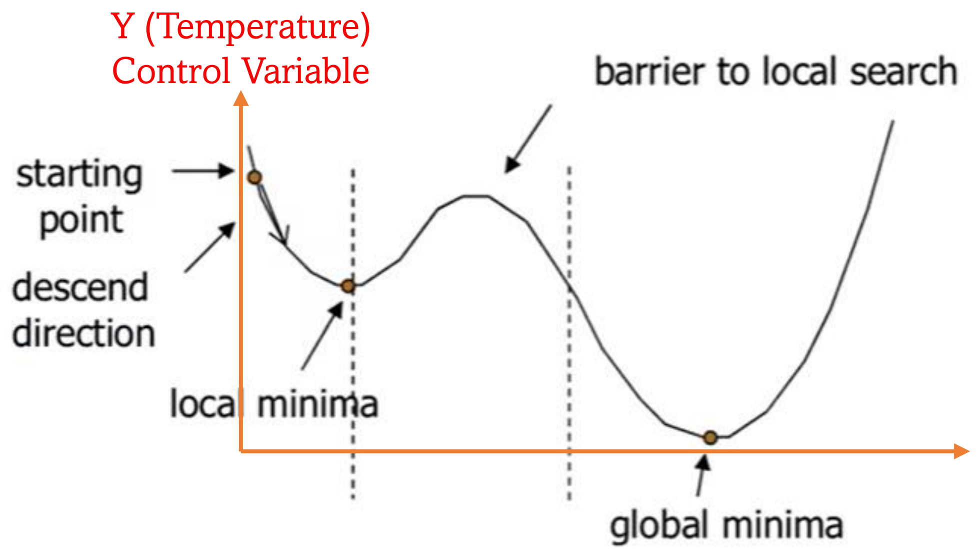


Applications:

Hill Climbing technique can be used to solve many problems, where the current state allows for an accurate evaluation functions, inductive learning methods, robotic coordination problems, etc.

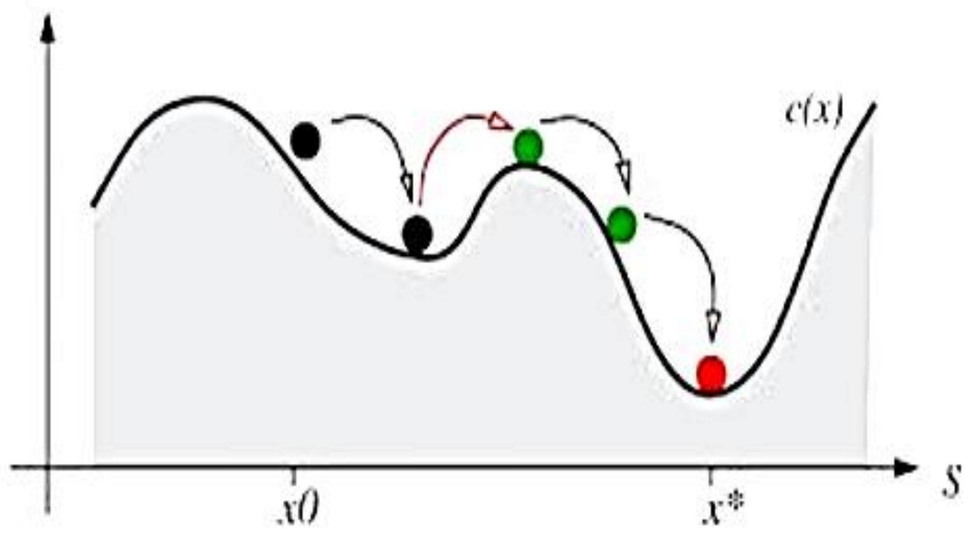
Simulated Annealing

- **Simulated Annealing** is a stochastic global search optimization algorithm and it is modified version of stochastic hill climbing.
- This algorithm appropriate for nonlinear objective functions where other local search algorithms do not operate well.
- **The simulated-annealing solution is to start by shaking hard (i.e., at a high temperature) and**
- **then gradually reduce the intensity of the shaking (i.e., lower the temperature).**
- Simulated Annealing (SA) is very useful for situations where there are a lot of local minima.



Simulated Annealing Search

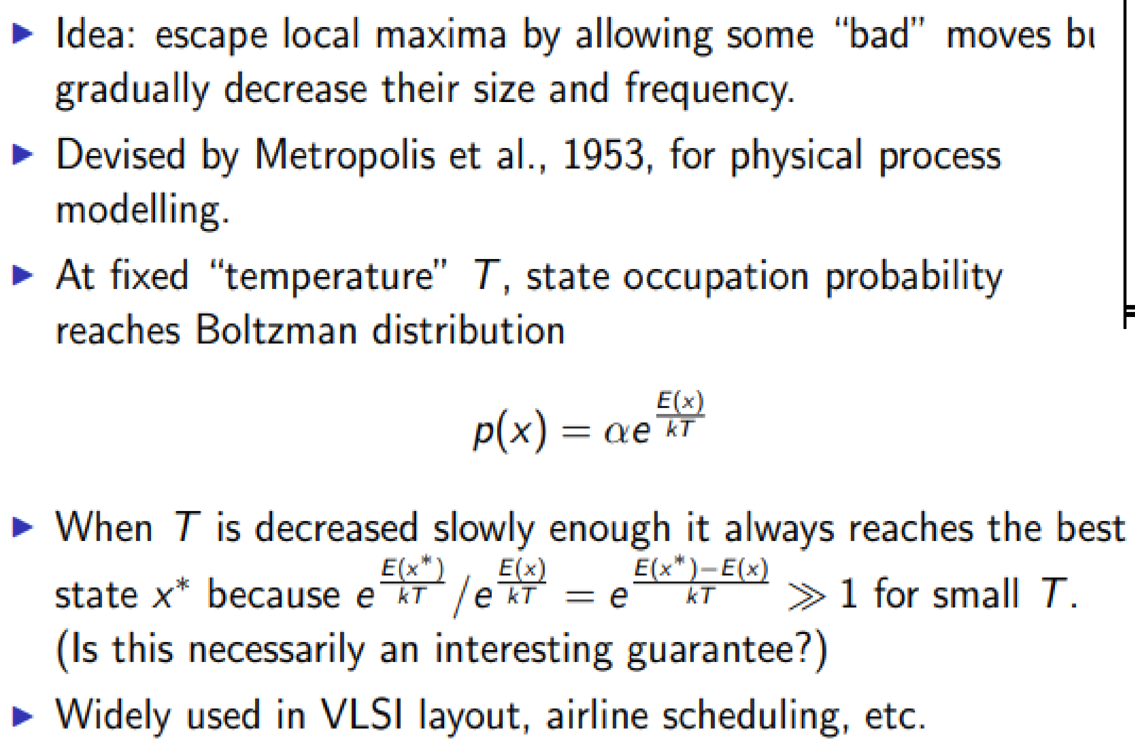
- To avoid being stuck in a local maxima, it tries randomly (using a probability function) to move to another state, if this new state is better it moves into it, otherwise try another move… and so on.
- Terminates when finding an acceptably good solution in a fixed amount of time, rather than the best possible solution.
- Locating a good approximation to the global minimum of a given function in a large search space.
- Widely used in VLSI layout, airline scheduling, etc.



