Local search algorithms

Chapter 4, Sections 3–4

Outline

◇ Hill-climbing

◇ Simulated annealing

◇ Genetic algorithms (briefly)

Iterative improvement algorithms

In many optimization problems, **path** is irrelevant;
the goal state itself is the solution

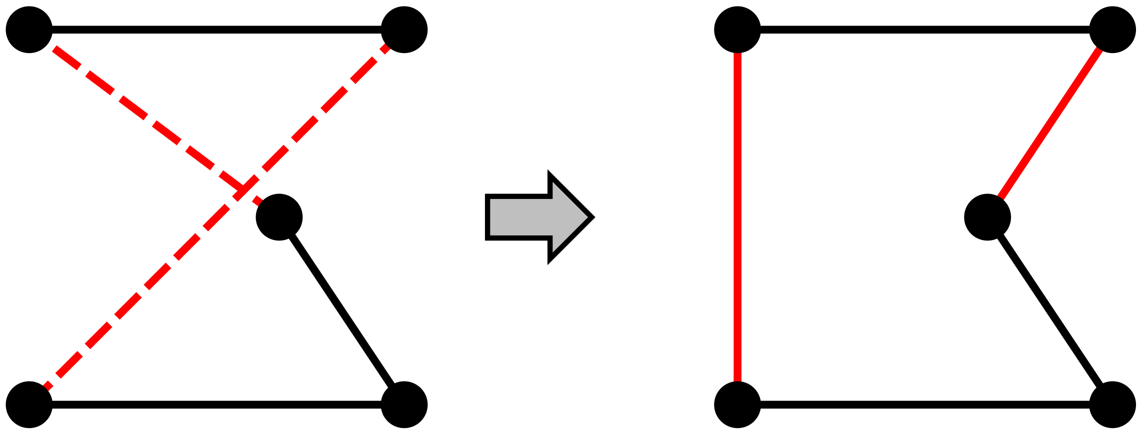
Then state space = set of "complete" configurations;
find **optimal** configuration, e.g., TSP
or, find configuration satisfying constraints, e.g., timetable

In such cases, can use iterative improvement algorithms;
keep a single "current" state, try to improve it

Constant space, suitable for online as well as offline search

Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

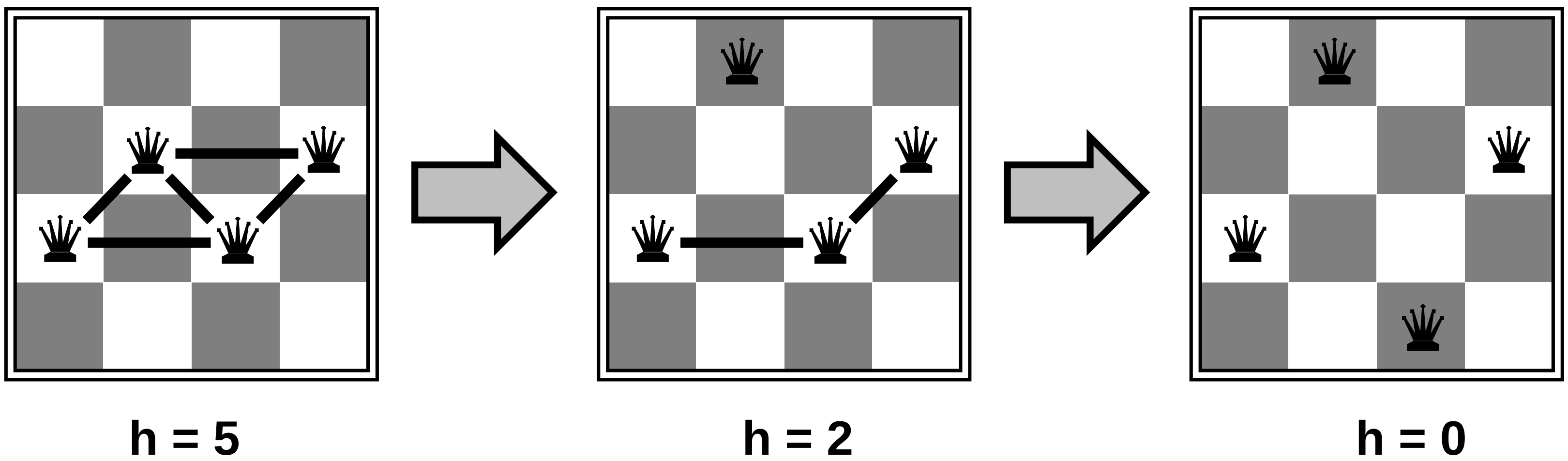


Variants of this approach get within 1% of optimal very quickly with thousands of cities

