



# Artificial Intelligence

Beyond Classical Search



# Previous Search Algorithms

- Designed to explore search space systematically:
  - keep one or more paths in memory
  - record which have been explored and which have not
  - a path to goal represents the solution



# 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, e.g., n- queens
  - what matters is the final configuration of queens, not the order in which they are added.
- In such cases, we can use **local search algorithms**
- keep a single "current" state, try to improve it



# Local search algorithms

- Local search algorithms work by iteratively improving the candidate solution by making small changes and evaluating the objective function after each change.
- The objective function helps guide the search process by determining whether a given solution is better or worse than the current solution and helps determine the direction of the search.
- The algorithm continues this process until it reaches a satisfactory solution, or it determines that it cannot find a better solution.



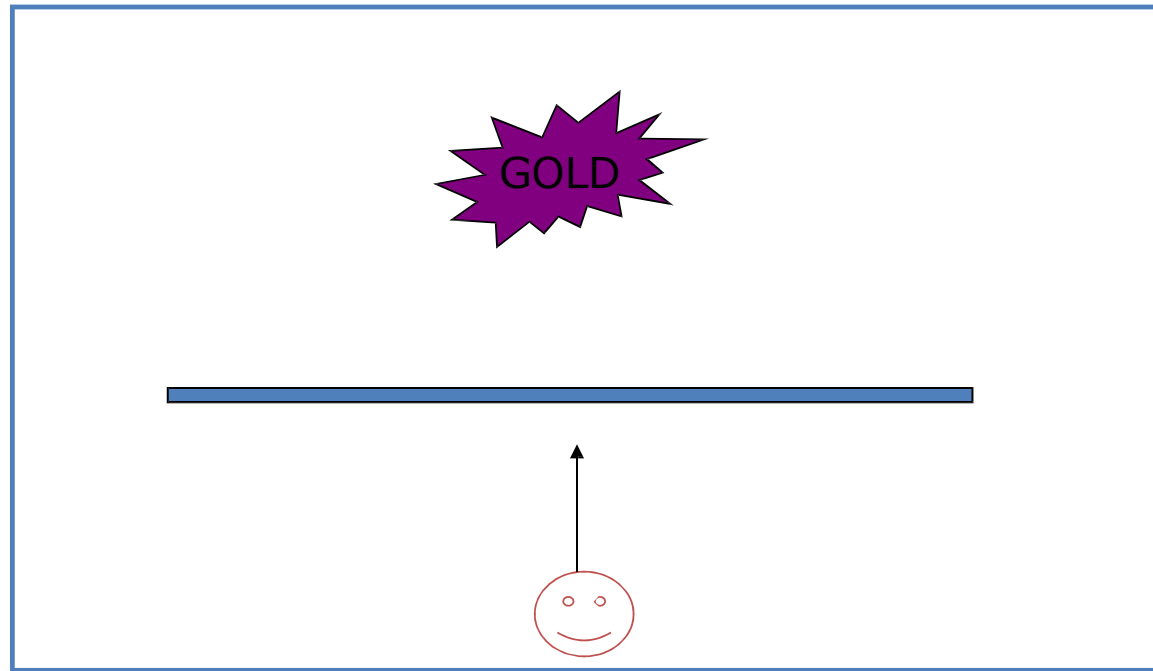
# Local Search Algorithms

- Advantages:
  - Use very little memory
  - Find reasonable solution in infinite(Continuous) State Space for which systematic algorithms are unsuitable

Local Search Algorithms are useful for Optimization Problems, in which aim is to find best state according to an **Objective Function**



