Local search algorithms

Chapter 4

outline

- Hill – climbing
- Simulated annealing
- Genetic algorithms

Local search algorithms

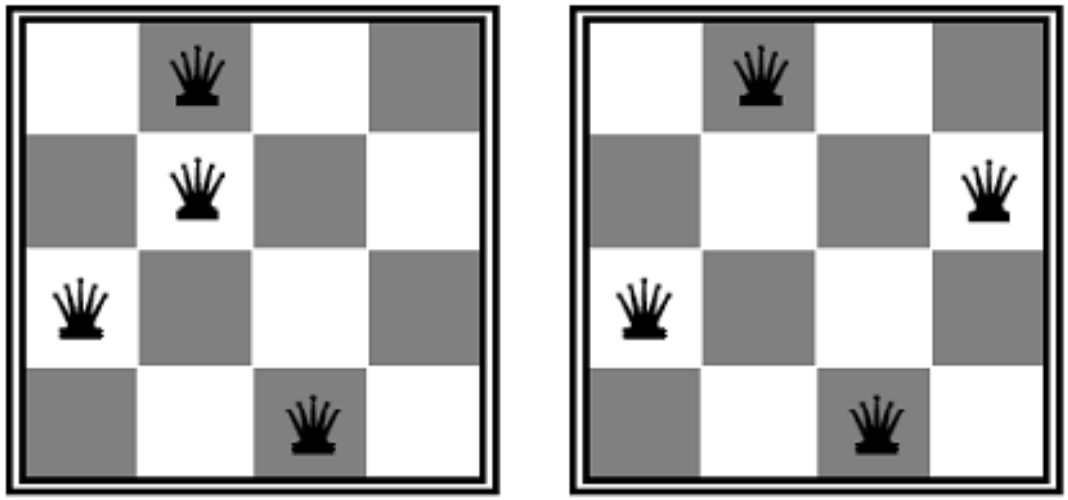
- Some types of search problems can be formulated in terms of **optimization**
  - Path is irrelevant; the **goal state** itself is a **solution**
  - **state space** = set of "complete" configueations
  - In such cases, can use iterative improvement algorithms; keep a single "current" state, try to improve it.
  - We have an **objective function** that tells us about the quality of a possible solution
    - we want to find a good solution by minimizing or maximizing the value of this function

Example: Travelling salesman problem

- Find the shortest tour connecting a given set of cities
- **State space**: all possible tours
- **Objective function**: length of tour

Eample: *n*-queens

- Put *n* queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- **State space:** all possible n-queen configurations
- What's the **objective function**?
  - Number of pairwise conflicts



Hill-climbing(gradient ascent/descent)

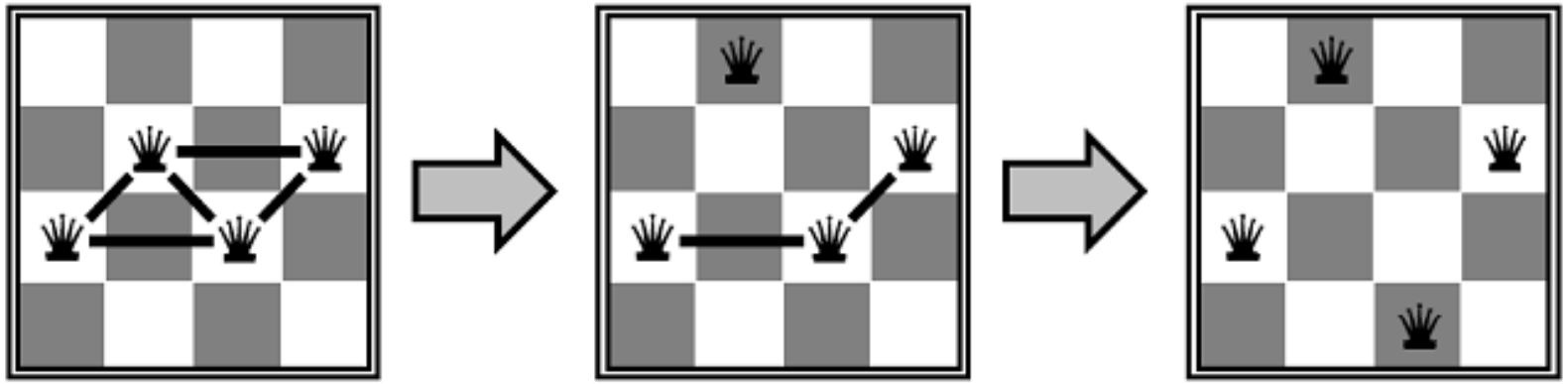
- Idea: keep a single"current"state and try to improve it.
- "Like climbing mount Everest in thick fog with amnesia"

```
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
   end
```

