Chapter 4

Search in Complex Environments

(Focus on Local Search)

Topic

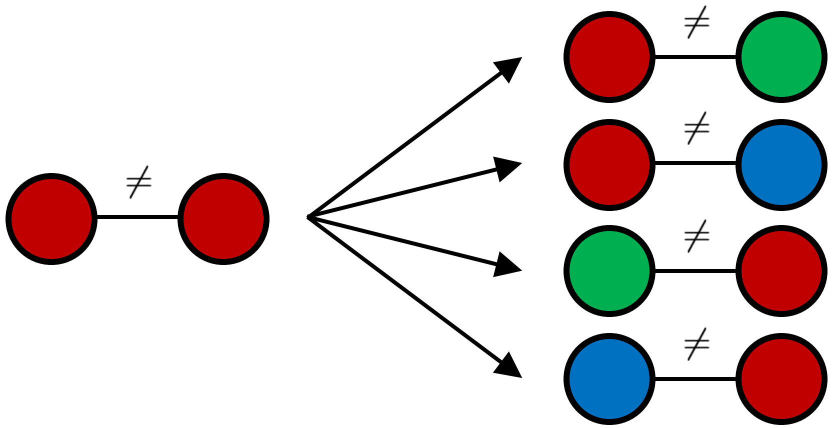
- Local Search and Optimization Problems
- Local search in Continuous spaces
  - Hill Climbing
  - Simulated Annealing
  - Local Beam Search
  - Genetic Algorithm

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

Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve a single option until you can't make it better (no fringe!)
- New successor function: local changes



- Generally, much faster and more memory efficient (but incomplete and suboptimal)