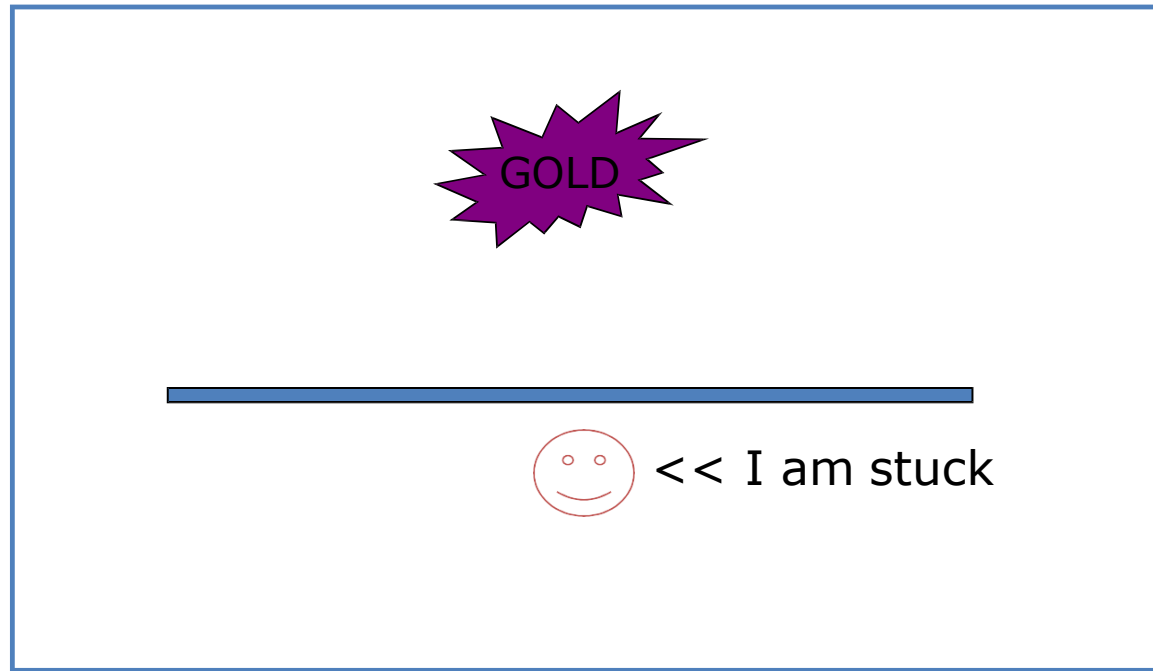
# Example: Initial State



Assume the objective function measures the straight-line distance



# Example: Local Minima

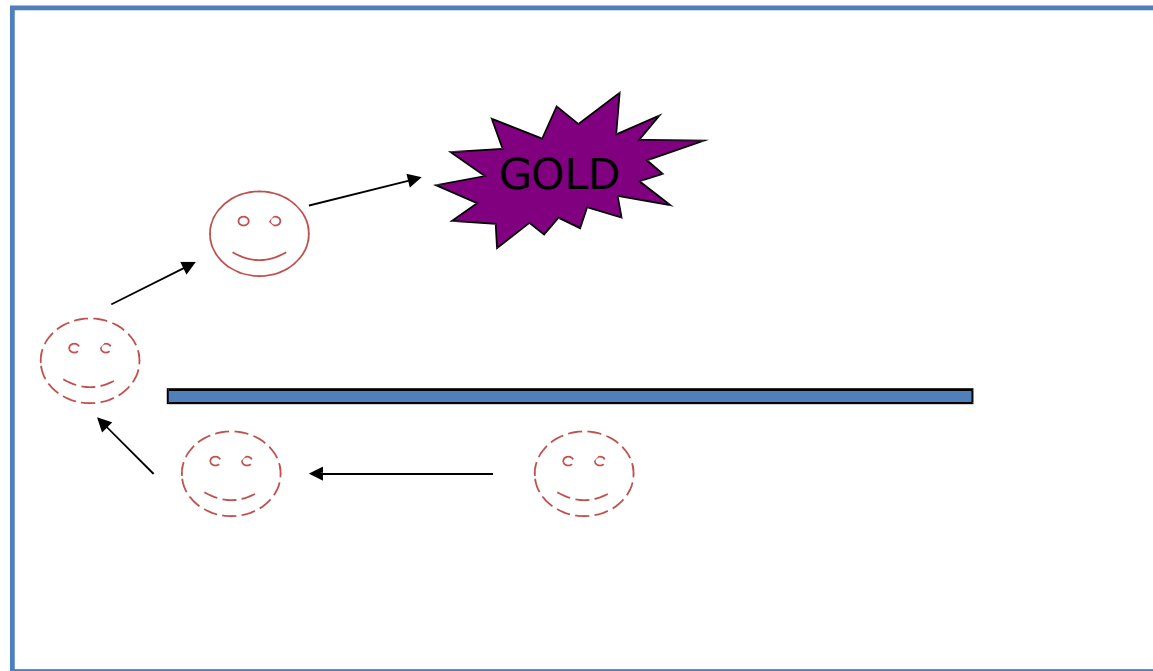


Assume the objective function measures the straight-line distance



# Example: A Plausible Solution

Making some “bad” choices is actually not that bad



Assume the objective function measure the straight-line distance





# Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

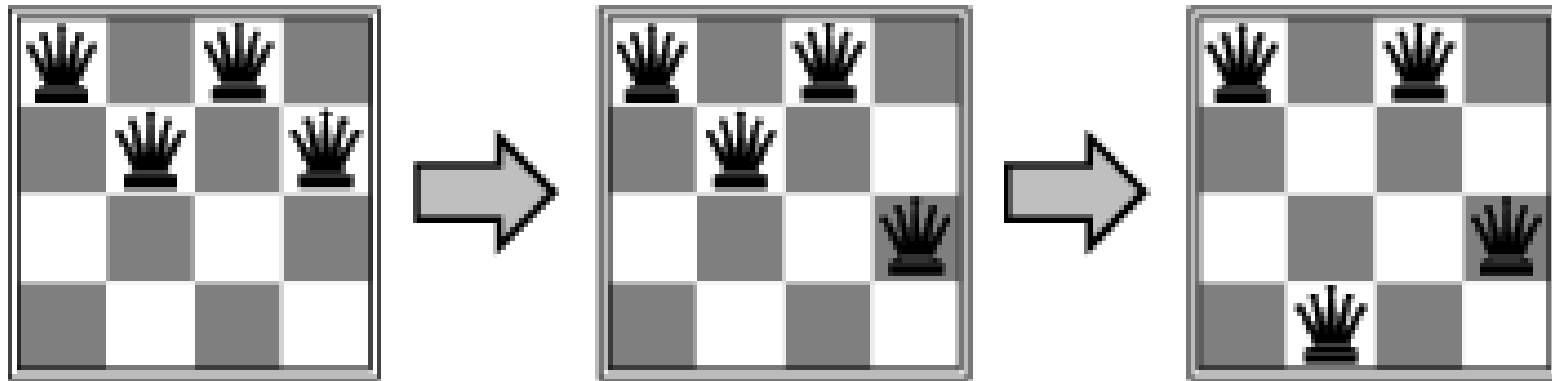
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                   neighbor, a node