Properties of Simulated Annealing Search

- The problem with this approach is that the neighbors of a state are not guaranteed to contain any of the existing better solutions which means that failure to find a better solution among them does not guarantee that no better solution exists.
- It will not get stuck to a local optimum.
- If it runs for an infinite amount of time, the global optimum will be found

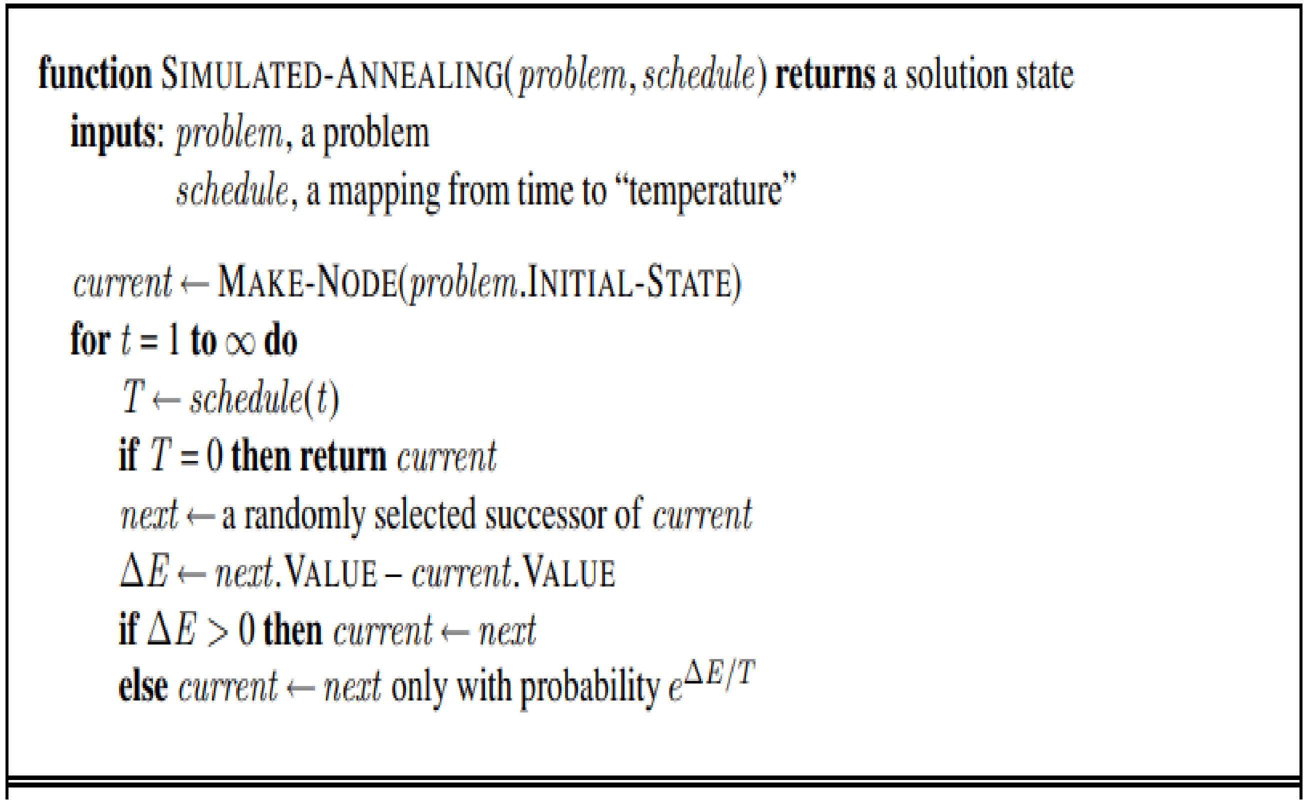
Simulated Annealing



- Idea: escape local maxima by allowing some "bad" moves bu gradually decrease their size and frequency.
- Devised by Metropolis et al., 1953, for physical process modelling.
- At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

- When T is decreased slowly enough it always reaches the best state x^* because $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T. (Is this necessarily an interesting guarantee?)
- Widely used in VLSI layout, airline scheduling, etc.



```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for $t$ = 1 to $\infty$ do
     $T$ ← schedule($t$)
     if $T$ = 0 then return current
     next ← a randomly selected successor of current
     $\Delta E$ ← next.VALUE – current.VALUE
     if $\Delta E > 0$ then current ← next
     else current ← next only with probability $e^{\Delta E/T}$
```

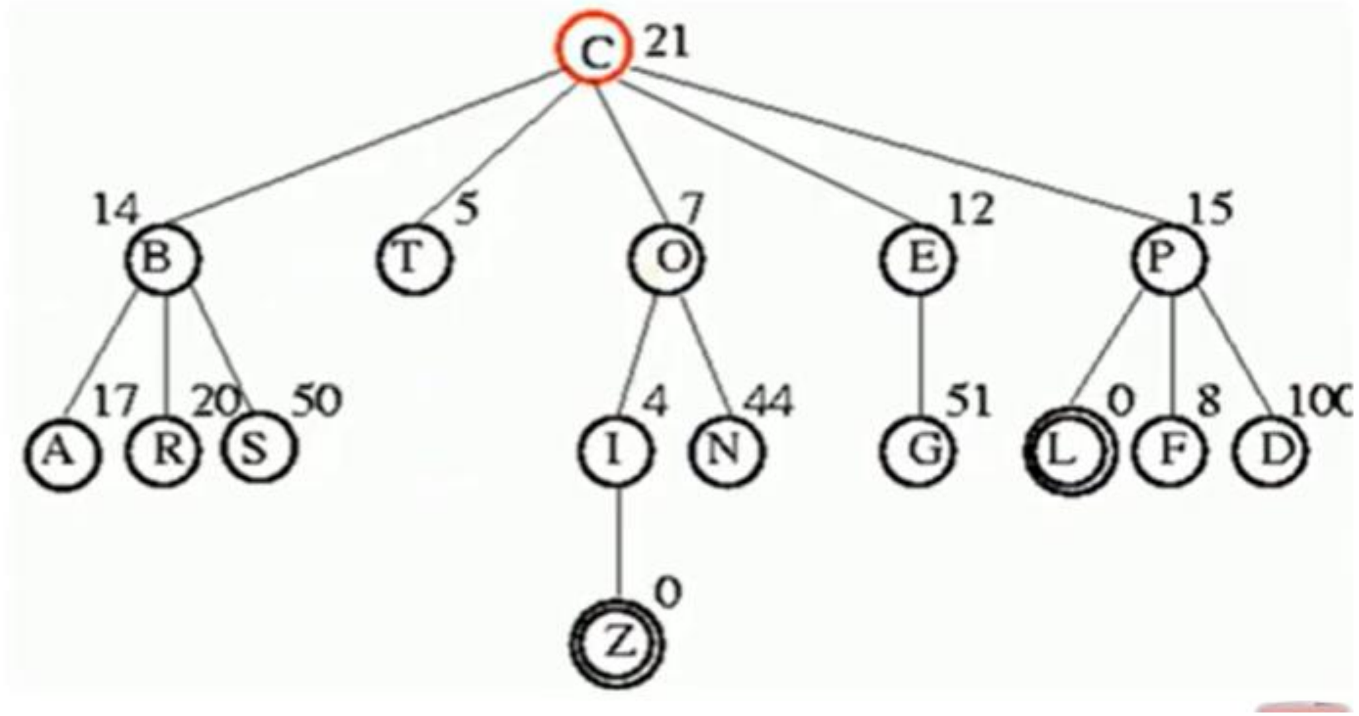
- **Figure 4.5** The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The schedule input determines the value of the temperature T as a function of time

