Introduction to Artificial Intelligence

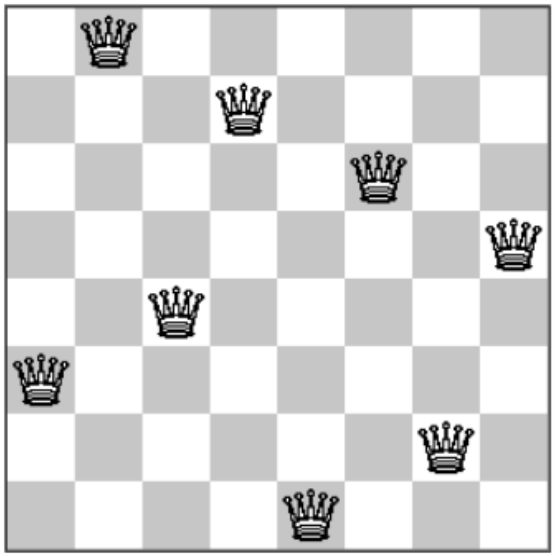
Lecture: Local Search

Outline

- Optimization Problems
- Hill-climbing Search
- Simulated Annealing Search
- Local Beam Search
- Genetic Algorithm

Global search algorithms

- Global search: explore search spaces systematically
  - Keep one or more paths in memory and record which alternatives have been explored at each point along the path
- Many problems do not fit the “standard” search model
  - The final configuration matters, not the order in which it is formed.
  - No “goal test” and no “path cost”



Optimization problems

- The optimization problems find the best state according to an objective function
  - E.g., the Knight’s tour, TSP, scheduling, integrated-circuit design, factory-floor layout, automatic programming, vehicle routing, etc.

Local search algorithms

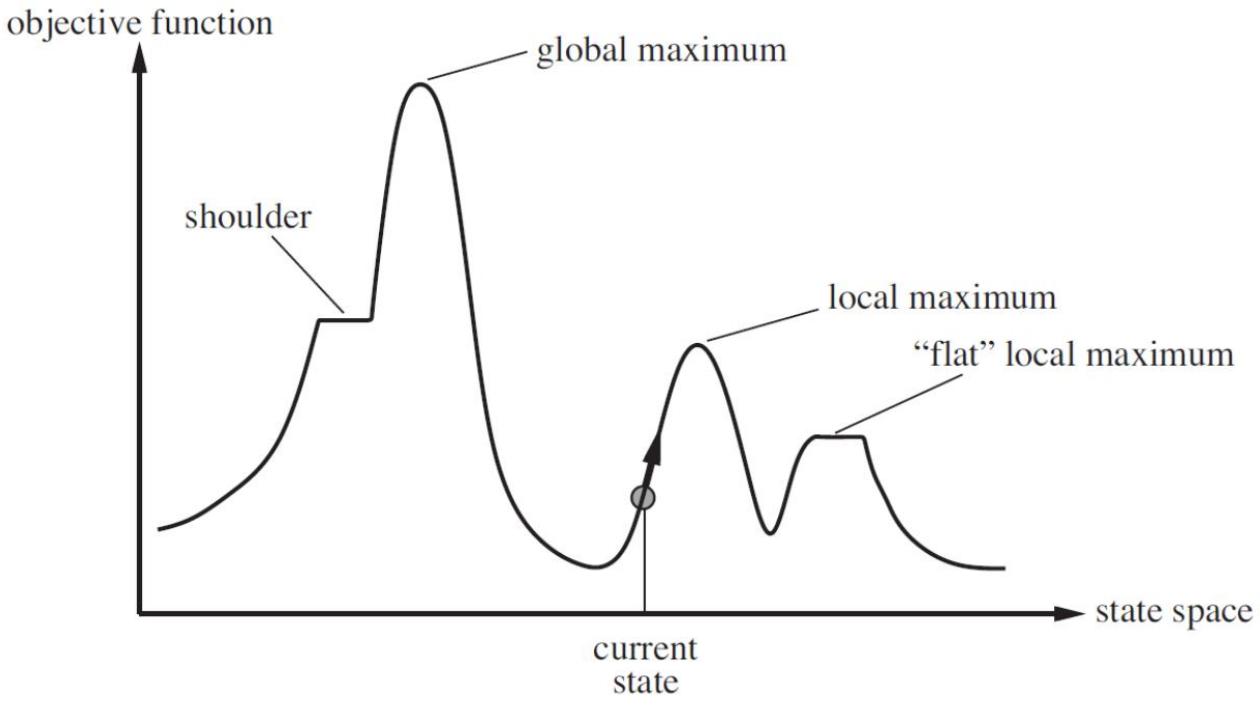
- Operate using a single current node and generally move only to neighbors of that node
- Not systematic: the paths followed by the search are typically not retained.
- Use very little memory: usually a constant amount
- Often able to find reasonable solutions in large or infinite (continuous) state spaces.