  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
```

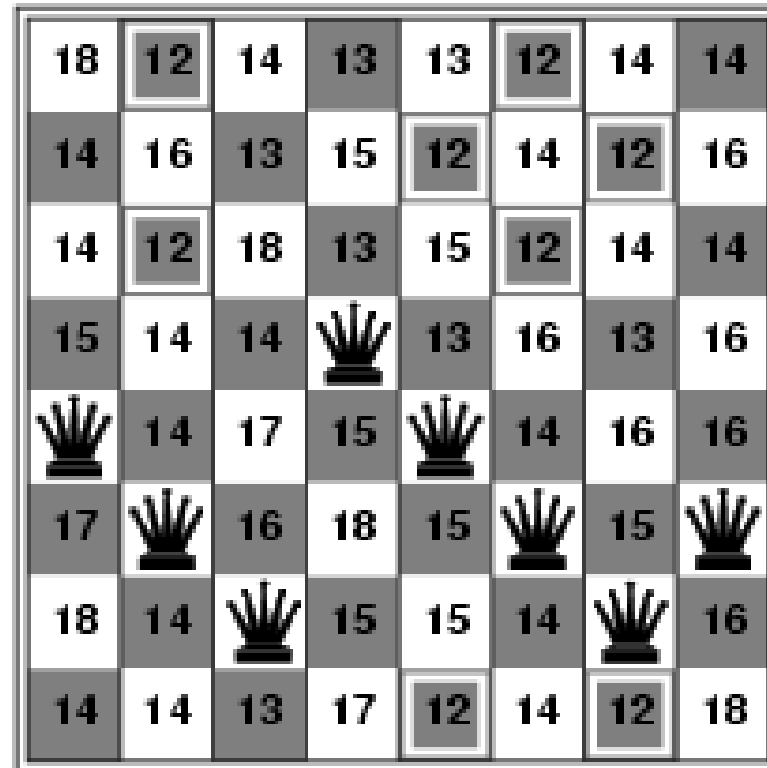


## Example: $n$ -queens

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



## Hill-climbing search: 8-queens problem



An 8x8 chessboard illustrating the 8-queens problem. Each cell contains a number representing the number of queens that attack it. Queens are placed on the board at the following positions: (row, column) = (5, 1), (6, 2), (7, 3), (4, 4), (5, 5), (6, 6), (7, 7), and (8, 8). The numbers in the board are as follows:

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	Queen	13	16	13	16
Queen	14	17	15	Queen	14	16	16
17	Queen	16	18	15	Queen	15	Queen
18	14	Queen	15	15	14	Queen	16
14	14	13	17	12	14	12	18

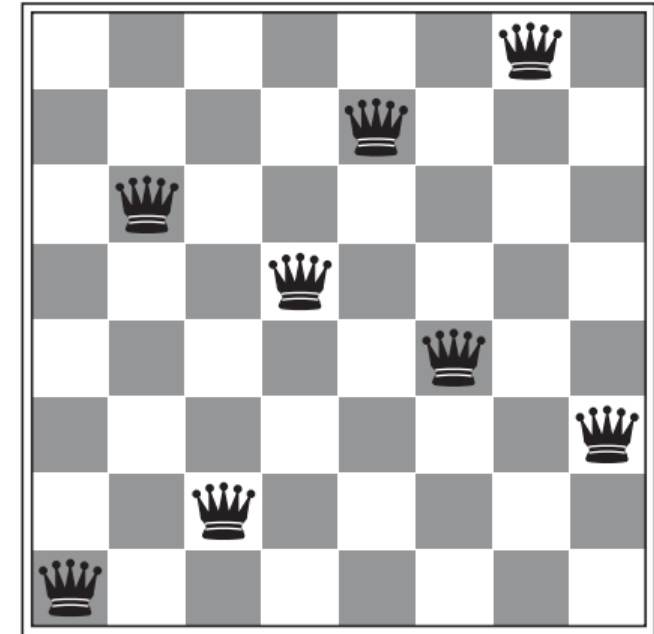
- $h$  = number of pairs of queens that are attacking each other, either directly or indirectly