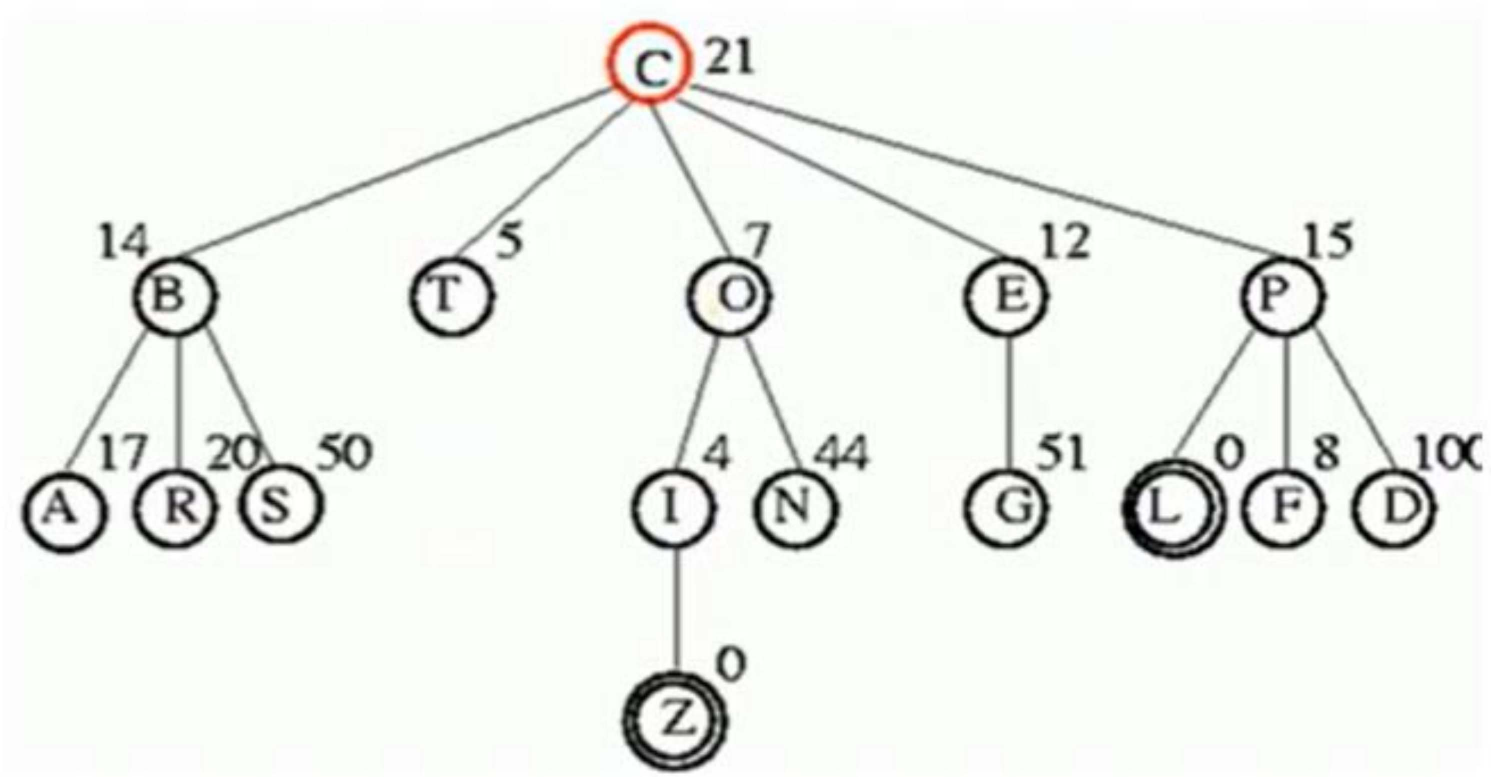
Local Beam Search

- Idea: keep k states instead of 1; choose top k of all their successors
- Not the same as k 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.
- To improve: choose k successors randomly, biased towards good ones.
- Observe the close analogy to natural selection

Example :

- Start State: C
- Goal State: Z and L
- n = 2 (beam size)
- Iteration 1
- Open List = {c}

