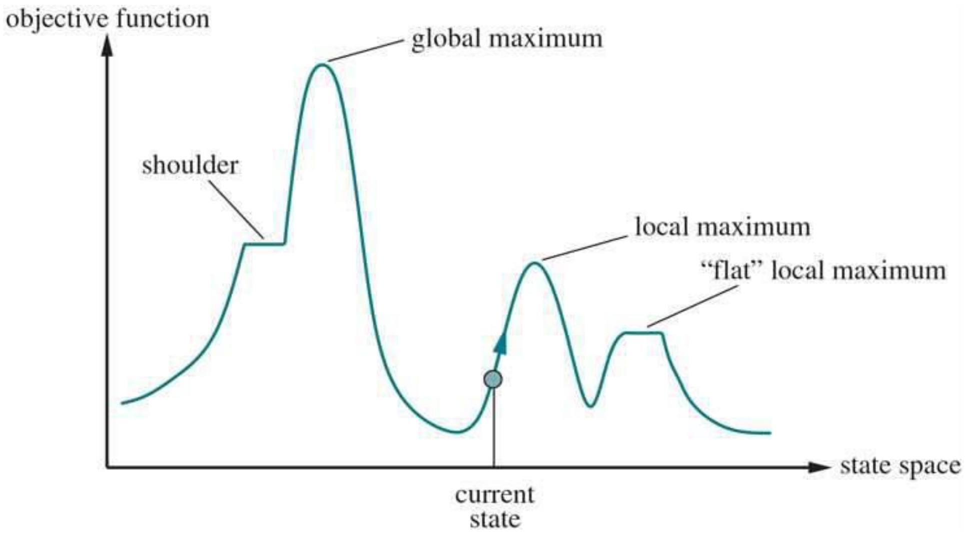
SEARCH IN COMPLEX ENVIRONMENT

LOCAL SEARCH



- Local search algorithms operate by searching from a start state to neighboring states,
  - without keeping track of the paths,
  - without keeping a track of the set of reached states.
- If Objective function = Elevation

then Objective: Global Maximum (hill Climbing)

- If Cost function = Elevation

then Objective: Global minimum (Gradient Descent)

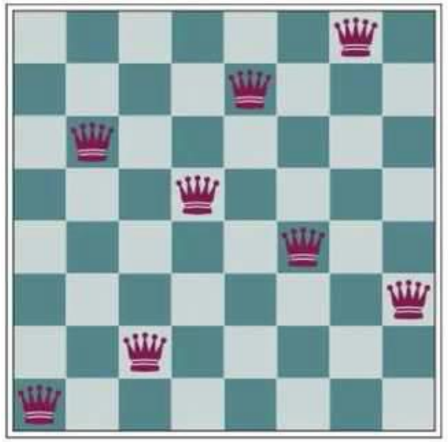
HILL-CLIMBING SEARCH

- It keeps track of one current state.
- On each iteration moves to the neighboring state with highest value—that is, it heads in the direction that provides the steepest ascent.
- It terminates when it reaches a "peak" where no neighbor has a higher value.
- Hill climbing does not look ahead beyond the immediate neighbors of the current state.
- Trying to find the top of Mount Everest in a thick fog while suffering from Amnesia.

HILL-CLIMBING SEARCH

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

HILL-CLIMBING SEARCH

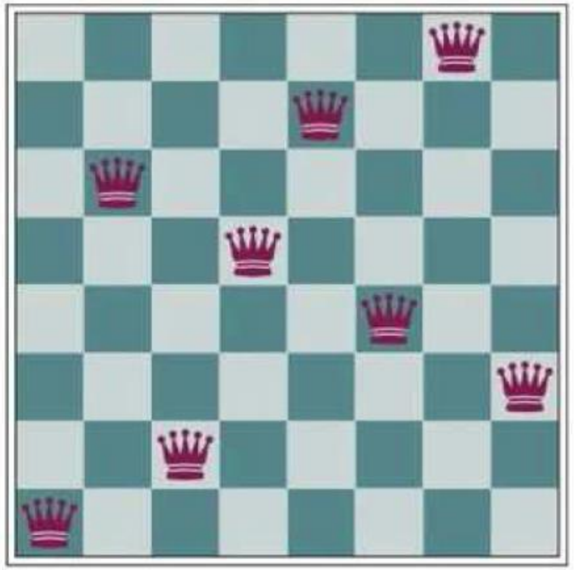


(a)