# Hill-climbing search: 8-queens problem

- It takes just five steps to reach the state in (b), which has  $h = 1$  and is very nearly a solution

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♙	13	16	13	16
♙	14	17	15	♙	14	16	16
17	♙	16	18	15	♙	15	♙
18	14	♙	15	15	14	♙	16
14	14	13	17	12	14	12	18

(a)



(b)

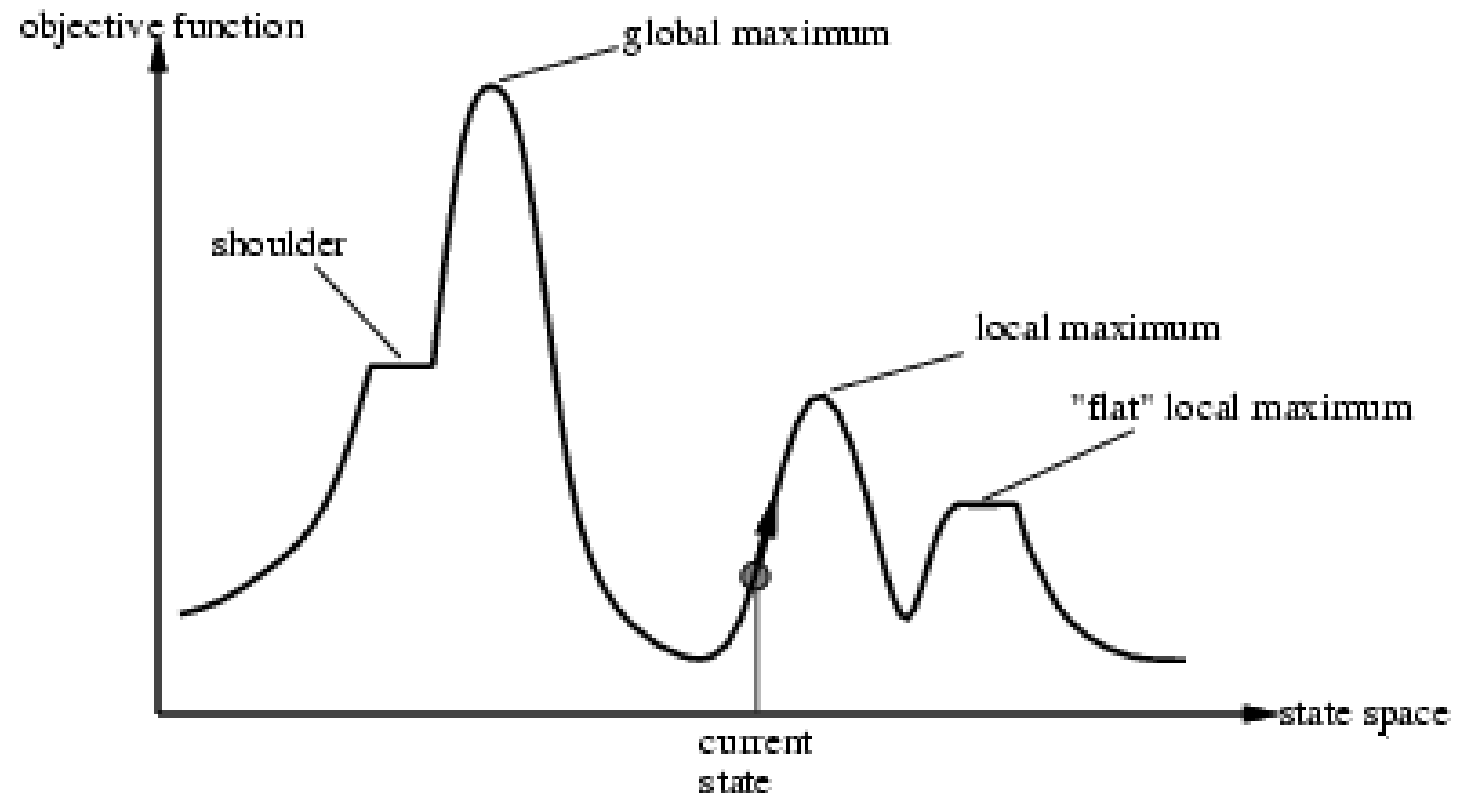
**Figure 4.3** (a) An 8-queens state with heuristic cost estimate  $h = 17$ , showing the value of  $h$  for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has  $h = 1$  but every successor has a higher cost.



# Hill-climbing search

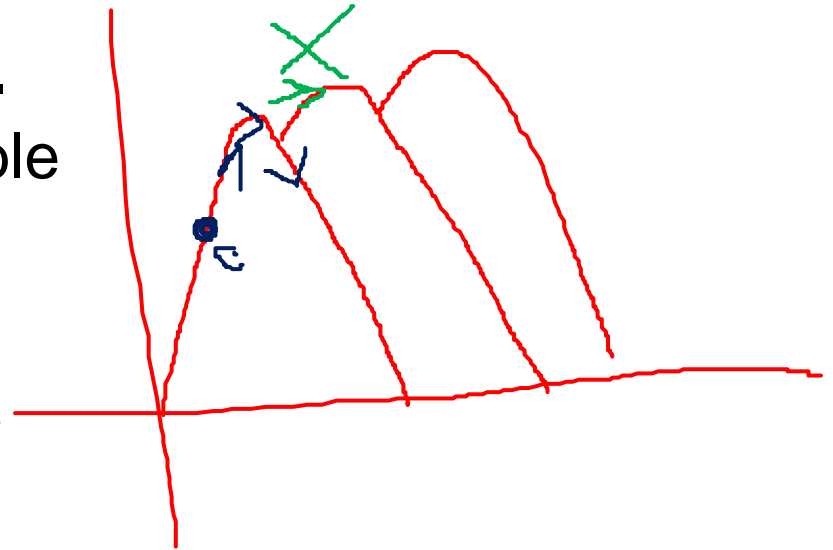
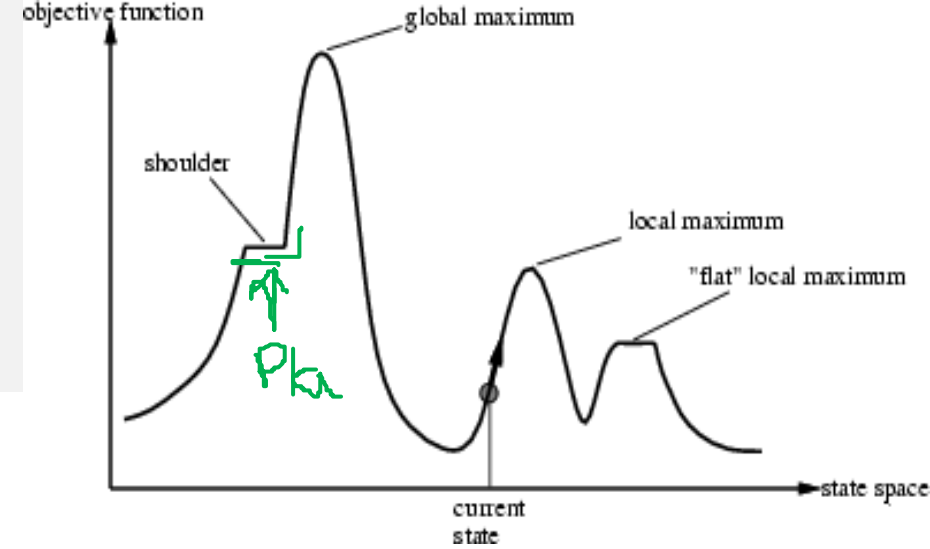
- Problem: depending on initial state, can get **stuck** in local maxima

- Local maxima
- Ridges
- Plateaux



# Problems in Hill-climbing search

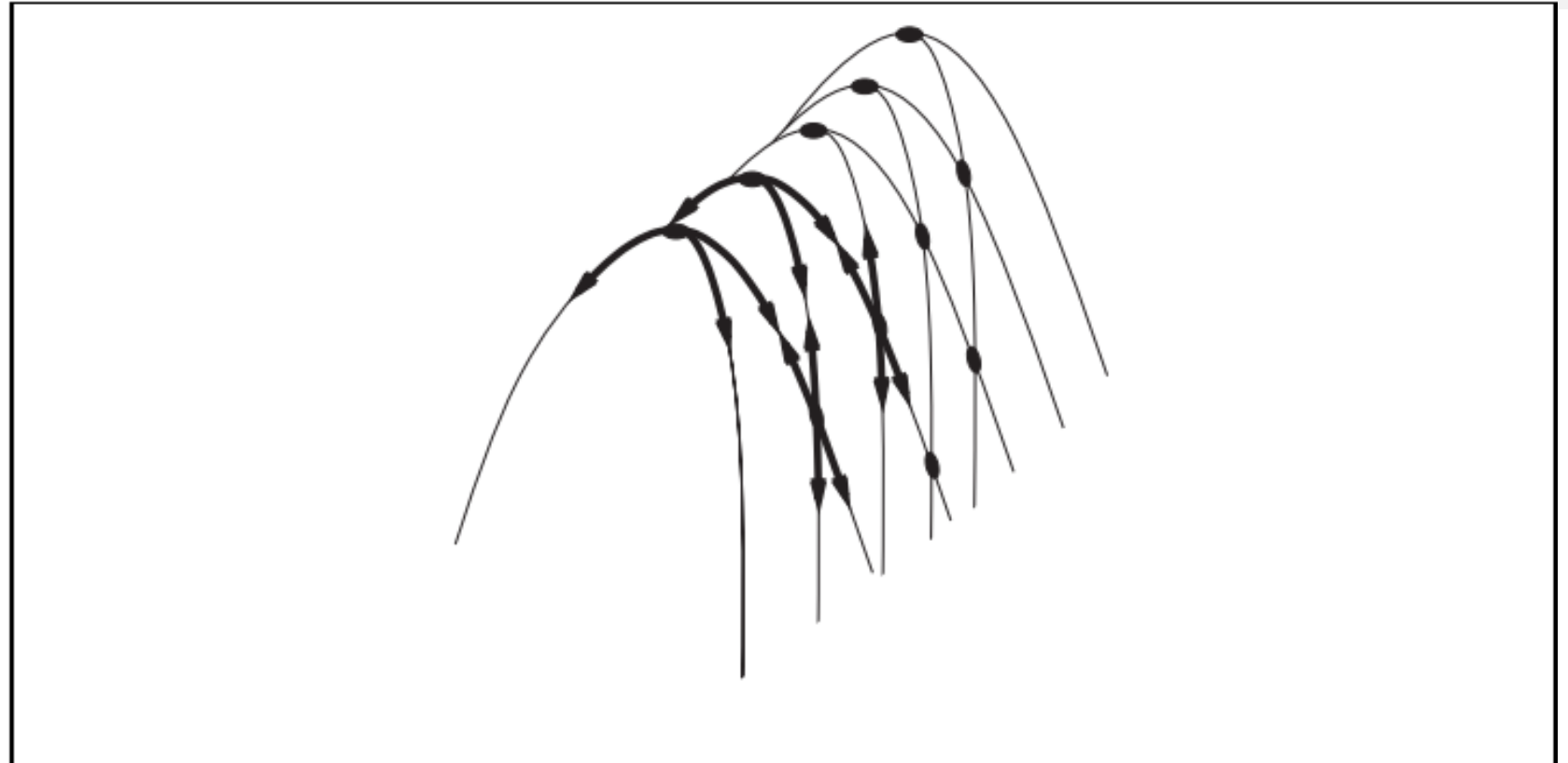