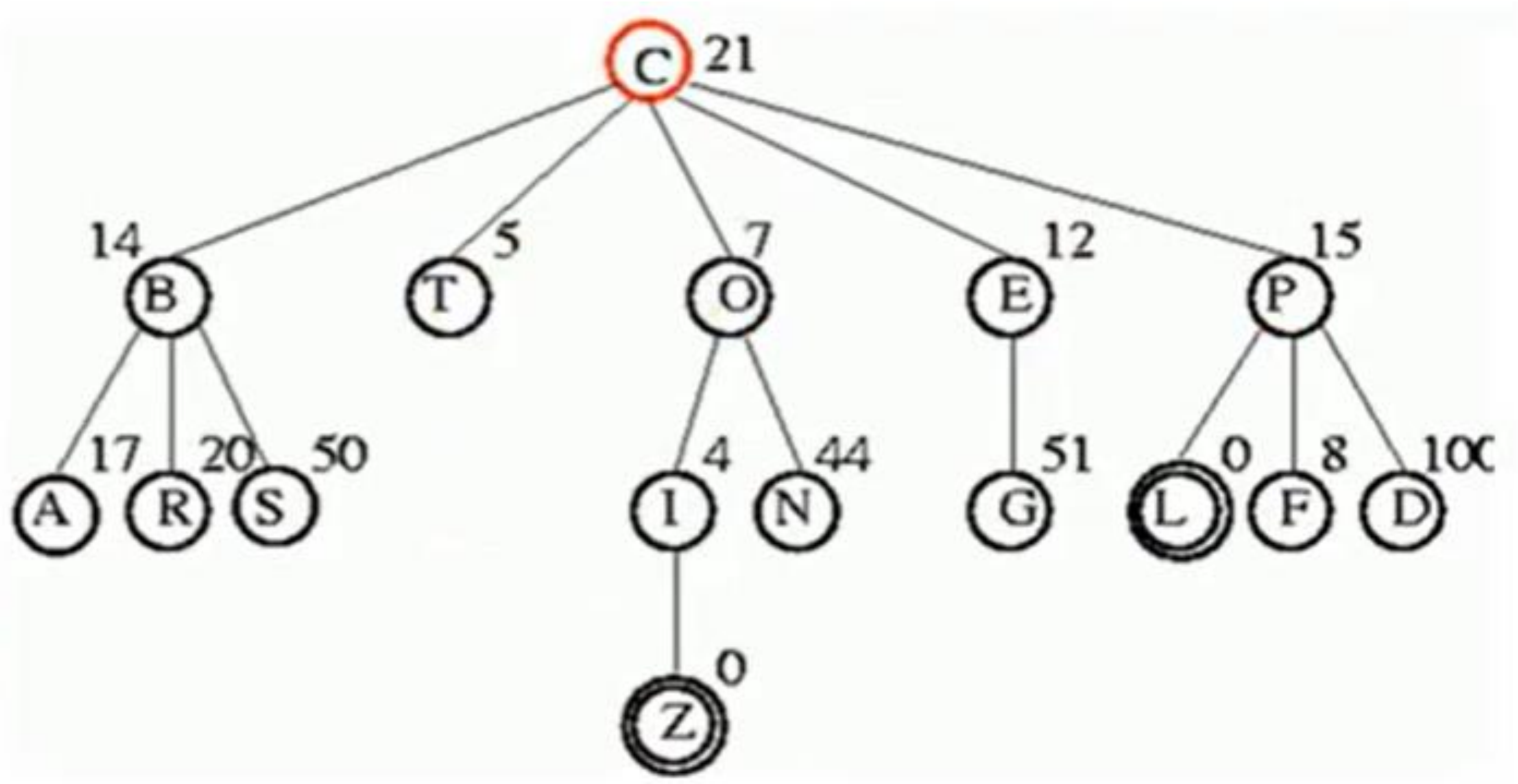




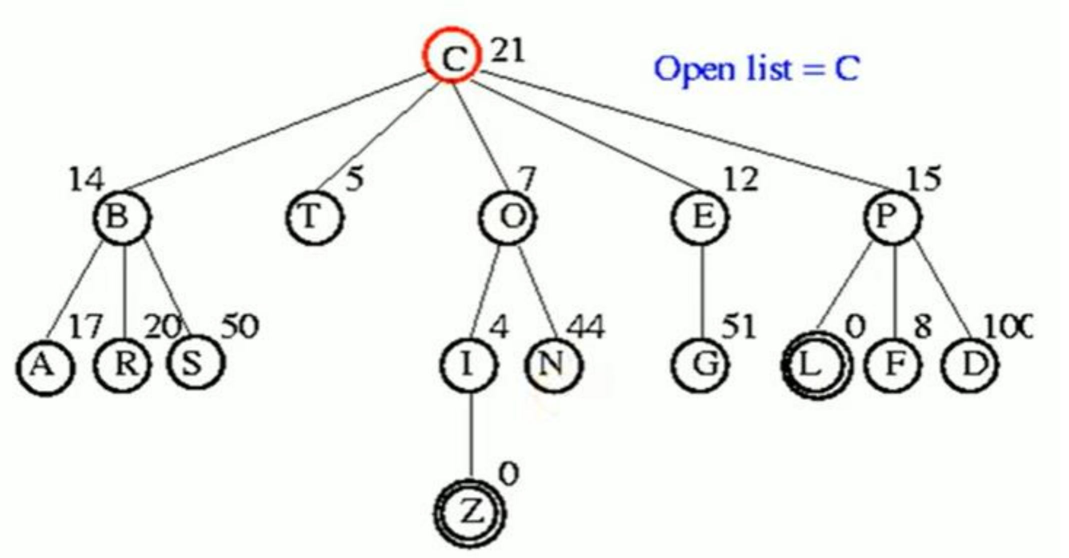
- Iteration 2
- Find successor of C
- = B T O E P
- Remove C from list, now
- Open list = {T,O}
- Iteration 3
- T has no successors, remove T from open list
- Find successor of O, replace O with I and N in open list, then
- Open list = {I, N}