Example: n-queens

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

Move a queen to reduce number of conflicts



Almost always solves n-queens problems almost instantaneously for very large n, e.g., $n = 1 million$

Hill-climbing (or gradient ascent/descent)

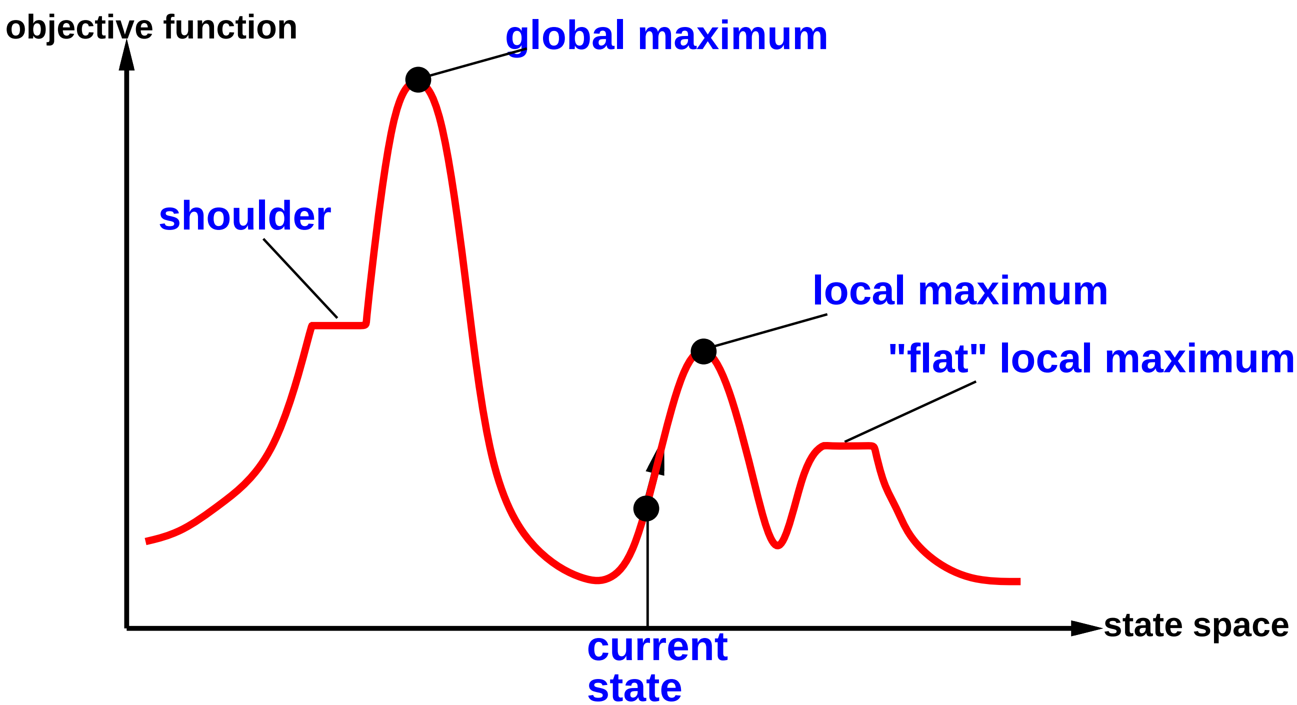
"Like climbing Everest in thick fog with amnesia"

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*
  end

Hill-climbing contd.

Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves 😈 escape from shoulders ☹ loop on flat maxima