- Local Maxima
- Plateau/ Flat Maxima
  - It is a flat region of state space where neighboring states have the same value.
  - **Shoulder**: from which progress is possible
- Ridges
  - sequence of local maxima that is very difficult for greedy algorithms to navigate





# Problems in Hill Climbing Search

Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate



**Figure 4.4** Illustration of why ridges cause difficulties for hill climbing. The grid of states (dark circles) is superimposed on a ridge rising from left to right, creating a sequence of local maxima that are not directly connected to each other. From each local maximum, all the available actions point downhill.



# Variants of Hill-climbing search

- **Simple Hill Climbing**
  - It examines the neighboring nodes one by one
  - selects the first neighboring node which optimizes the current cost as the next node
- **Steepest Ascent**
  - It first examines all the neighboring nodes
  - selects the node closest to the solution state as the next node.





# Variants of Hill-climbing search

- **Stochastic hill climbing:**
  - chooses at random from among uphill moves
  - probability of selection can vary with the steepness of the uphill move
  - converges more slowly, but finds better solutions in some landscapes

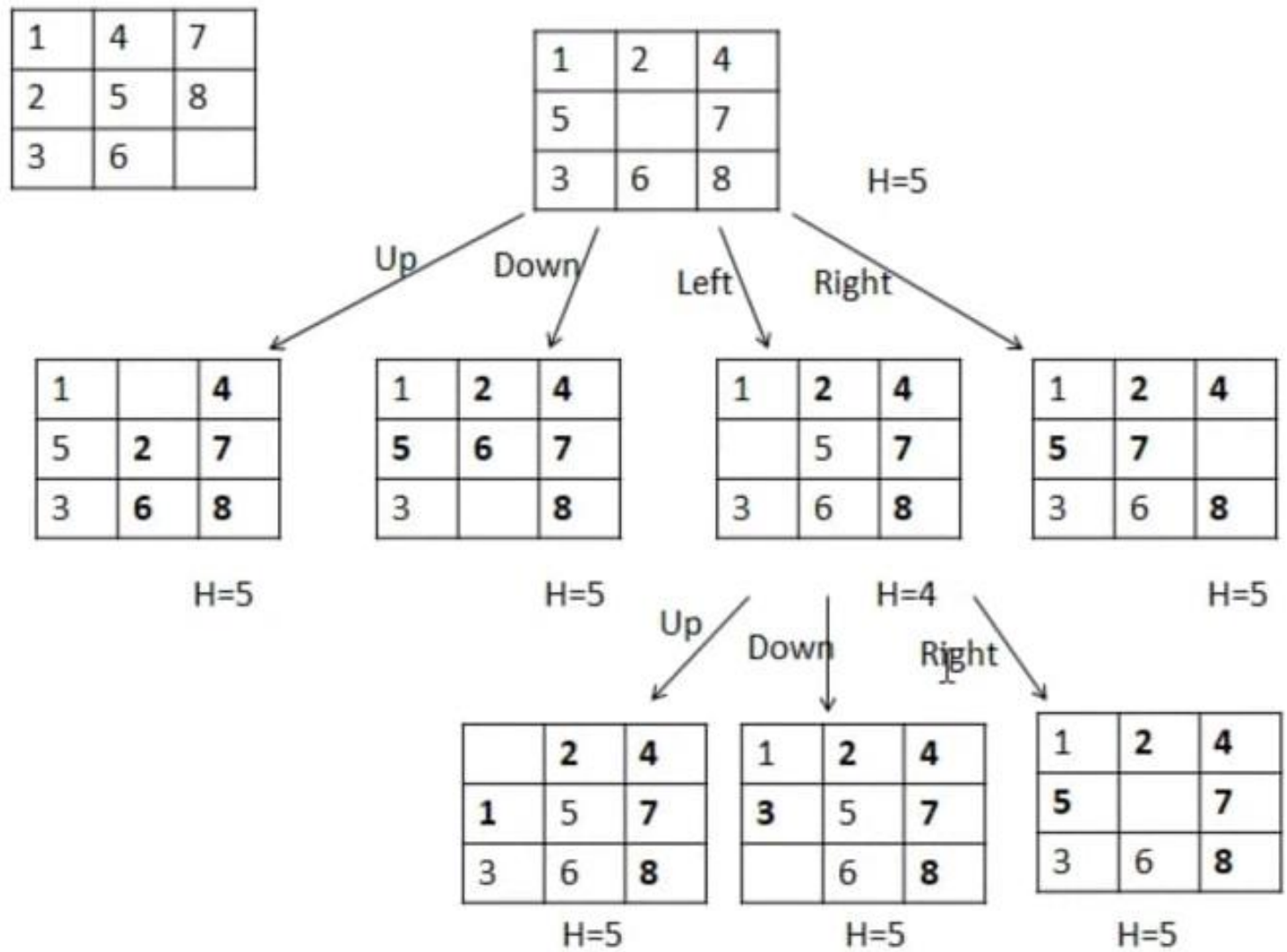
*It does not examine all the neighboring nodes before deciding which node to select.*

*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.*

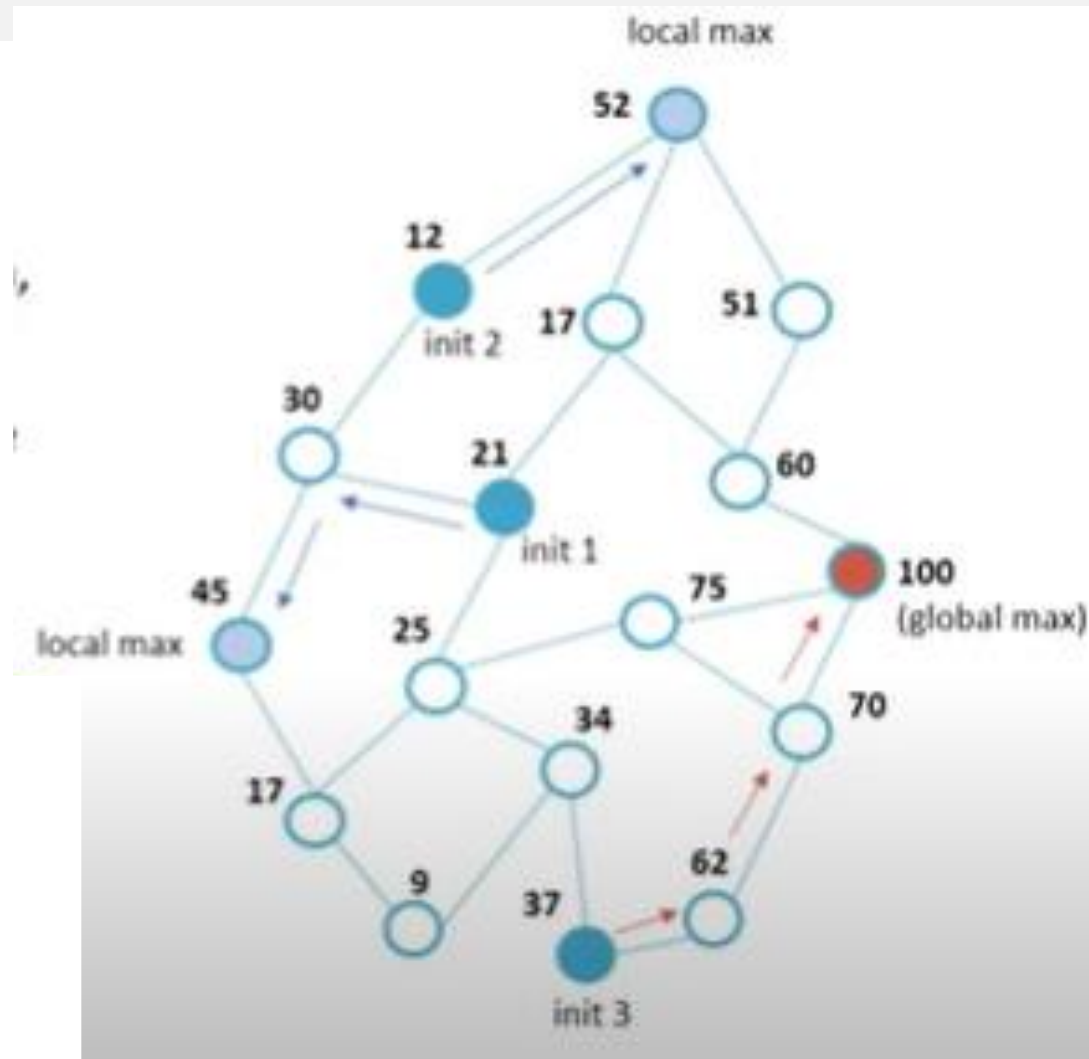


# Variants of Hill-climbing search

- **Random-restart hill climbing:**
  - conducts a series of hill climbing searches from ***randomly generated initial states***, stops when a goal is found
