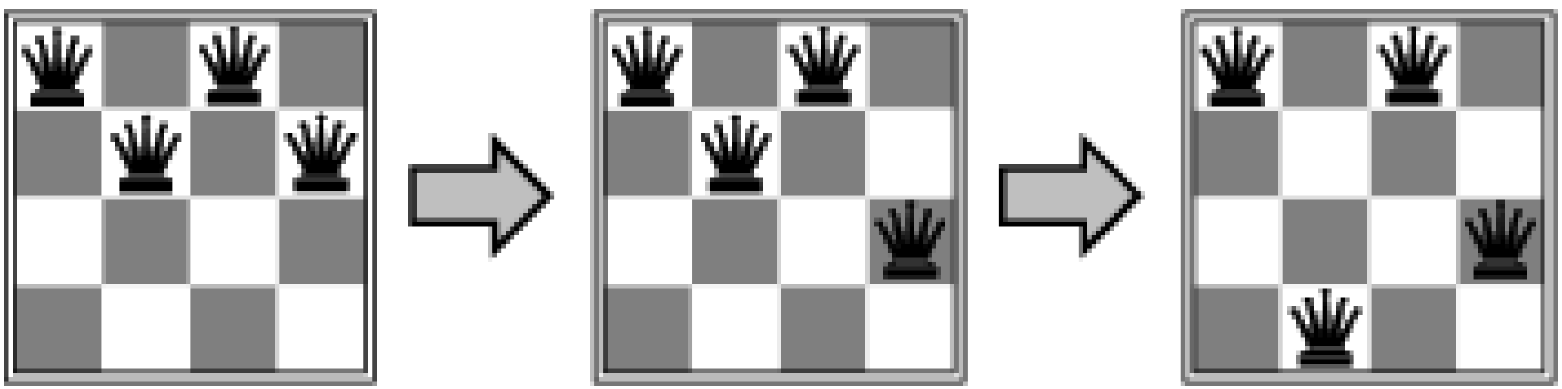
Local search algorithms

- 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

- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

Example: *n*-queens

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

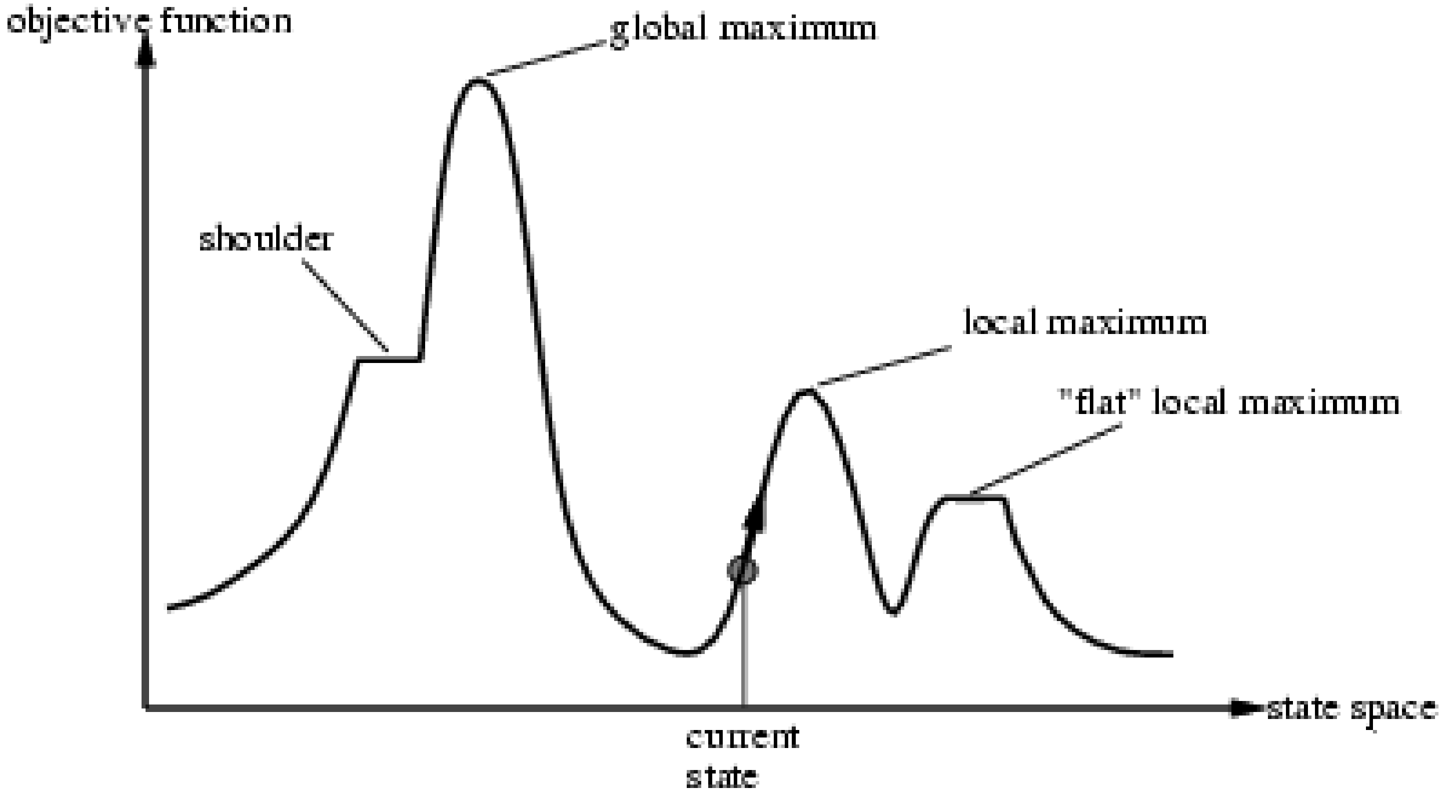
Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"
-

