



# Outline

---

- ▶ **Local Search**
  - ▶ Hill Climbing
  - ▶ Simulated Annealing
  - ▶ Genetic Algorithm

# Local Search

---

- ▶ They apply mostly to problems for which we **don't need to know the path to the solution** but only the solution itself.
- ▶ They operate using a single state or a small number of states and explore the neighbours of that state. **They usually don't store the path.**
- ▶ **Example:** optimization problems for which we search for the best solution according to an objective function.

# Local Search

---

## ► Advantages

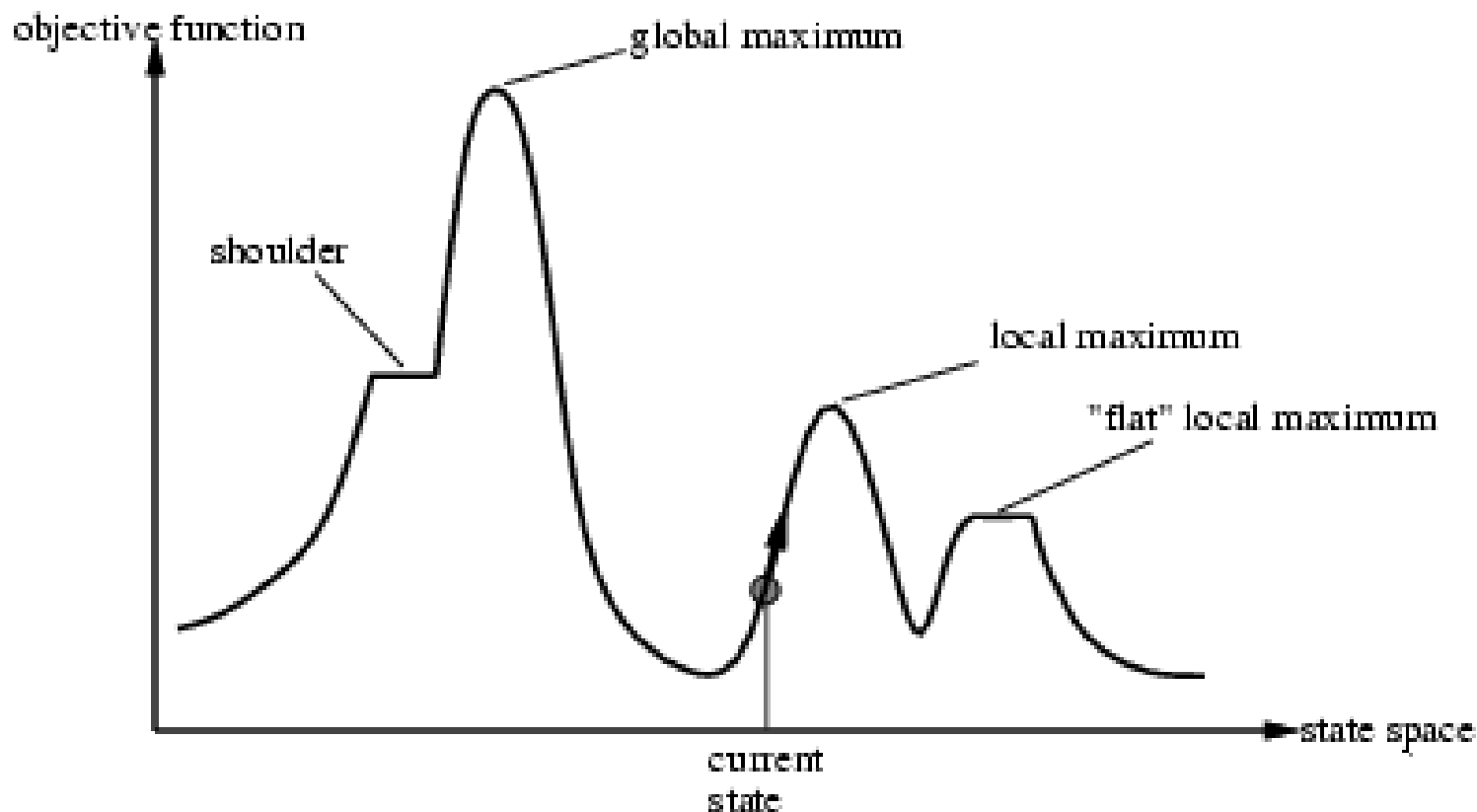
- Use very little memory
- Find reasonable solution in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.