Local search and Optimization

- “Pure optimization” problems
  - All states have an objective function
  - Goal is to find state with max (or min) objective value
- Local search can do quite well on these problems.

State-space landscape

- A landscape has both “location” and “elevation”: location state, elevation heuristic cost or objective function
- A complete local search algorithm finds a goal if one exists.
- An optimal local search algorithm finds a global extremum



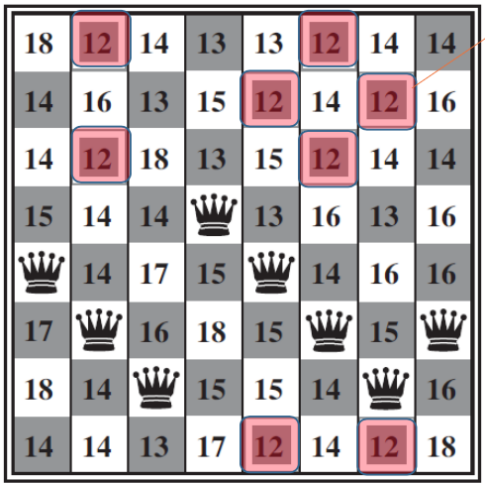
Hill-climbing Search

- A loop that continually moves in the direction of increasing value and terminates when it reaches a “peak”.
- Not look ahead beyond the immediate neighbors of the current state.
- No search tree maintained, only the state and the objective function’s value for the current node recorded.

```
function HILL-CLIMBING(problem) returns a local maximum
    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

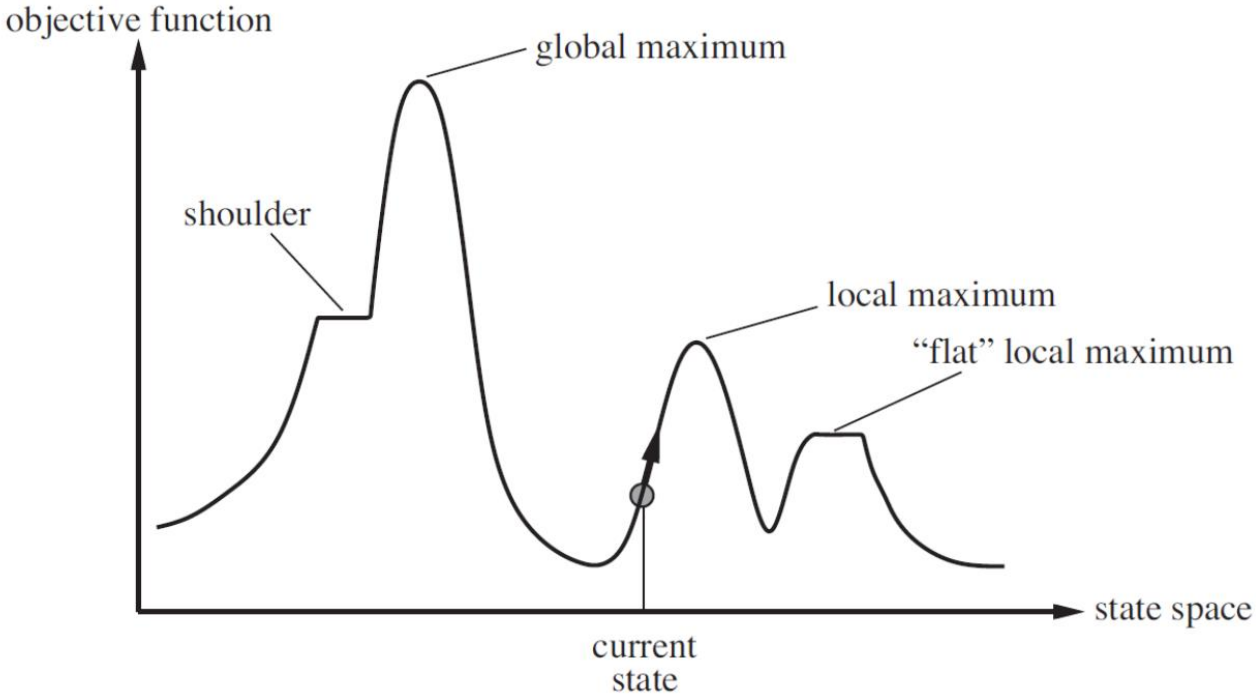
Hill-climbing: 8-queens problem

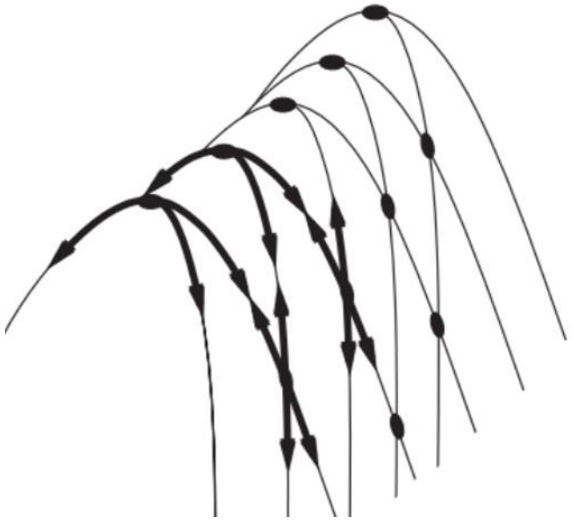
- Complete-state formulation
  - All 8 queens on the board, one per column
- Successor function
  - Move a single queen to another square in the same column
  - 8×7=56 successors
- Heuristic cost function h(n)