  - It's trivially complete with probability approaching 1, because it will eventually generate a goal state as the initial state.



# Random-Restart Hill Climbing





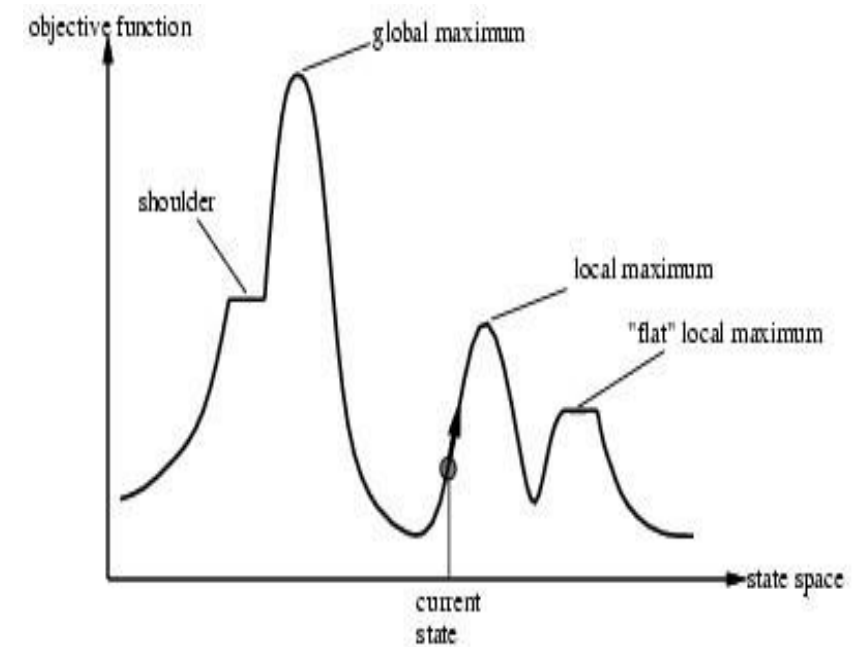
# Random-Restart Hill Climbing

- Assume each hill climbing search has a probability  $p$  of success, then the expected number of restarts required is  $1/p$
- For the 8-Queen problem with no sideways allowed,  $p = 14\%$ , so we need roughly 7 iterations to find a goal (6 failures and 1 success)
  - if the algorithm is run multiple times, in the majority of cases, it will not find a solution in the first 6 iterations. However, in 1 out of 7 iterations, it will find a solution

# Hill Climbing Search

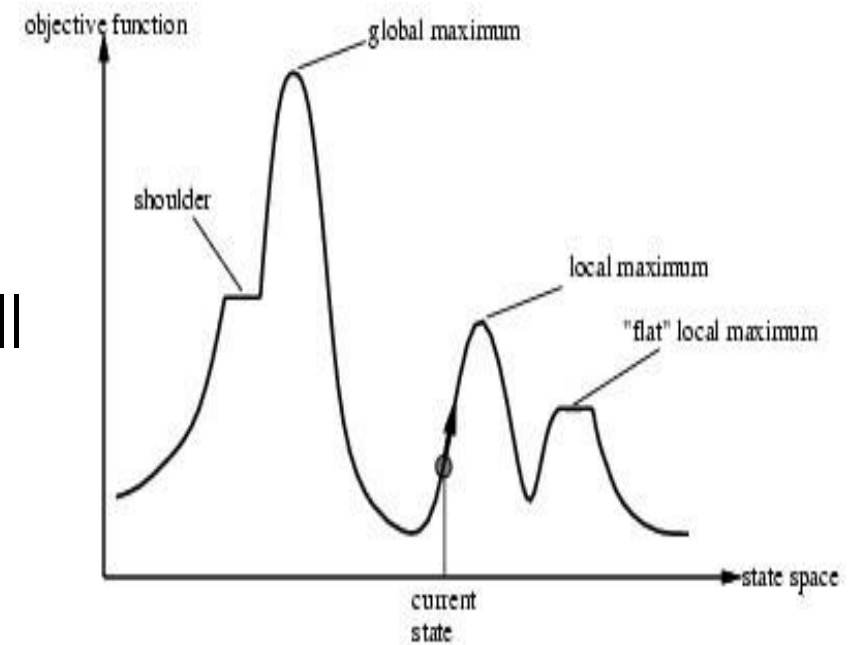
- Random-restart hill climbing is very effective for n-queen problem if we allow sideways moves
- 3 million queens can be solved < 1 min
- Complete? Optimal?

"sideways moves" refer to moves that do not necessarily improve the current solution but instead maintain the same level of fitness or cost.



# Hill Climbing Search

- Complete? Optimal?
- Hill climbing is sometimes called **greedy local search**
- Although greedy algorithms often perform well hill climbing gets stuck when:
  - Local maxima/minima
  - Ridges
  - Plateau (shoulder or flat local maxima/minima)





# Hill-Climbing Search

- NP-hard problems typically have an exponential number of local maxima/minima to get stuck on
- A hill climbing algorithm that never makes “downhill” (or “uphill”) moves is guaranteed to be incomplete
- A purely random walk – moving to a successor chosen uniformly at random – is complete, but extremely inefficient