# Local search algorithms

---

- ▶ In many optimization problems, the **path** to the goal is irrelevant; the goal state itself is the solution
- ▶ State space = set of **complete** configurations
- ▶ Find configuration satisfying constraints.
- ▶ In such cases, we can use **local search algorithms**
- ▶ **Idea:** keep a single "current" state, try to improve it

# Terminology of Local Search



# Local search algorithms

---

- ▶ **State Space Landscape**
  - ▶ **Global maximum** - a state that maximizes the objective function over the entire landscape.
  - ▶ **Local maximum** - a state that maximizes the objective function in a small area around it.
  - ▶ **Plateau** - a state such that the objective function is constant in an area around it. (Area of state space)
  - ▶ **Shoulder** - a plateau that has an uphill edge.
  - ▶ **Flat** - plateau whose edges go downhill.
  - ▶ **Ridge** - sequences of local maxima.



# Types of Local Search

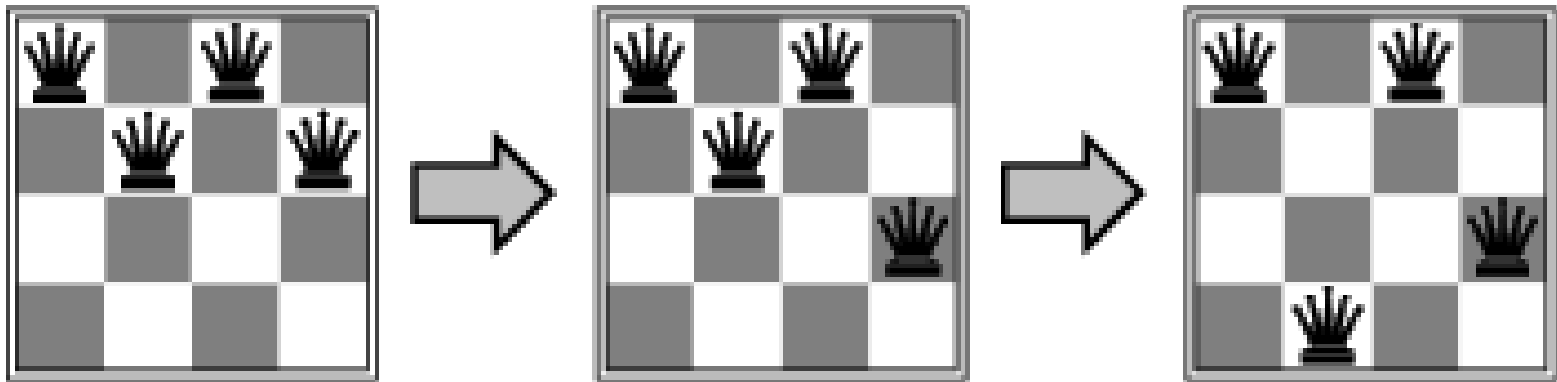
---

- ▶ Hill-climbing Search
- ▶ Simulation Annealing Search
- ▶ Genetic Algorithm



# Example: $n$ -queens

- Put  $n$  queens on an  $n \times n$  board with no two queens on the same row, column, or diagonal





# Hill-climbing search

---

- ▶ It always moves towards the goal
- ▶ Using heuristics it finds which direction will take it closest to the goal
- ▶ It is actually combination of **Generate-and-test** + **Direction to move**
- ▶ Here the heuristic function is to estimate how close a given state is to a goal state