choose first better successor, randomly choose among better successors

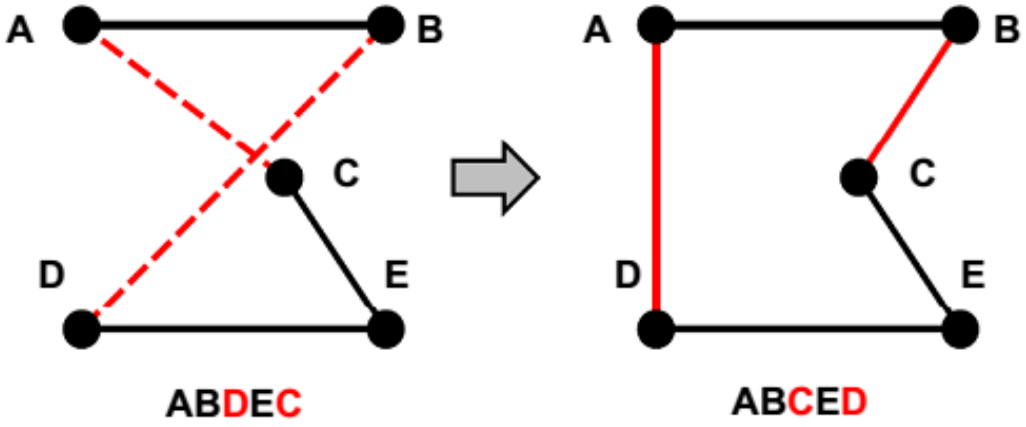
Eample: *n*-queens

- Put *n* queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- **State space:** all possible n-queen configurations
- **objective function**: Number of pairwise conflicts
- What's a possible local improvement strategy?
  - Move one queen within its column to reduce conflicts



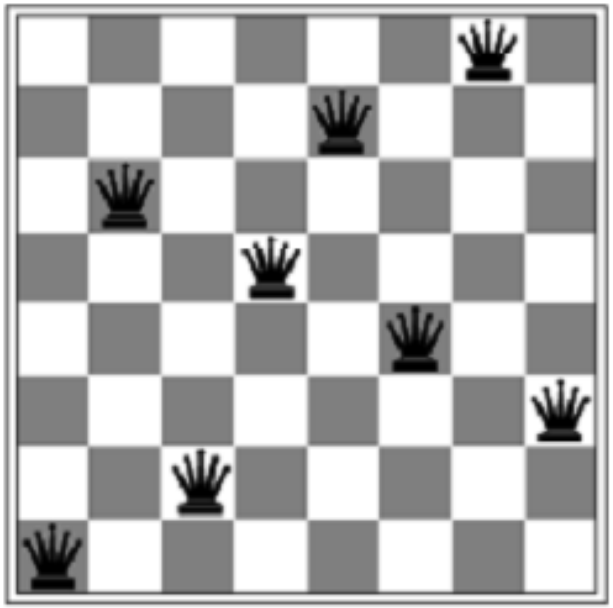
Example: Traveling Salesman Problem

- Find the shortest tour connectingn cities
- **State space:** all possible tours
- **Objective function:**length of tour
- What's a possible local improvement strategy?
  - Start with any complete tour, perform pairwise exchanges