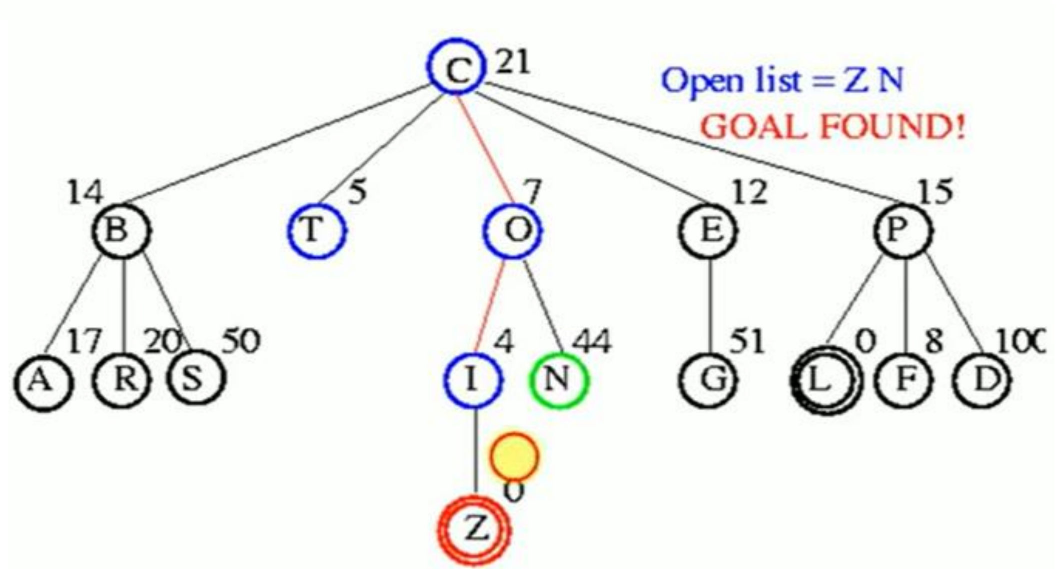
- Open list = {I, N}

- Iteration 4
- Find successor of I
- Z is Goal



Path : C – O – I – Z





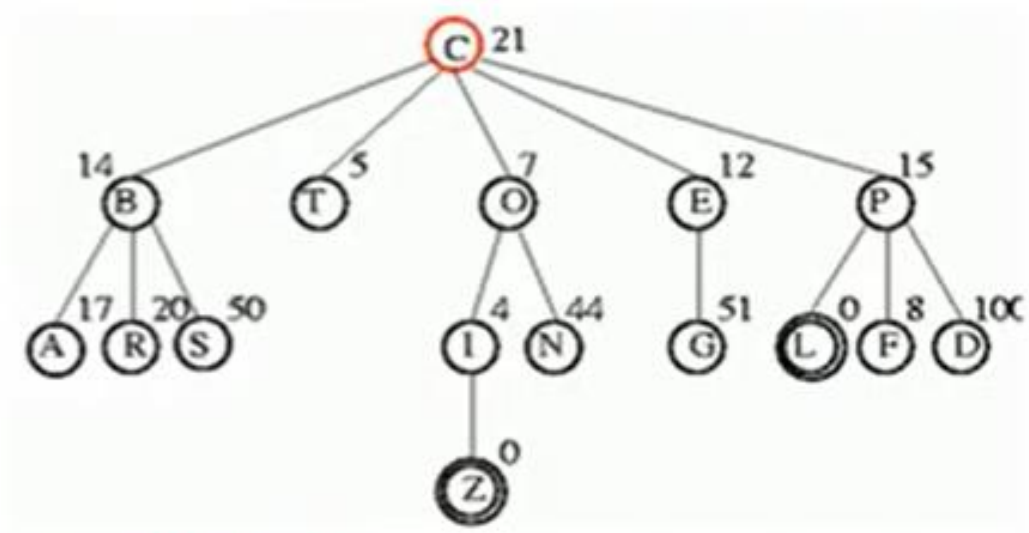
Steps:

- Consider lowest successor (having low estimated value).
- Here T Is considered and Explored.
- Since T cant be explored further (T need to be removed from open lost)
- Open List
- C
- T,O (remove T)
- I N (Remove O)
- Z, N (Explore I)
- Keep only 2 nodes in open list (means value of n is 2)

- Beam search is "intermediate Algorithm" (Between Hill climbing & Best First Search)
- In case of Hill climbing – It wont be able to reach goal state as per this problem. Because it will reach T and Stops there.

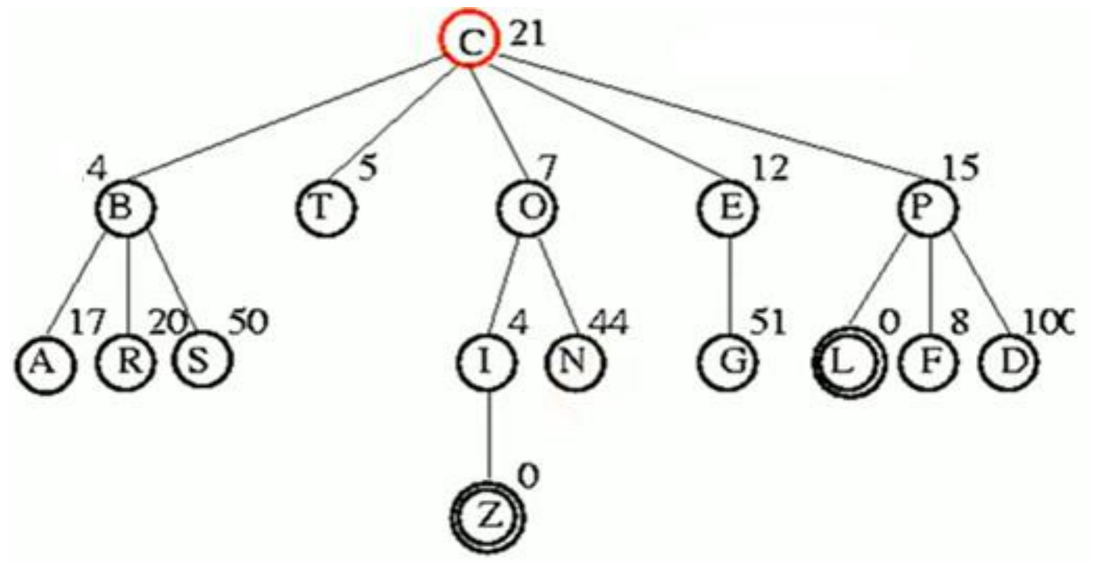
- In Case of BFS:
- Exploring space is Higher (AS it keeps all its successors in open list)

Beam Search Algorithm…



- Beam search algorithm is not complete
- It is not optimal
- The time complexity: The worst-case time = ***O(B*m)***
- The space complexity: The worst-case space complexity = ***O(B*m)***
- ***B*** is the beam width, and m is the maximum depth of any path in the search tree.