Simulated annealing

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and 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)
```

Simulated annealing

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.

Local beam search

Idea: keep k states instead of 1; choose top k of all their successors

Searches that find good states recruit other searches to join them

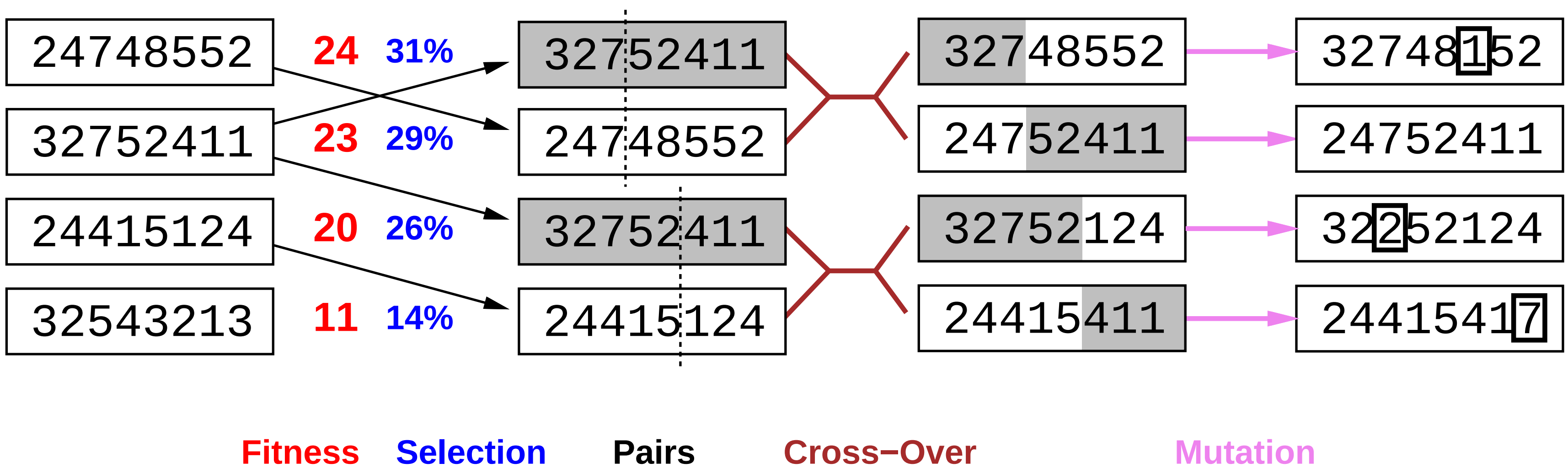
Problem: quite often, all k states end up on same local hill

Idea: choose k successors randomly, biased towards good ones

Observe the analogy to natural selection!

Genetic algorithms

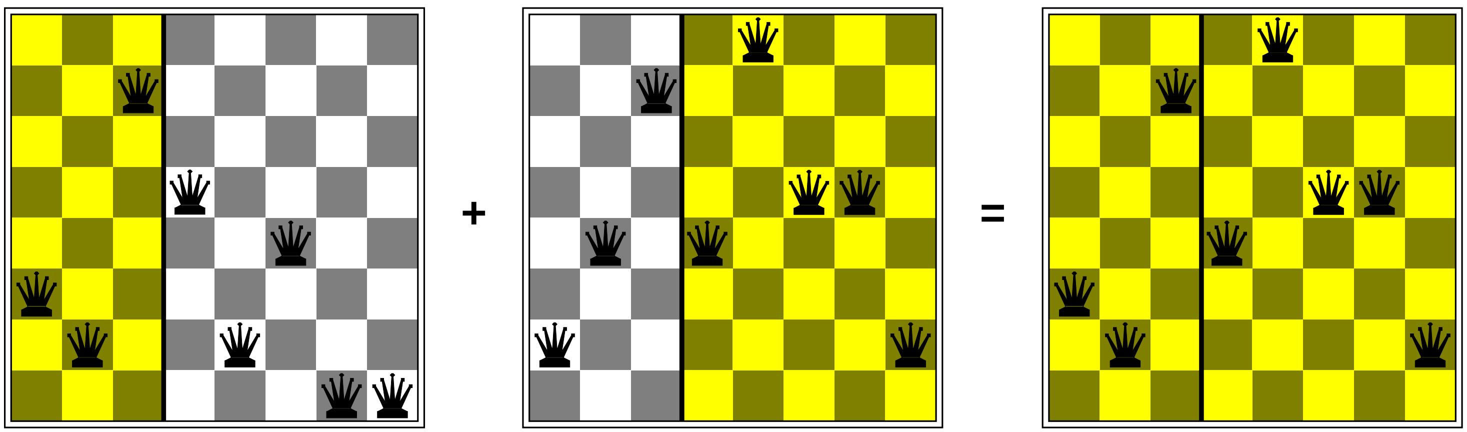
= stochastic local beam search + generate successors from **pairs** of states



Genetic algorithms contd.

GAs require states encoded as strings (GPs use programs)

Crossover helps **iff substrings are meaningful components**



GAs $\neq$ evolution.