  - The number of pairs of queens that are ATTACKING each other, either directly or indirectly
  - Global minimum has h(n) = 0



Hill-climbing Search

- Hill climbing is often suboptimal, due to **local maxima**, **ridges** and **plateau**



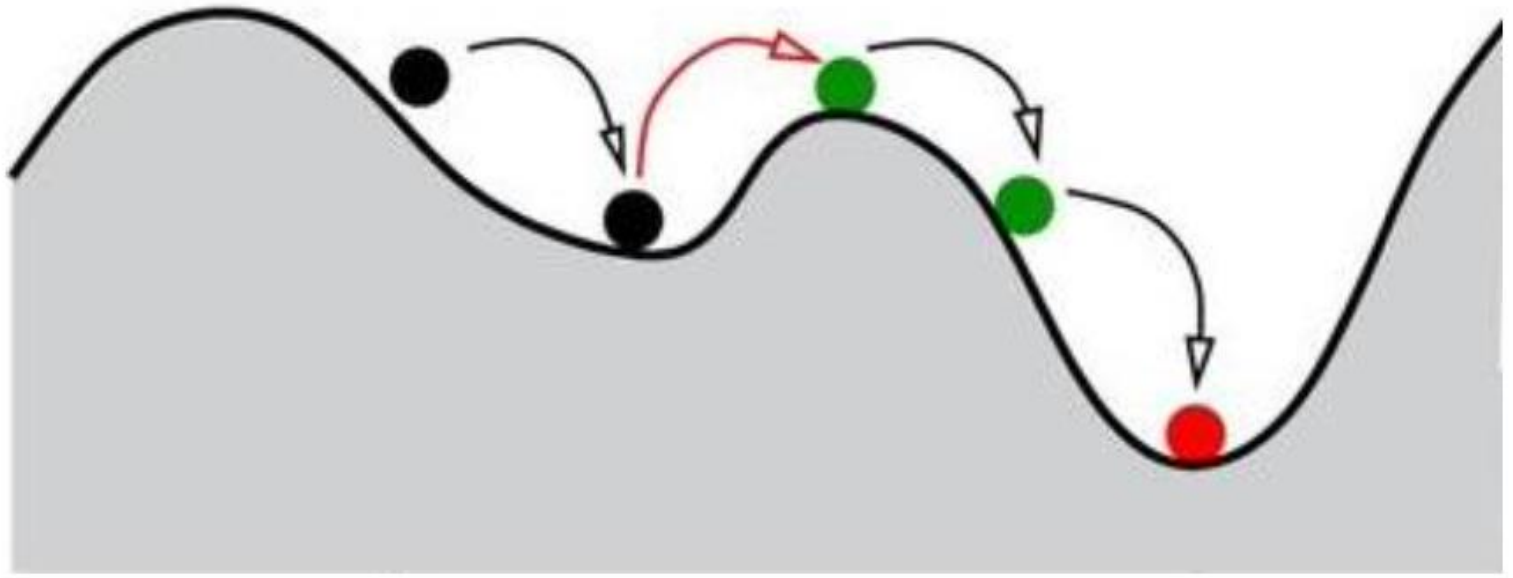


Hill-climbing Search Variants

- Stochastic hill climbing:
  - choose at random from among the uphill moves with a probability of selection varied with the moves' steepness
- First-choice hill climbing:
  - generate successors randomly until one is generated that is better than the current state
- Random-restart hill climbing:
  - A series of hill-climbing searches from randomly generated initial states until a goal is found

Simulated Annealing

- Combine hill climbing with a random walk in some way that yields both efficiency and completeness
- Shake hard (i.e., at a high temperature) and then gradually reduce the intensity of shaking (i.e., lower the temperature)



Simulated Annealing

function SIMULATED-ANNEALING(problem, schedule)

returns a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

current ← MAKE-NODE(problem.INITIAL-STATE)

for t = 1 to ∞ do

T ← *schedule*(t)

if T = 0 **then return** current

next ← a randomly selected successor of current

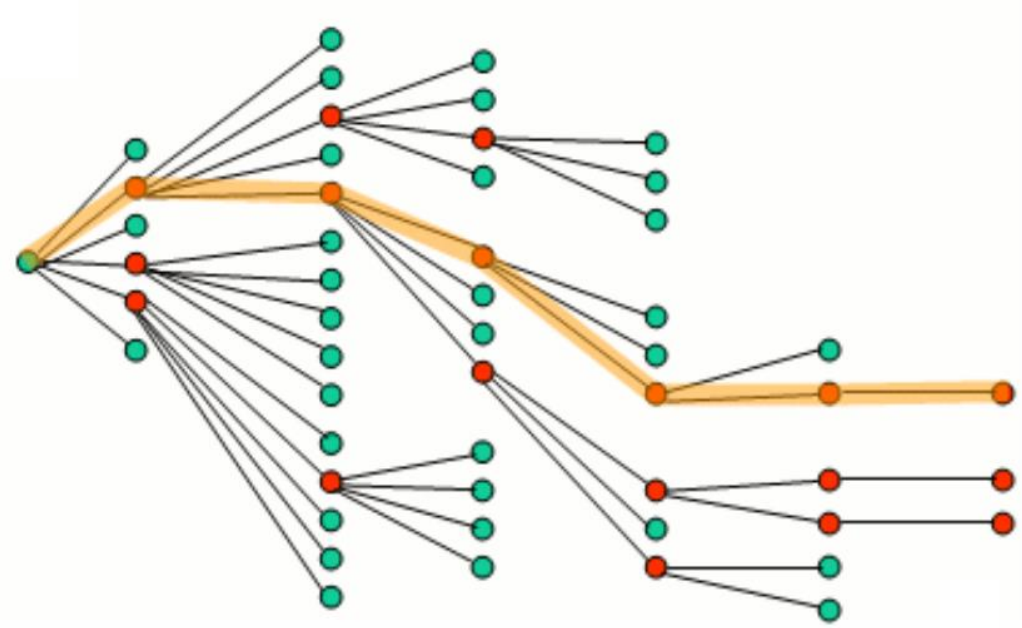
ΔE ← next.VALUE – current.VALUE