Example : Incomplete Solution

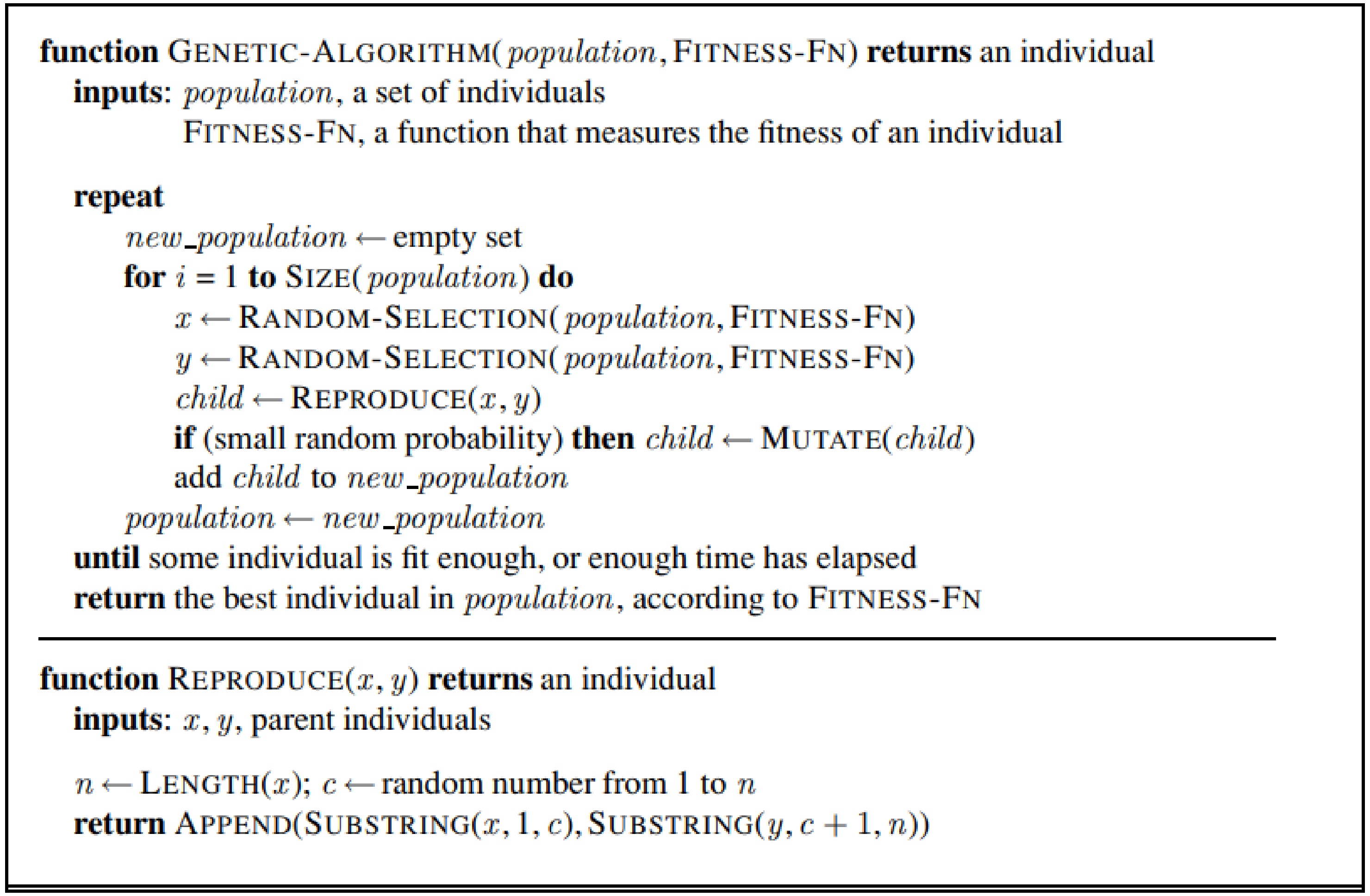


- Beam Search is not complete .
  - In this example initially we will select node B & T.
  - T is ignored.
  - B is explored (but we wont reach goal state)

Genetic Algorithm

- Inspired by evolutionary biology and natural selection, such as inheritance. • Evolves toward better solutions. • A successor state is generated by combining two parent states, rather by modifying a single state. • Start with k randomly generated states (population), Each state is an individual
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s) • Evaluation function (fitness function). Higher values for better states. • Produce the next generation of states by selection, crossover, and mutation. • Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population

Genetic Algorithm



```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

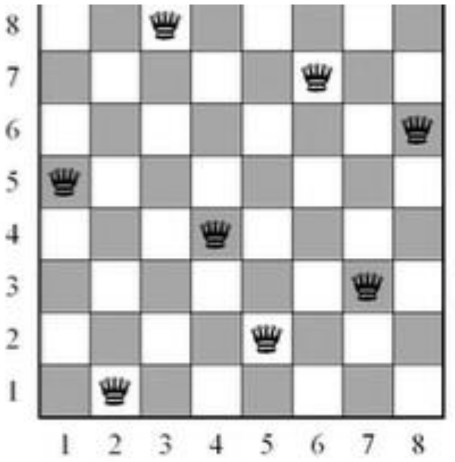
```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

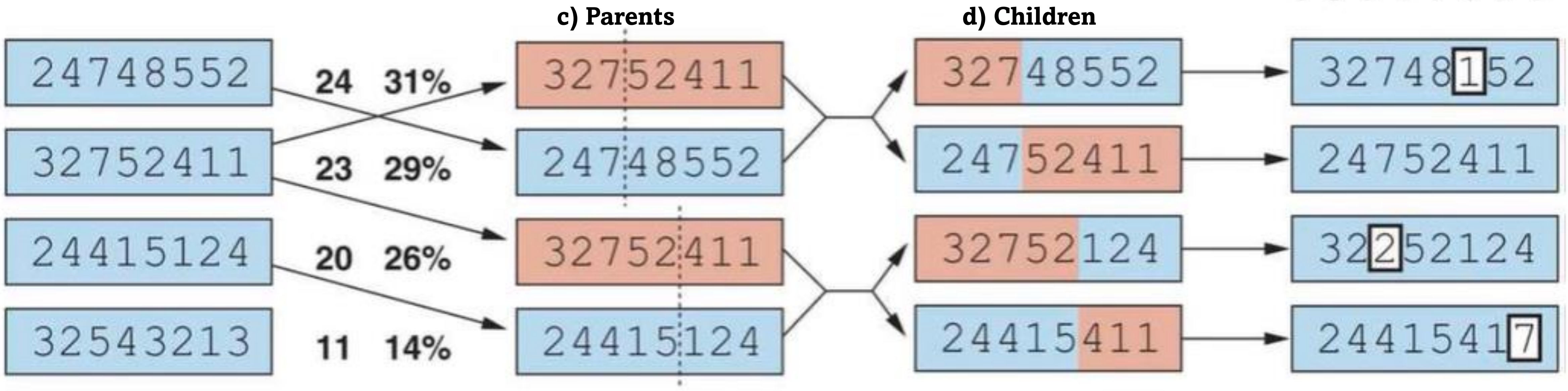
Figure 4.8 A genetic algorithm. The algorithm is the same as the one diagrammed in Figure 4.6, with one variation: in this more popular version, each mating of two parents produces only one offspring, not two.

- Idea: stochastic local beam search + generate successors from pairs of states
- GAs require states encoded as strings.
- Crossover helps iff substrings are meaningful components.
- GAs 6= evolution.
  - e.g., real genes encode replication machinery.

Genetic algorithm for 8 Queens problem

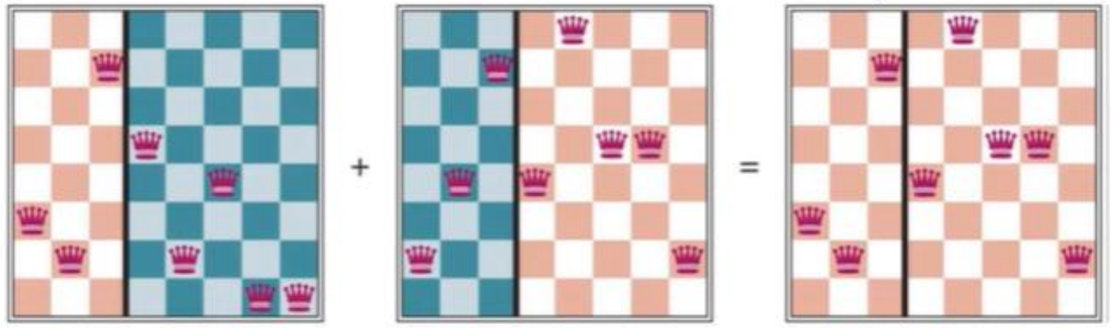
- A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).