# Algorithm Steps for Hill-climbing

---

**STEP 1:** Evaluate the initial state, if it is goal state then quit otherwise make current state as initial state

**STEP 2:** Select a new **operator** that could be applied to this state and generate a new state

**STEP 3:** Evaluate the new state if this new state is closer to the goal state, then current state make the new state as the current if it is not better ignore this state and proceed with the current state

**STEP 4:** If the current state is goal state or no new operators are available, Quit. Otherwise go to **STEP 2**.

---





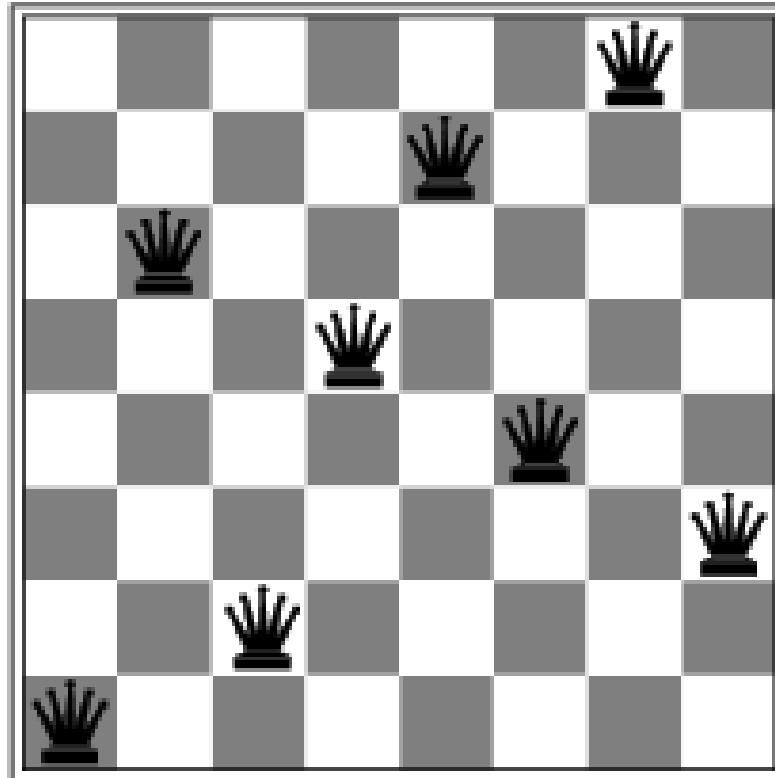
# Hill-climbing search

- ▶ "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
```

# Hill-climbing search: 8-queens problem



A local minimum with  $h = 1$



# Variations of Hill Climbing

---

- ▶ **Stochastic HC:** Choose randomly among the neighbours going uphill.
  - ▶ **First-choice HC:** generate random successors until one is better. Good for states with high numbers of neighbours.
  - ▶ **Random restart:** the sideway moves restart from a random state.
  - ▶ **Evolutionary hill-climbing:** represents potential solutions as strings and performs random mutations. Keeps the mutations that are better states. It's a particular case of first-choice and the ancestor of the genetic algorithms.
- 



# Simulated annealing search

---

- ▶ Annealing is the process used to temper or harden the metals and glass by heating them to a high temperature and then cooling them.
- ▶ Thus allowing the material to coalesce in to low energy crystalline state.

# Simulated annealing search

---

- ▶ Initially the whole space is explored.
- ▶ It makes the procedure less sensitive to the starting point
- ▶ It avoid false foot hills based on the following changes done in this approach
  - ▶ Rather than creating maxima; **minimisation** is done
  - ▶ The term **objective function** is used rather than heuristic



# Algorithm form of Simulated annealing search

**STEP 1:** Start with evaluating the initial state

**STEP 2:** Apply each operator and loop until a goal state is found or till no new operators left to be applied:

Set T according to an annealing schedule