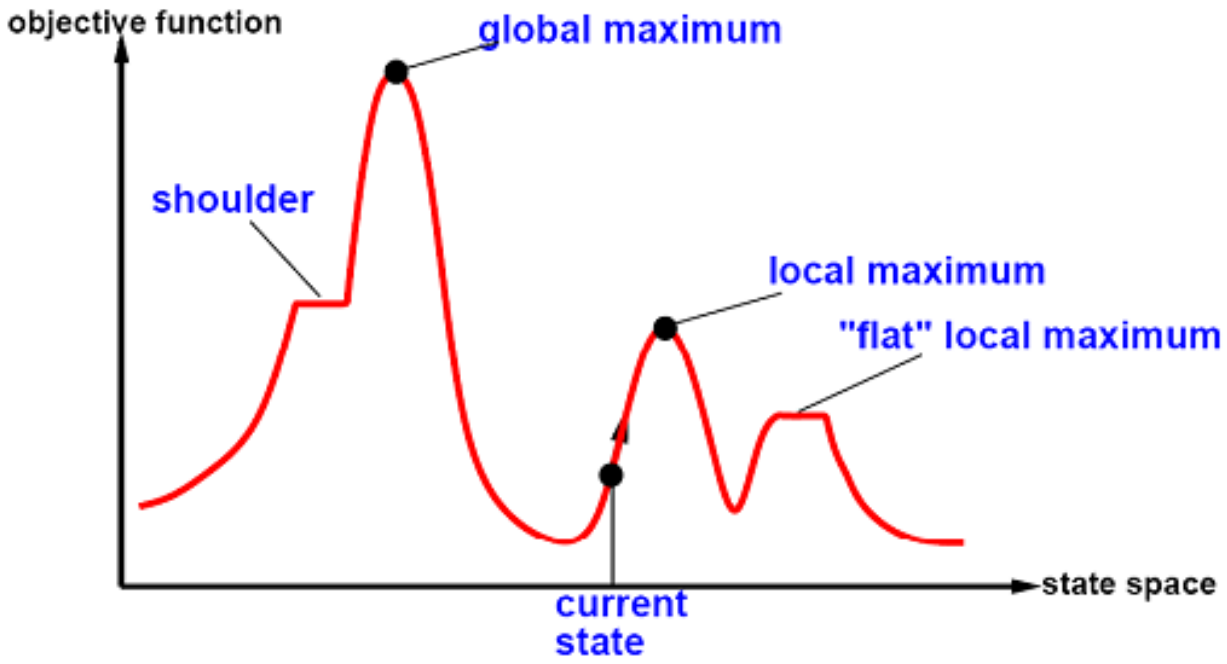
Hill-climbing search

- Is it complete/optimal?
  - No – can get stuck in local optima
  - Example: local optimum for the 8-queens problem



$h = 1$

Hill-climbing search



- How to escape local maxima?
  - Random restart hill-climbing
- What about “shoulders”?
- What about “plateaux”?

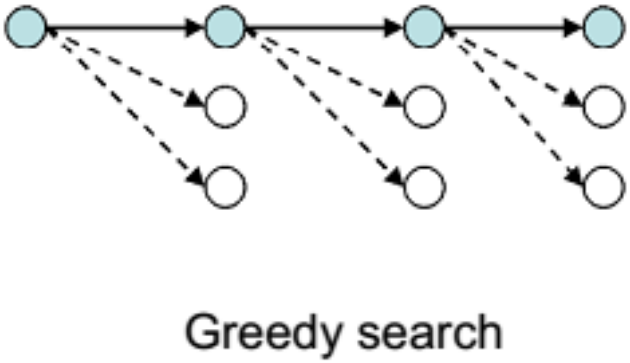
Simulated annealing

- Idea: escape local maxima by allowing some "bad"moves but gradually decrease their siza and frequency
- If temperature decreases slowly enough, then simulated annealing search will find a global optimum.

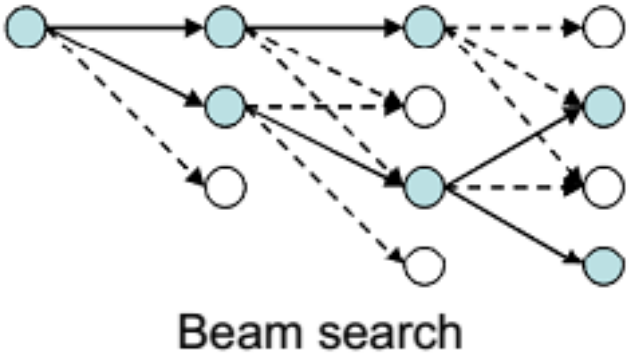
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps