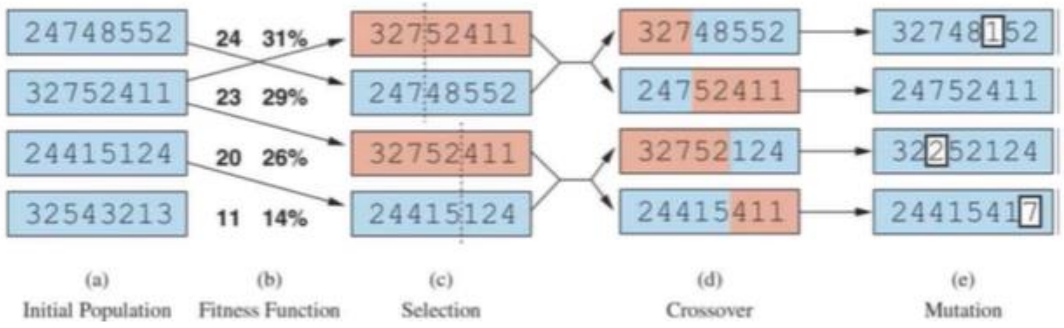


a) Initial Population b) Fitness Function c) Selection d) Crossover e) Mutation

- The 8-queens states corresponding to the first two parents, and the first offspring,
- The green columns are lost in the crossover step and the red columns are retained. row 1 is the bottom row, and 8 is the top row.



= Result
(New population)



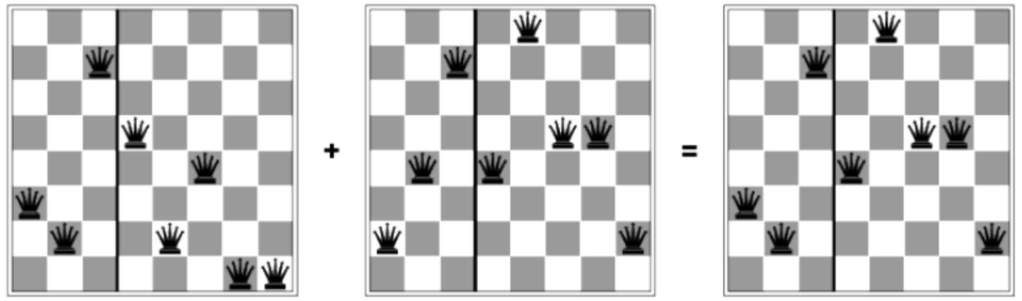
Result

- Consider this "Result" as "New Population".
- For "New Population" perform 5 steps procedure.
- Iteration- Continues until "Goal State "is reached.
- Goal state = Placement of all queen in chess board (No queen should attack other queen in single movement).

8 –Queens Problem Solution : In Detail

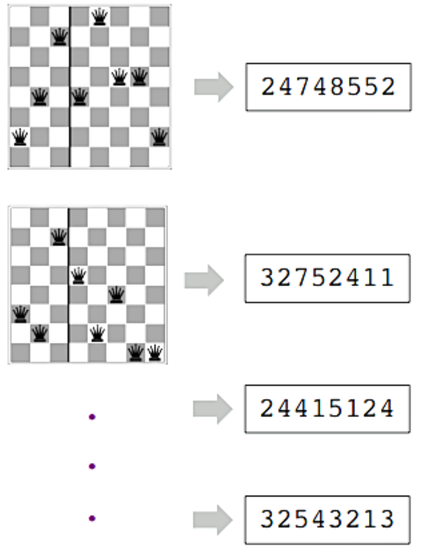
Try to better position the queens using the genetic algorithm.
A better state is generated by combining two parent states.

The good genes (features) of the parents are passed onto the children.



Represent individuals (chromosomes) :

Can be represented by a string digits 1 to 8, that represents the position of the 8 queens in the 8 columns.

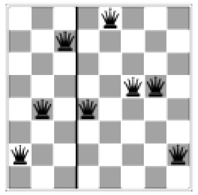


Step 1: Represent Individuals (Chromosomes)

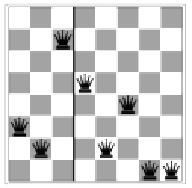
8 –Queens Problem Solution :Step 2

Fitness Function :

Possible fitness function is the number of non-attacking pairs of queens. (min = 0, max = 8 × 7/2 = 28)



24
24748552



23
32752411

20
24415124

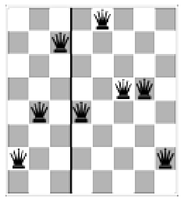
11
32543213

·
·
·

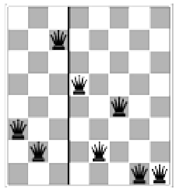
Step 2: Represent Individuals (Chromosomes)

Fitness Function :

Calculate the probability of being regenerated in next generation. For example: 24/(24+23+20+11) = 31% , 23/(24+23+20+11) = 29% , etc.



24
24748552 31%



23
32752411 29%

20
24415124 26%

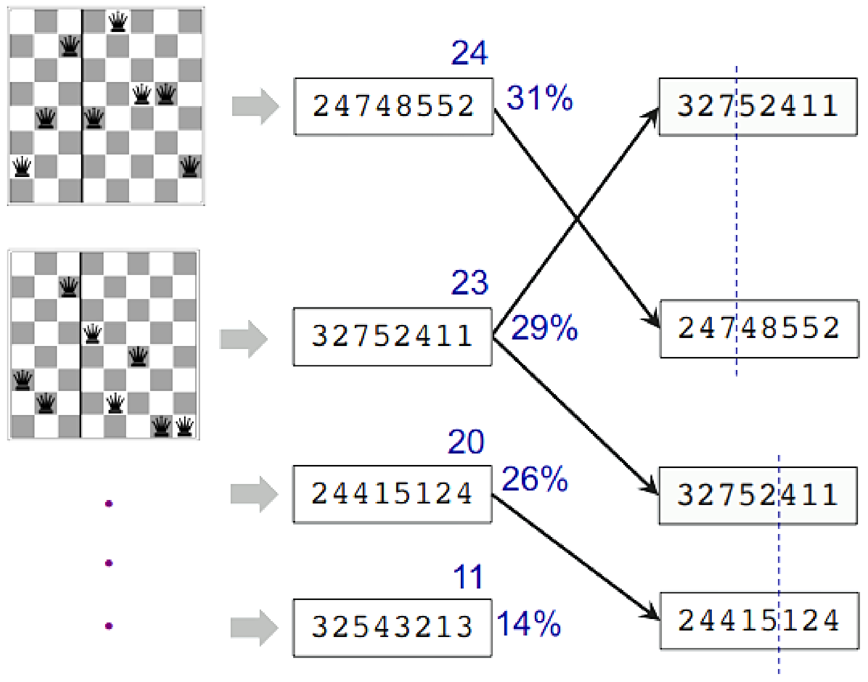
11
32543213 14%

·
·
·

8 –Queens Problem Solution : Step 3 & 4

Selection:

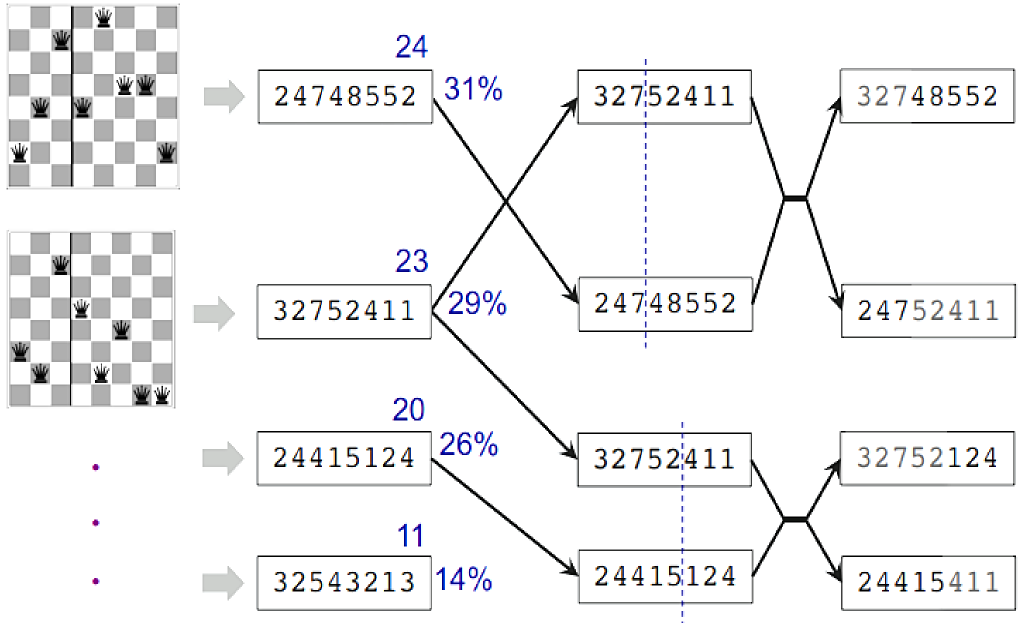
Pairs of individuals are selected at random for reproduction w.r.t. some probabilities. Pick a crossover point per pair.



Step 3: Selection

Crossover

A **crossover** point is chosen randomly in the string. **Offspring** are created by crossing the parents at the crossover point.

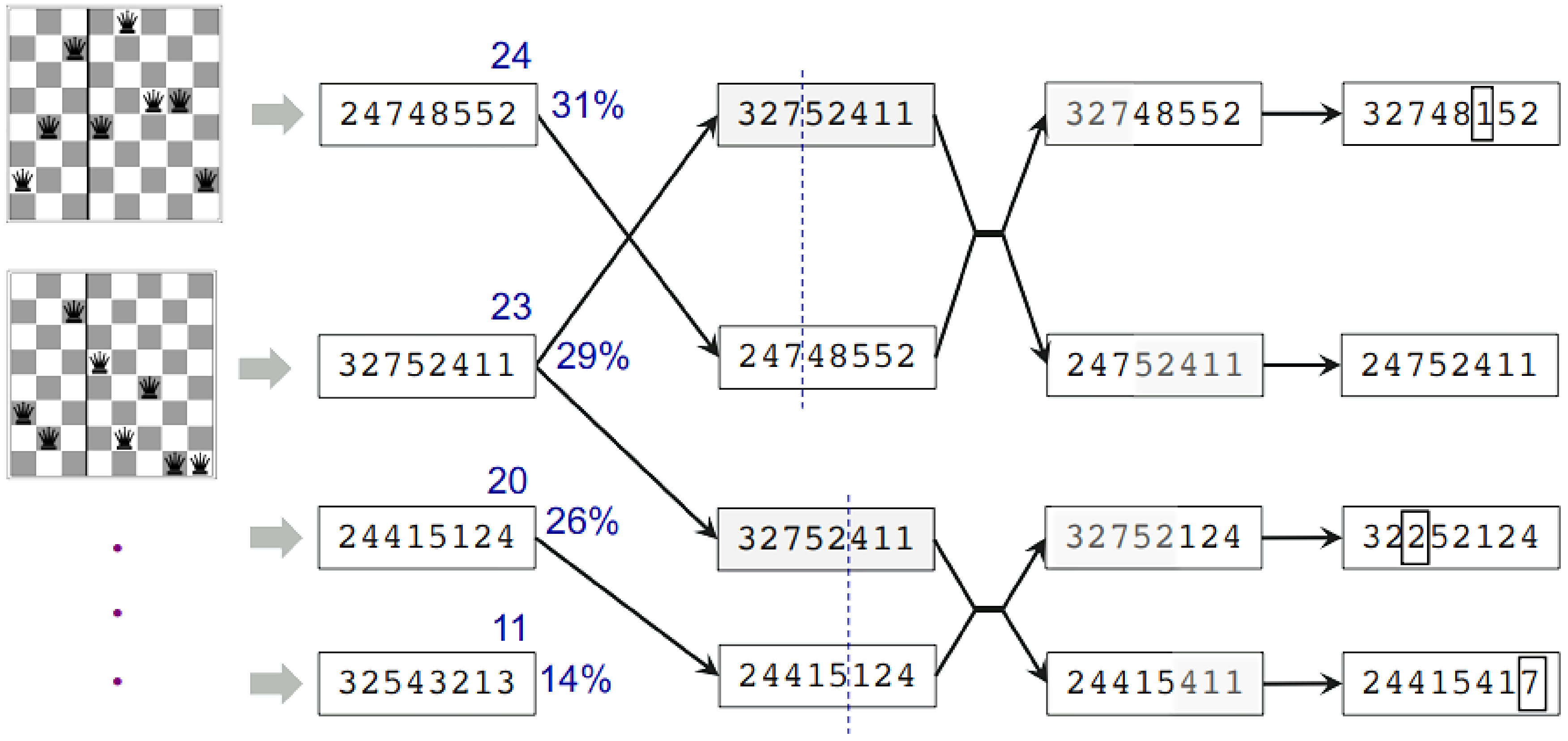


Step 4: Crossover

8 –Queens Problem Solution : Step 5

Mutation

Each element in the string is also subject to some mutation with a small probability.



Step 5: Mutation

Summary

- Hill climbing is a steady monotonous ascent to better nodes.
- Simulated annealing, local beam search, and genetic algorithms are "random" searches with a bias towards better nodes.
- All need very little space which is defined by the population size.
- None guarantees to find the globally optimal solution

References

1. S. Russell and P. Norvig: Artificial Intelligence: A Modern Approach Prentice Hall, 2003, Second Edition.
2. AIMA textbook (3rd edition)
3. AIMA slides (http://aima.cs.berkeley.edu/
4. http://en.wikipedia.org/wiki/SMA*
5. Moonis Ali: Lecture Notes on Artificial Intelligence http://cs.txstate.edu/~ma04/files/CS5346/SMA%20search.pdf
6. Max Welling: Lecture Notes on Artificial Intelligence https://www.ics.uci.edu/~welling/teaching/ICS175winter12/A-starSearch.pdf
7. Kathleen McKeown: Lecture Notes on Artificial Intelligence http://www.cs.columbia.edu/~kathy/cs4701/documents/InformedSearch-AR-print.ppt
8. Franz Kurfess: Lecture Notes on Artificial Intelligence http://users.csc.calpoly.edu/~fkurfess/Courses/Artificial-Intelligence/F09/Slides/3-Search.ppt

Thank you