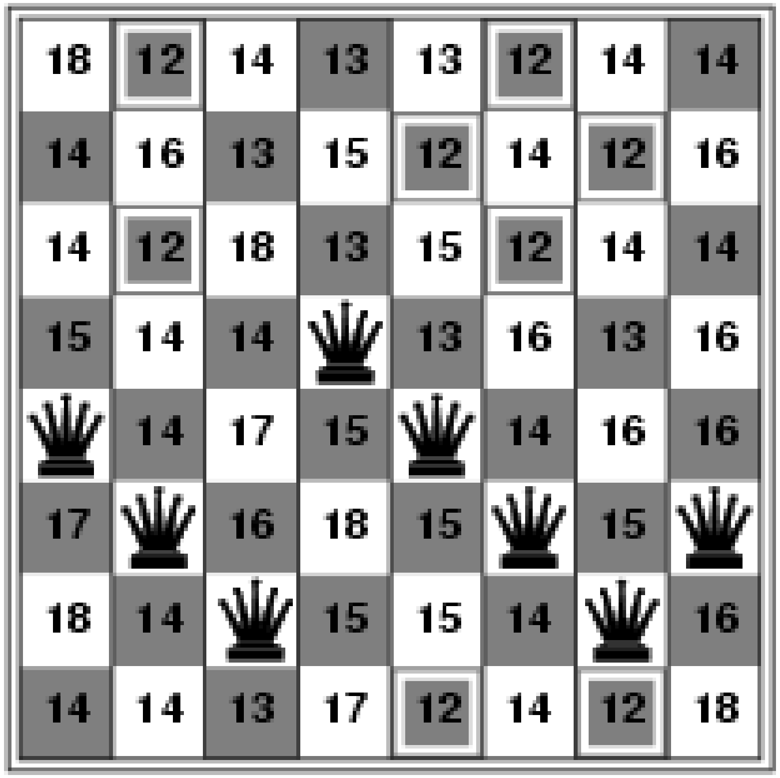
```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    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

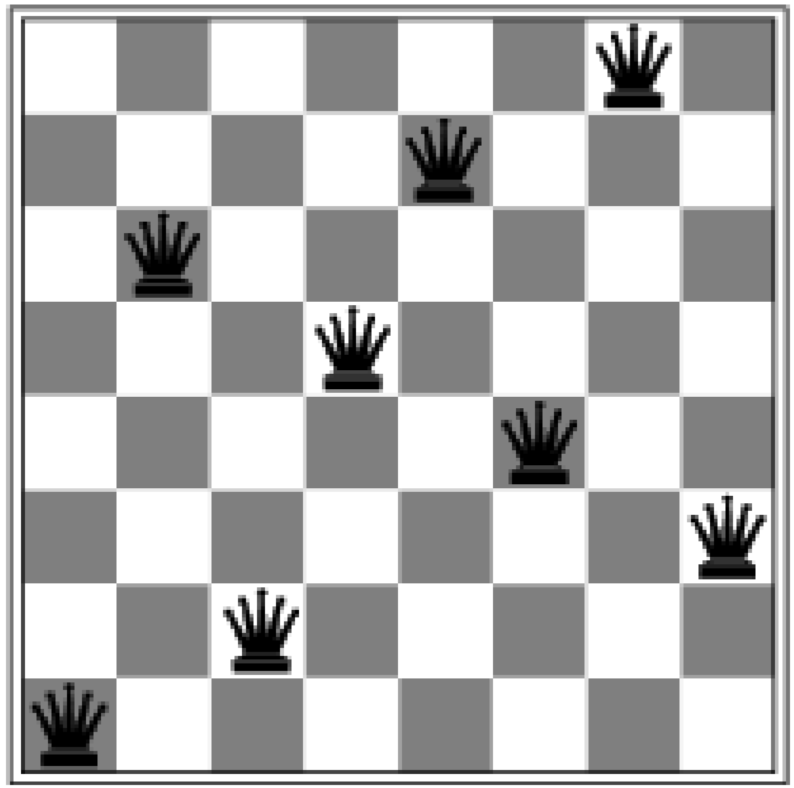
- Problem: depending on initial state, can get stuck in local maxima
- 

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

Hill-climbing search: 8-queens problem



- A local minimum with $h = 1$
-

Simulated Annealing

Annealing is the process of metal casting, where the metal is first melted at a high temperature beyond its melting point and then is allowed to cool down until it returns to the solid form. Thus in physical process of annealing, the hot material gradually loses energy and finally at one point of time reaches a state of minimum energy.

A common observation is that most physical processes have transitions from higher to lower energy states, but there still remains a small probability, p, that it may move up to a higher energy state.

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency
-