if ΔE > 0 **then** current ← next

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

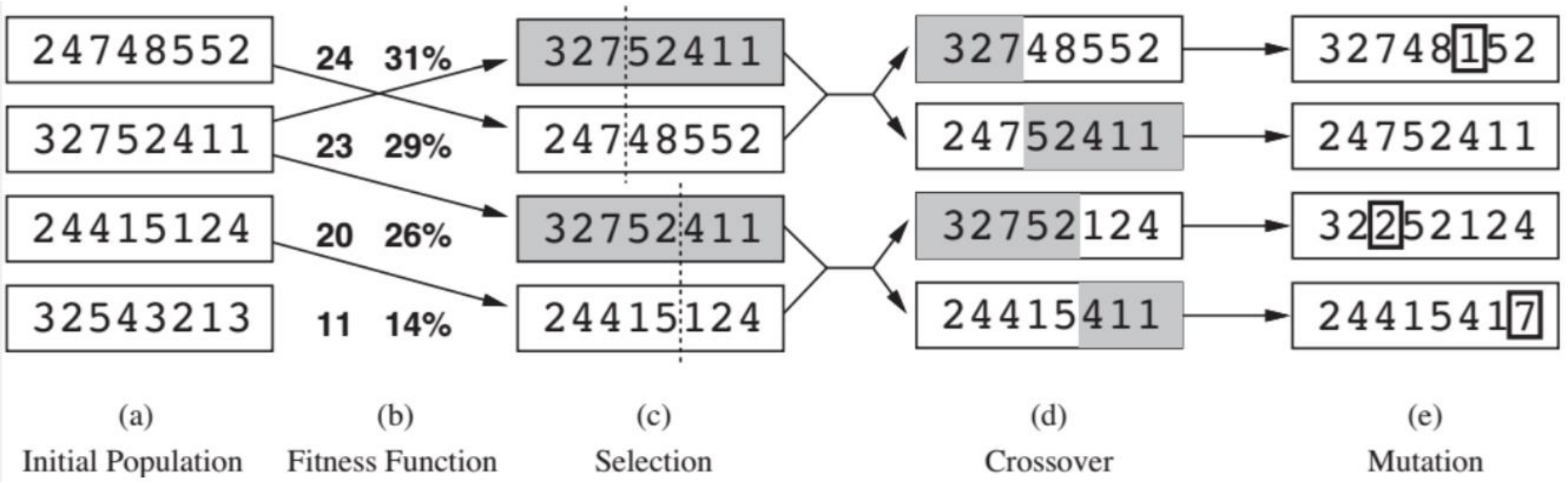
Local Beam Search

- Keep track of k states rather than just one
- Begin with k randomly generated states
- At each step, all successors of all k states are generated
- If the goal is not found, select the k best successors from the complete list and repeat



Genetic Algorithms

- A variant of stochastic beam search
- Successor states are generated by combining two parent states rather than by modifying a single state
- The reproduction are sexual rather than asexual



Genetic Algorithms

- Population: a set of k randomly generated states that GA begins with
- Fitness function: an objective function that rates each state
  - Higher values for better state
  - E.g., 8-queens: the number of non-attacking pairs of queens (min = 0, max = 8×7/2 = 28)
- Produce the next generation by “simulated evolution”
  - Random selection
  - Crossover
  - Random mutation
- Each state, or individual, is represented as a string over a finite alphabet – most commonly, a string of 0s and 1s.

Genetic Algorithms

- Pairs are selected at random for reproduction.
- The probability of being chosen for reproducing is directly proportional to the fitness score.
- One individual may be selected several times or not at all.