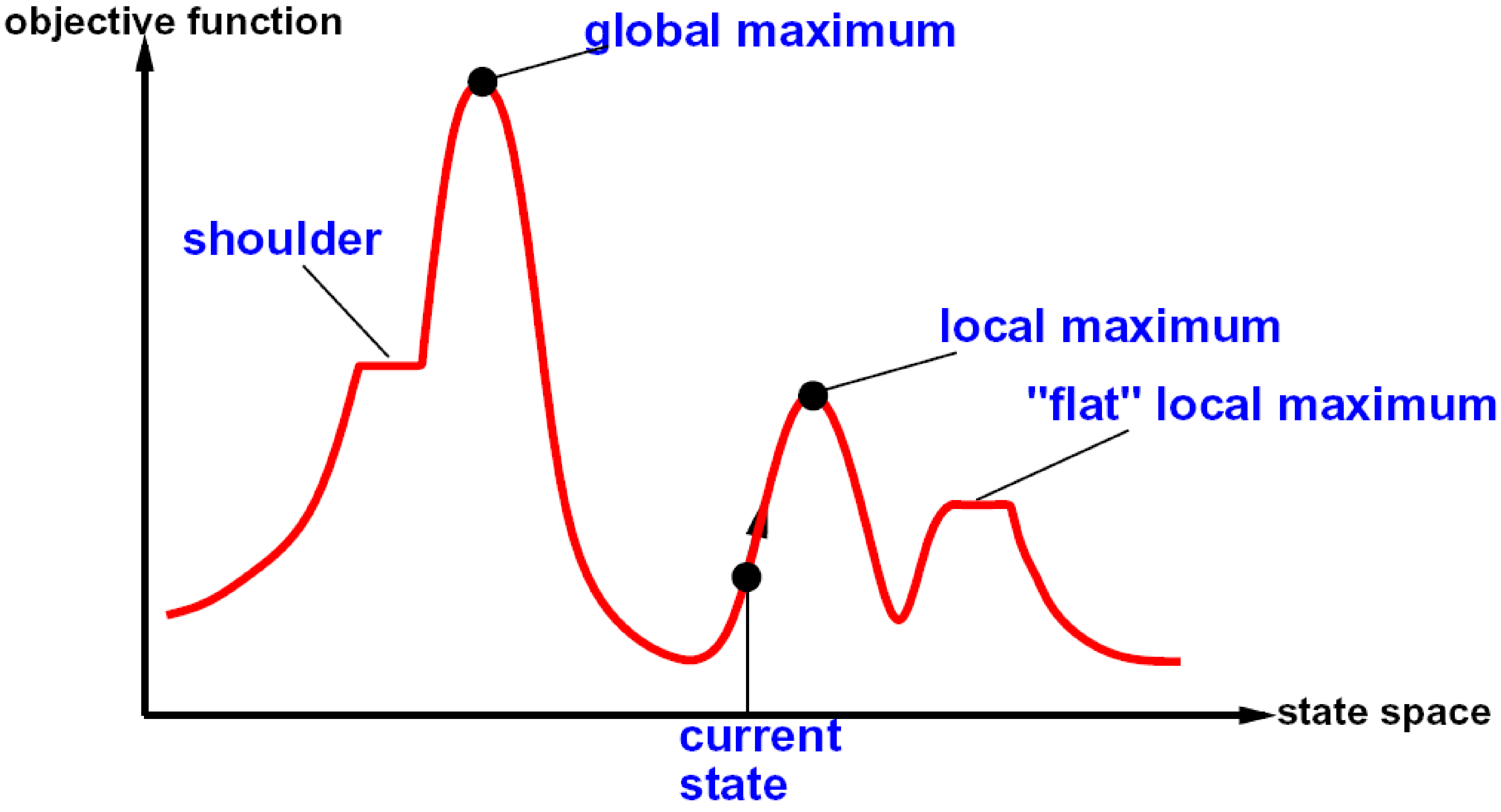
Hill Climbing

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

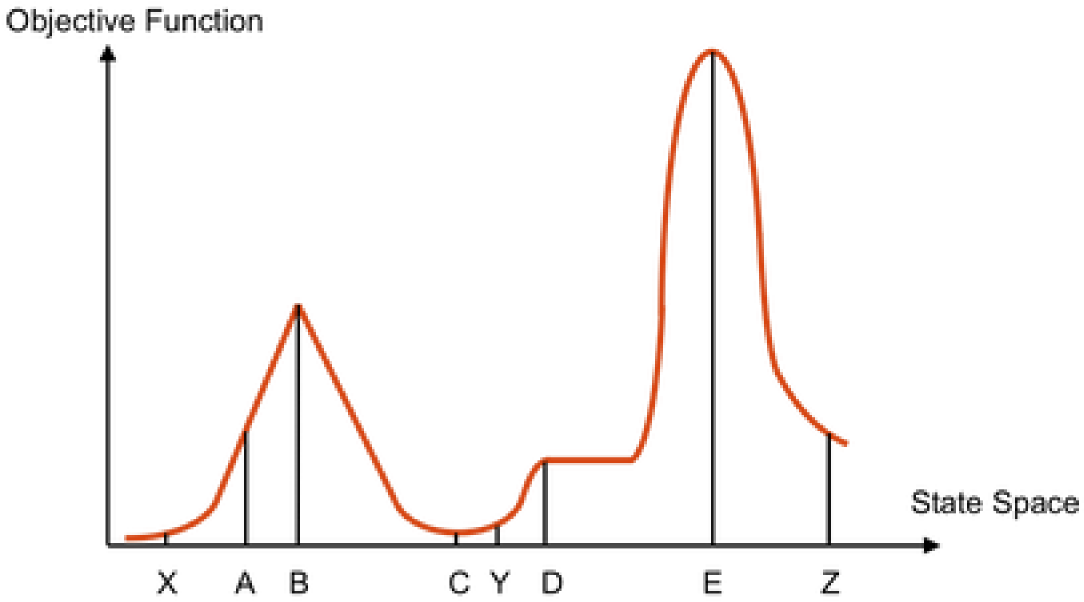
- What’s bad about this approach?
  - Complete?
  - Optimal?

- What’s good about it?

Hill Climbing Diagram



Hill Climbing Quiz



Starting from X, where do you end up ?

Starting from Y, where do you end up ?