Select and applies a new operator

Evaluate the new state: If it is a goal state quit; Otherwise

$$\Delta E = \text{Val}(\text{current state}) - \text{Val}(\text{new state})$$

If  $\Delta E < 0$  then this is the new current state

Else find a new current state with probability  $e^{-E/T}$







# Simulated annealing search

## Homework!!!

**function** SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

**inputs:** *problem*, a problem

*schedule*, a mapping from time to “temperature”

**local variables:** *current*, a node

*next*, a node

*T*, a “temperature” controlling prob. of downward steps

*current*  $\leftarrow$  MAKE-NODE(INITIAL-STATE[*problem*])

**for** *t*  $\leftarrow$  1 **to**  $\infty$  **do**

*T*  $\leftarrow$  *schedule*[*t*]

**if** *T* = 0 **then return** *current*

*next*  $\leftarrow$  a randomly selected successor of *current*

$\Delta E \leftarrow$  VALUE[*next*] – VALUE[*current*]

**if**  $\Delta E > 0$  **then** *current*  $\leftarrow$  *next*

**else** *current*  $\leftarrow$  *next* only with probability  $e^{\Delta E/T}$



# Properties of simulated annealing search

---

- ▶ One can prove: If  $T$  decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
  
- ▶ Types of searches in this method as shown below
  - ▶ Local Beam search
  - ▶ Stochastic Beam search



# Local Beam search

- ▶ In this search, instead of single state in memory  **$K$  states are kept in memory**
- ▶ A successor function plays an important role by generating successor of all  $K$  states
- ▶ If any one successor state is goal state then no further processing is required
- ▶ In other case i.e. if goal state is not achieved it observed it observes the  $K$  best successors from the list of all successor and process is repeated

# Local Beam search

---

- ▶ Keep track of  $k$  states rather than just one
- ▶ Start with  $k$  randomly generated states

## Limitations

- ▶ Lack of variation among the  $K$  states
- ▶ If the state concentrate on small area of state space then search becomes more expensive