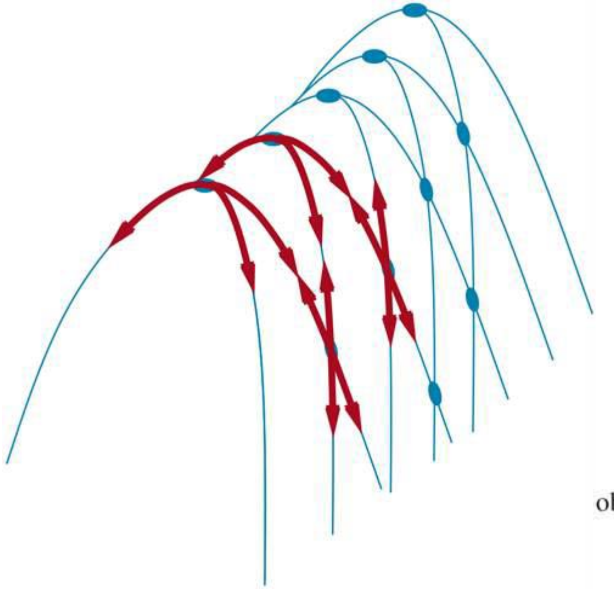
(b)

- The heuristic cost function is the number of pairs of queens that are attacking each other; this will be zero only for solutions.
- Number of Successors to initial state = 56
- Greedy local search

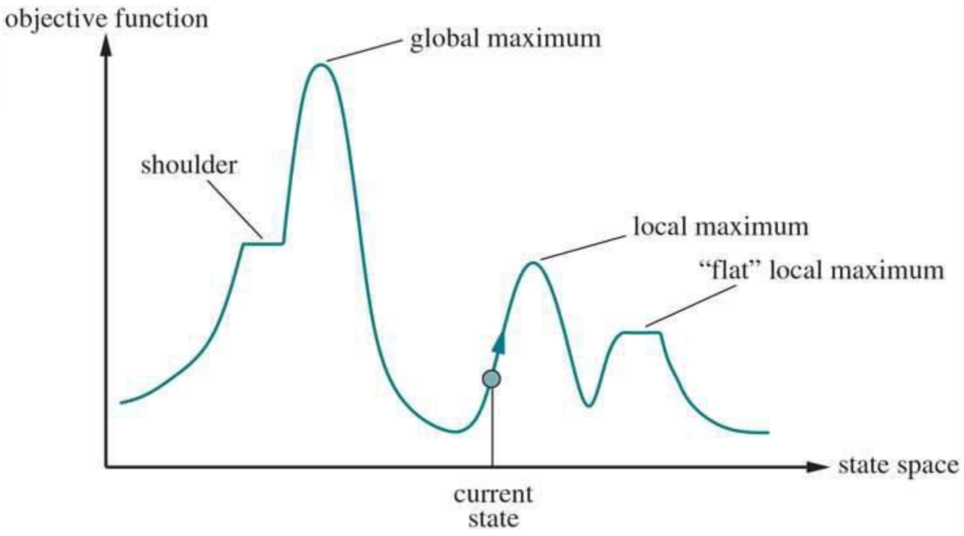
CHALLENGES FOR HILL-CLIMBING SEARCH



- Local Maxima
- Ridges
- Plateaus



- Solutions
  - Sideways move
  - Stochastic hill climbing
  - First-choice hill climbing
  - Random-restart hill climbing



SIMULATED ANNEALING

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

LOCAL BEAM SEARCH

- Keeps track of k states rather than just one.
- Useful information is passed among the parallel search threads.

- Challenges:
- Lack of diversity among the states.

LOCAL SEARCH IN CONTINUOUS SPACE

- Suppose we want to place three new airports anywhere in Romania, such that the sum of squared straight-line distances from each city on the map to its nearest airport is minimized.
- The objective function $f(x) = f(x_1,y_1,x_2,y_2,x_3,y_3)$
- Let C_i be the set of cities whose closest airport (in the state x) is airport i.

$$f(\mathbf{x}) = f(x_1,y_1,x_2,y_2,x_3,y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

- Discretize the Continuous space!

LOCAL SEARCH IN CONTINUOUS SPACE

- (x_i, y_i) fixed points on a rectangular grid with spacing of size δ (delta).
- Each state in the space would have only 12 successors, corresponding to incrementing one of the 6 variables by ±δ.

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

- Solve the equation for $\nabla f = 0$.
- Local Gradient

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c)$$

LOCAL SEARCH IN CONTINUOUS SPACE

- Local hill climbing $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$
- α is step size (ML term: learning rate)
- Newton–Raphson method for finding roots of functions

$$x \leftarrow x - g(x)/g'(x)$$

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$$

- $H_f(x)$ is the Hessian matrix.

- Constrained optimization!