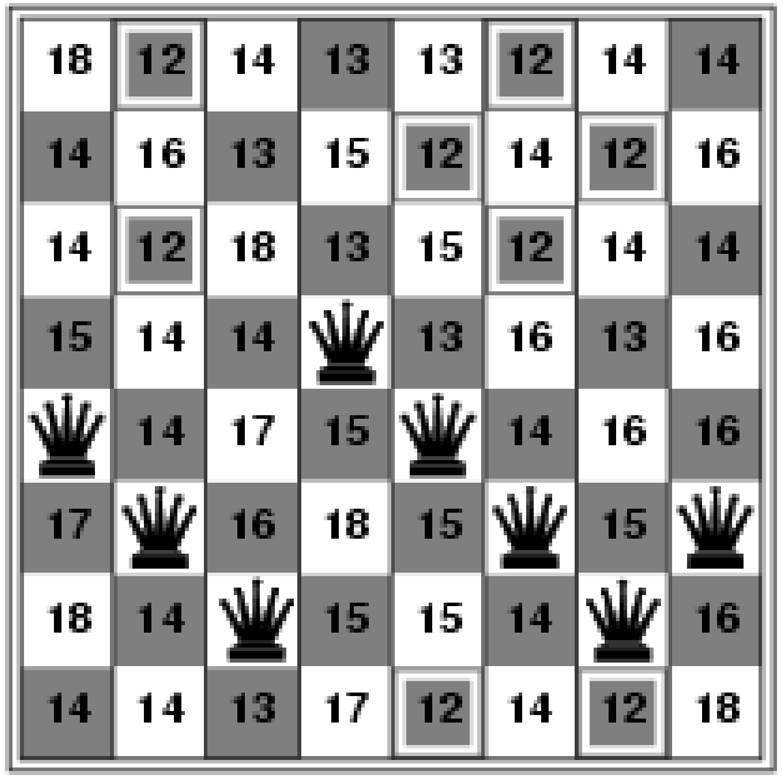
Starting from Z, where do you end up ?

Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

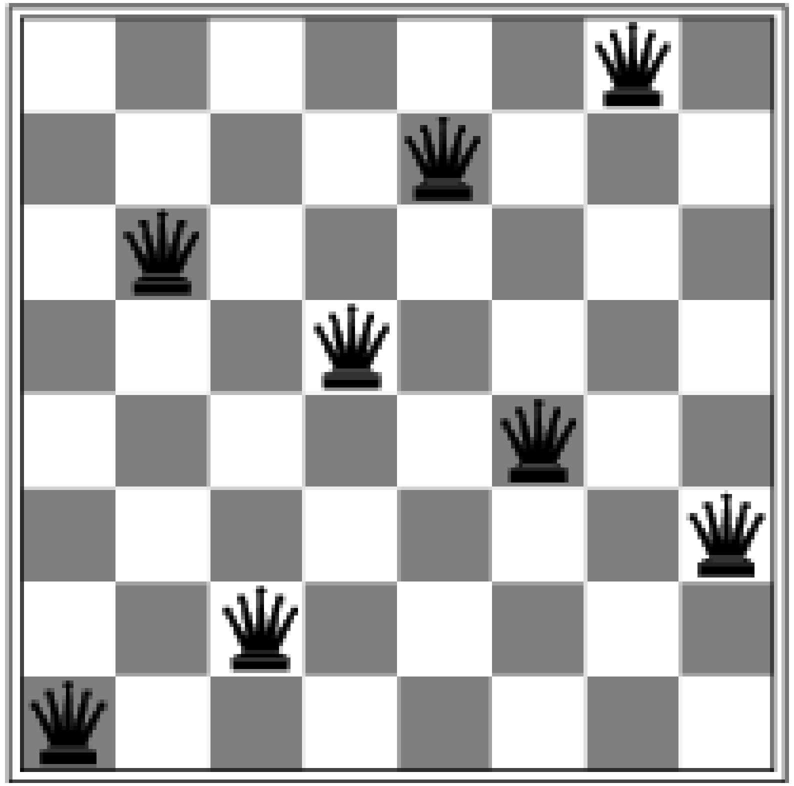
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node
    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Hill-climbing search: 8-queens problem



- *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

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

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** ∞ **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 $e^{\Delta E/T}$

Simulated Annealing

- Theoretical guarantee:
  - Stationary distribution: $$p(x) \propto e^{\frac{E(x)}{kT}}$$
  - If T decreased slowly enough, will converge to optimal state!
- Is this an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - People think hard about *ridge operators* which let you jump around the space in better ways