# Stochastic Beam search

---

- ▶ It's a flavour of Local Beam search it resolve the limitations exist in Local Beam search
- ▶ It focus on **random selection of K successor** instead of selecting K best successor from candidate successor



# Genetic algorithms

---

- ▶ A successor state is generated by combining two parent states
  - ▶ Start with  $k$  randomly generated states (**population**)
  - ▶ A state is represented as a **string (gene)** 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
- 



# Genetic algorithms

- ▶ **Population:** Population is set of states which are generated randomly
- ▶ **Individual:** Each state or individual is a string of finite alphabet (0's & 1's )
- ▶ **Fitness function:** The evaluation function which specifies the rating of each state
- ▶ **Crossover:** For each state pairs are divided that division point or meeting point is called crossover point
- ▶ **Mutation:** Mutation is one of the genetic operation. It works on random selections or changes.
- ▶ **Schema:** The schema is a substring in which position of some bit can be unspecified



# Algorithm Representation

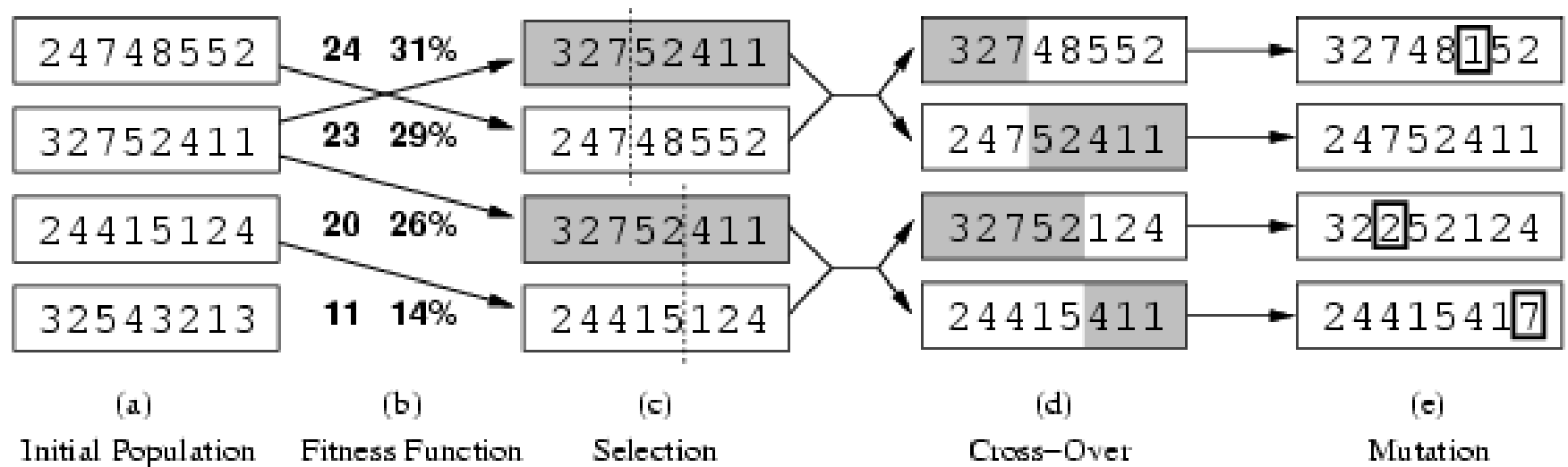
---

1. Use number of state population or a set of individual
2. Use fitness function, function of rating
3. Create an individual 'x' (parent) by using random selection with fitness function 'A' (rate A)
4. Create an individual 'y' (parent) by using random selection with fitness function 'B' (rate B)
5. Child with good fitness is created for  $x+y$
6. For small probability apply mutate operator on child
7. Add child to new population
8. The above process is repeated until child (an individual) in not fit as specified fitness function

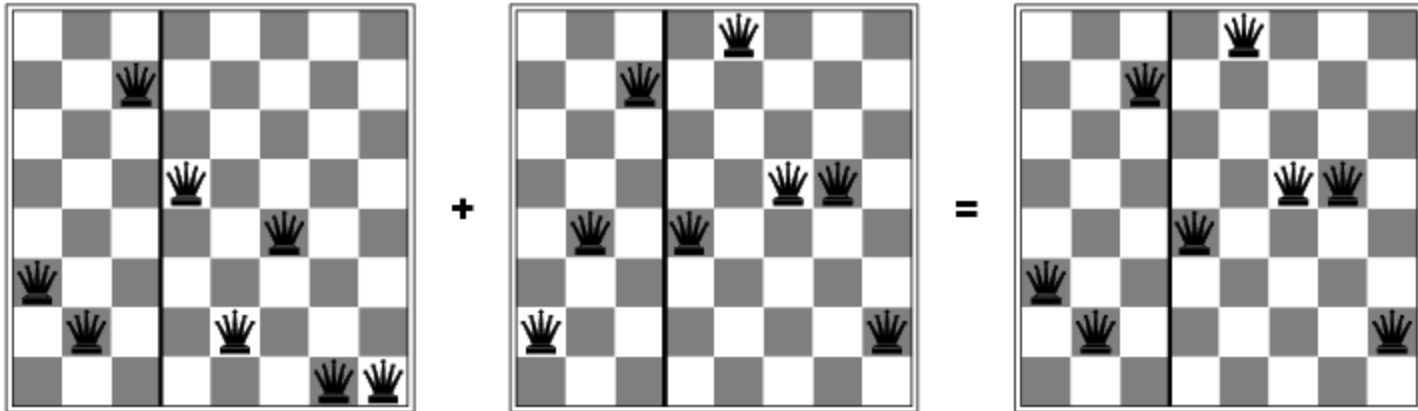




# Genetic algorithms



# Genetic algorithms



- ▶ **Fitness function: number of non-attacking pairs of queens**
- ▶  $\text{min} = 0$ ,
- ▶  $\text{max} = 8 \times 7/2 = 28$  i.e.  $(n*(n-1)/2)$



# VIT<sup>®</sup>

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

*Any Query ???*

Thank you !



# VIT<sup>®</sup>

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)