```
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 ← 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)
```

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
- Widely used in VLSI layout, airline scheduling, etc

Local beam search

- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

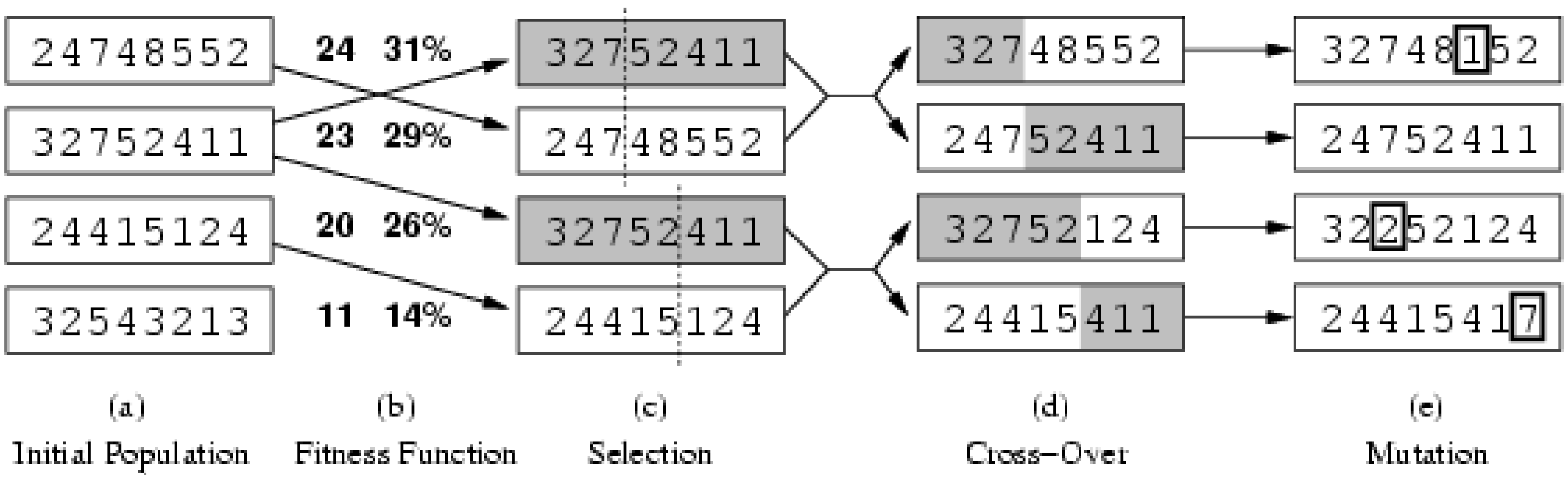
Genetic algorithms

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



- 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 algorithms