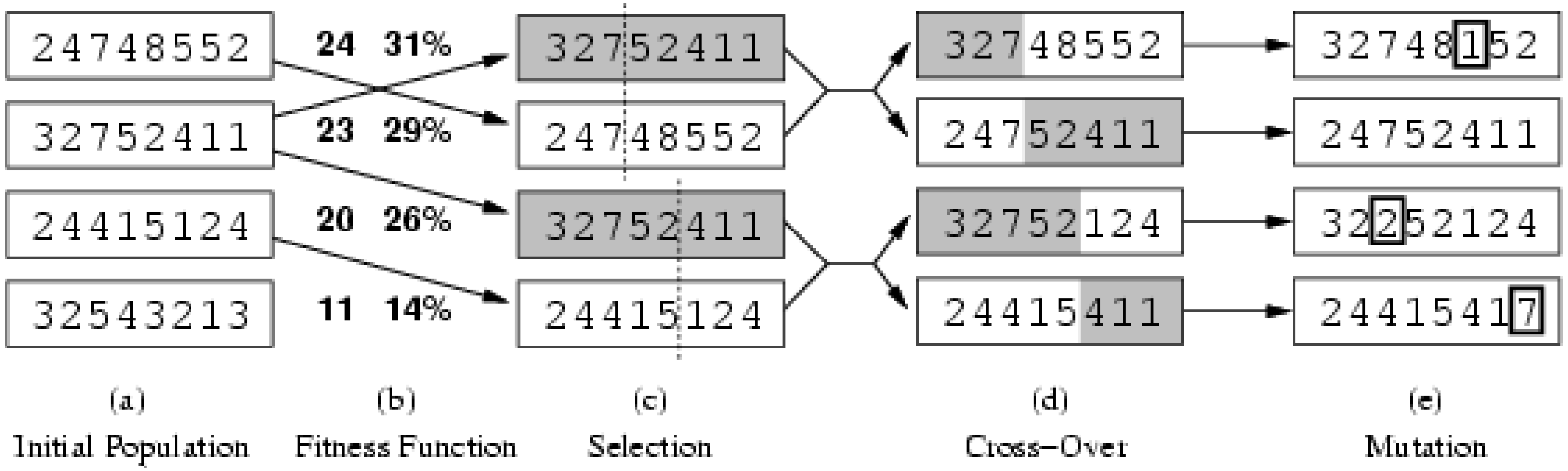
Local beam search

- 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

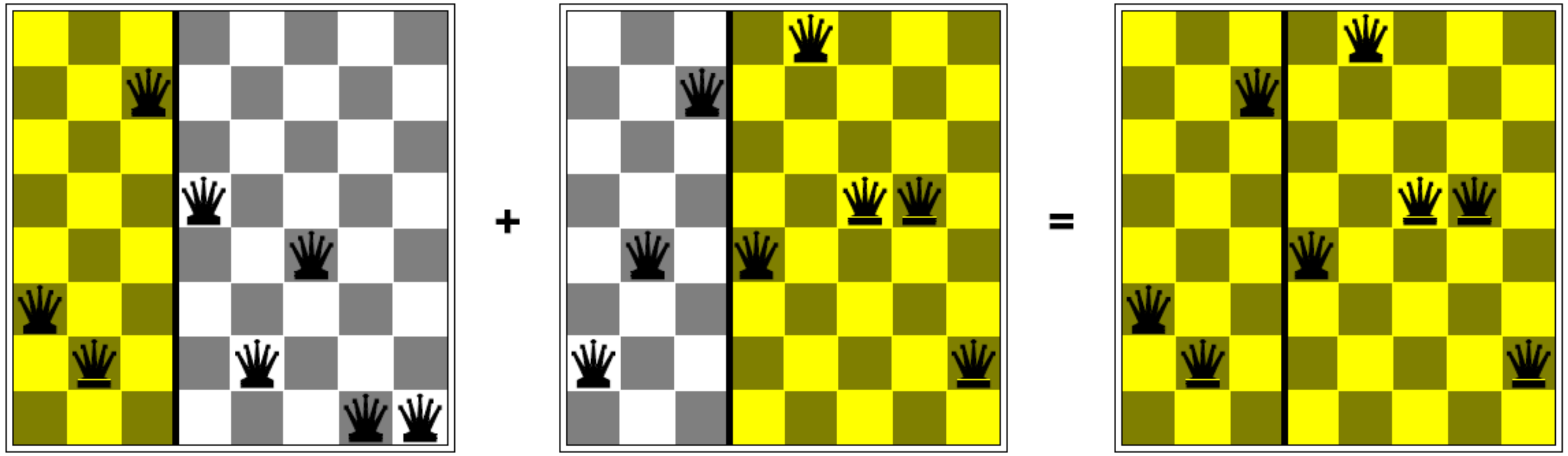
- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next 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

Example: N-Queens



Consent

Data for this course is taken from several books specifically AIMA by Russel Norvig, slides and already similar course taught in other universities specifically MIT, UC-Berkley, and Stanford and is the sole property of the respective owner. The copyright infringement is not intended, and this material is solely used for the academic purpose or as a teaching material.This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.