  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^(Δ E/T)
```

Local beam search

- 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



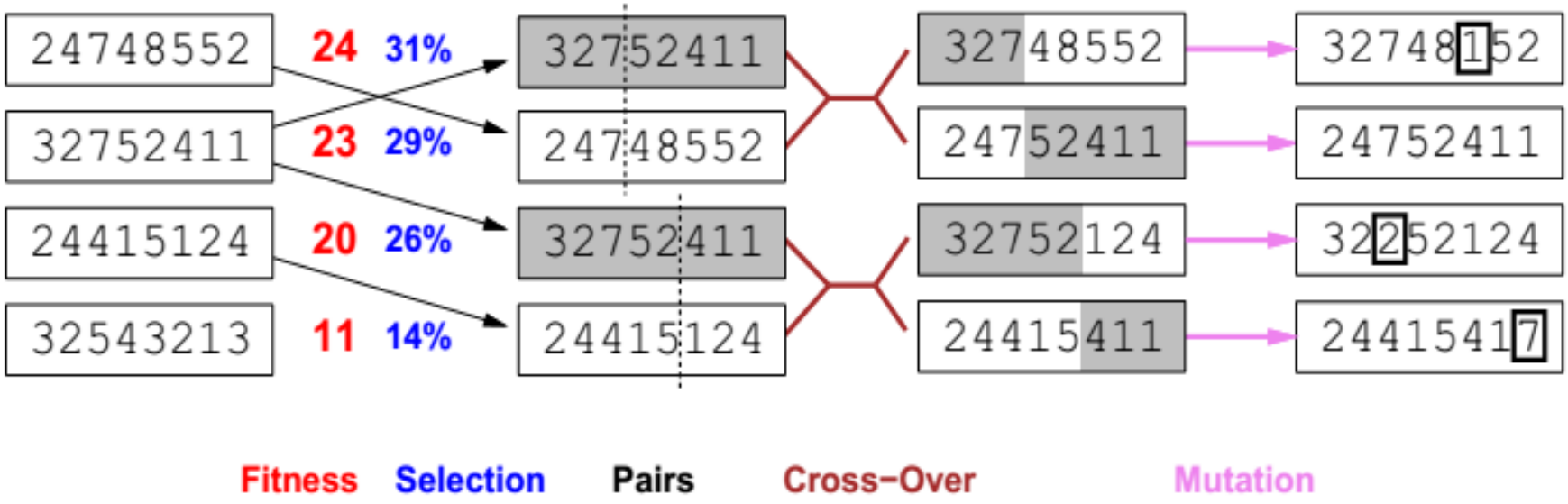
Greedy search



Beam search

Genetic algorithms

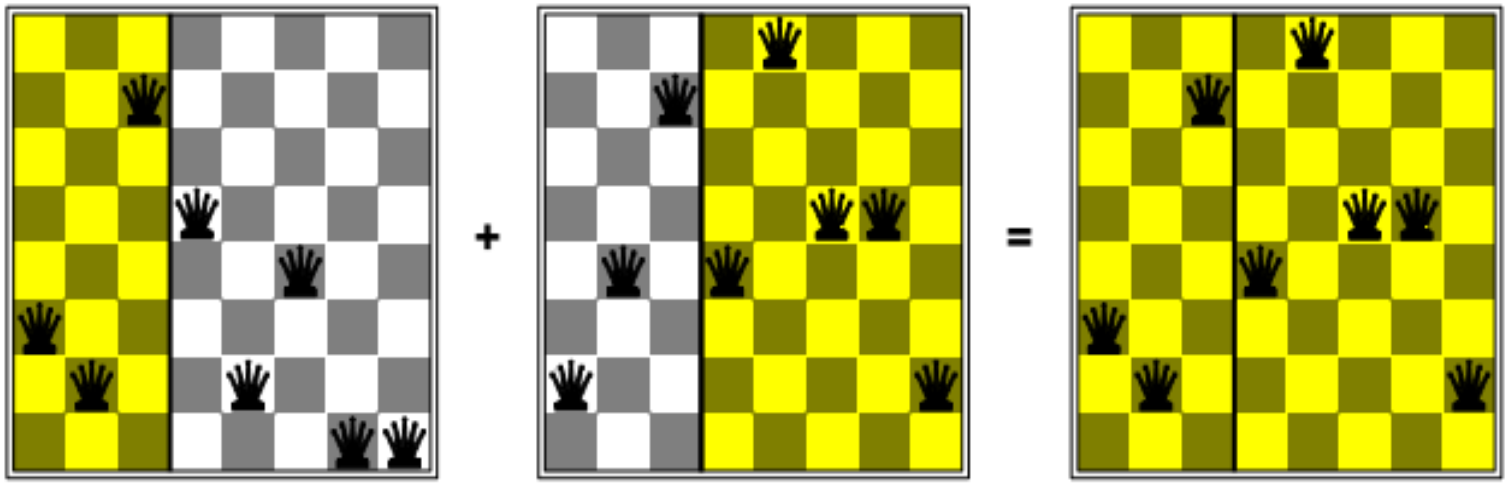
= stochastic local beam search + generate successors from **pairs** of states



Example: *n*-queens

- Gas require states encoded as string
- Crossover helps iff substrings are meaningful compunents

- Example: string for first state= 67247588



Genetic algorithms

```
function GENETIC-ALGORITHM( population, FITNESS-FN) returns an individual
    input: population, a set of individuals
           FITNESS-FN, a function that measures the fitness of an individual

    repeat
        new_population ← empty set
        loop for i from 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
```

Project???

- Solving 8-queens problem by genetic algorithm (4 group)
- Solving 8-puzzle problem by genetic algorithm (4 group)
- Solving 8-queens problem by A* search (4 group)
- solving 8-puzzle problem by A* search (4 group)
- explain GAs completely by giving an example (1 group)

End of chapter 4