Genetic Algorithms

function GENETIC-ALGORITHM(population, FITNESS-FN)

returns an individual

inputs: *population*, a set of individuals

FITNESS-FN, a function that measures the fitness of an individual

repeat

new_population ← empty set

for i = 1 to SIZE(population) do

x ← RANDOM-SELECTION(population, FITNESS-FN)

y ← RANDOM-SELECTION(population, FITNESS-FN)

child ← REPRODUCE(x , y)

if (small random probability) then child ← MUTATE(child)

add child to new_population

population ← new_population

until some individual is fit enough, or enough time has elapsed

return the best individual in population, according to FITNESS-FN

Genetic Algorithms

function REPRODUCE(x , y)

returns an individual

inputs: x , y, parent individuals

    n ← LENGTH(x)

    c ← random number from 1 to n

    return APPEND(SUBSTRING(x , 1, c), SUBSTRING(y, c + 1, n))

Homework

- Conduct homework in the given notebook.

References

- Stuart Russell and Peter Norvig. 2009. Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.
- Lê Hoài Bắc, Tô Hoài Việt. 2014. Giáo trình Cơ sở Trí tuệ nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.
- Nguyễn Ngọc Thảo, Nguyễn Hải Minh. 2020. Bài giảng Cơ sở Trí tuệ Nhân tạo. Khoa Công nghệ Thông tin. Trường ĐH Khoa học Tự nhiên, ĐHQG-HCM.