Local search algorithms

Chapter 4, Sections 1–2

Review

Chapter 3: uninformed and informed searching were systematic algorithms for solving problems with following characteristics:

◇ Observable

◇ Deterministic

◇ Known environment

◇ Solution were sequence of actions

Outline

Assumptions relaxed: state space is not completely known.

◇ Hill-climbing

◇ Simulated annealing

◇ Genetic algorithms

◇ Local search in continuous spaces (briefly)

Local search

Local search: algorithms that perform local search in the state space, evaluating and modifying one or more current states rather than systematically exploring paths from an initial state.

♢ Operate using a single (or few) current node and gererally move only to neighbors of the node.

♢ Paths followed are not retained

♢ No goal test and path cost

♢ Use very little memory usage and can find reasonable solutions in large or infinte state space.

♢ Suitable form for problems in which all that matters is the solution state, not the path cost to reach it. EX) Pure optimization problems, in which the aim is to find the best state according to an objective function.

Iterative improvement algorithms

In many optimization problems, **path** is irrelevant;
the goal state itself is the **solution** according to an objective function

Then state space = set of complete-state formulation configurations, i.e. configuration of all atoms in proteins;

find **optimal** configuration, e.g., 8-queens problem

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

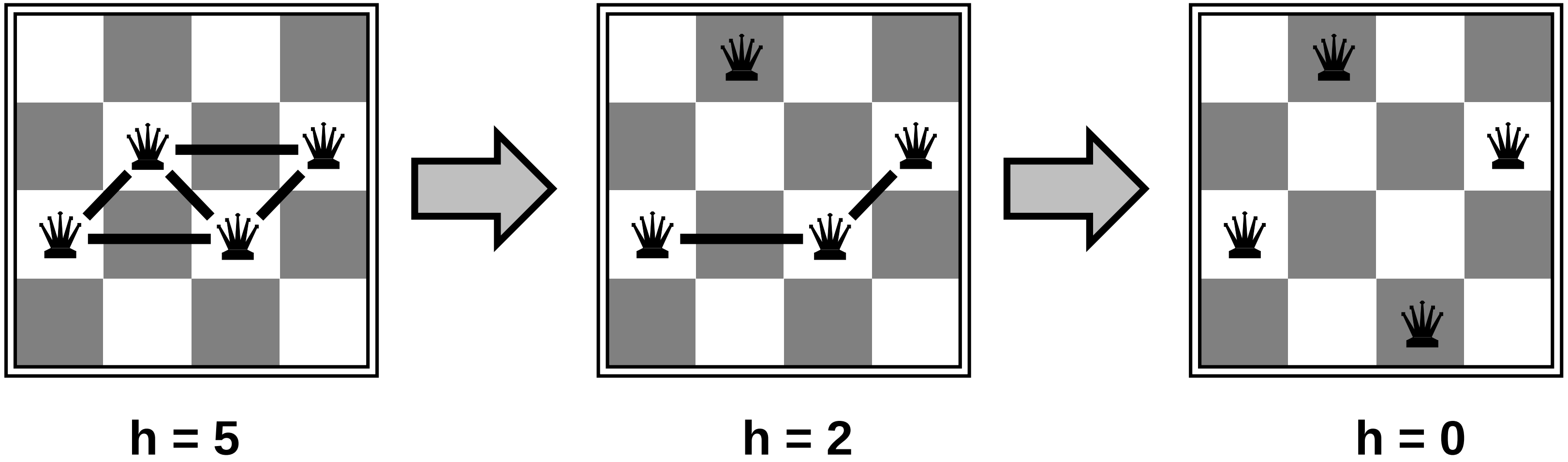
List of combinatorial optimization algorithms: genetic algorithms, simulated annealing, Tabu search, ant colony optimization, river formation dynamics (see swarm intelligence) and the cross entropy method.

Constant space, suitable for online as well as offline search

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, our heuristic cost function $h(n)$ or objective function



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

Hill-climbing (or gradient ascent/descent)

Also called greedy local search

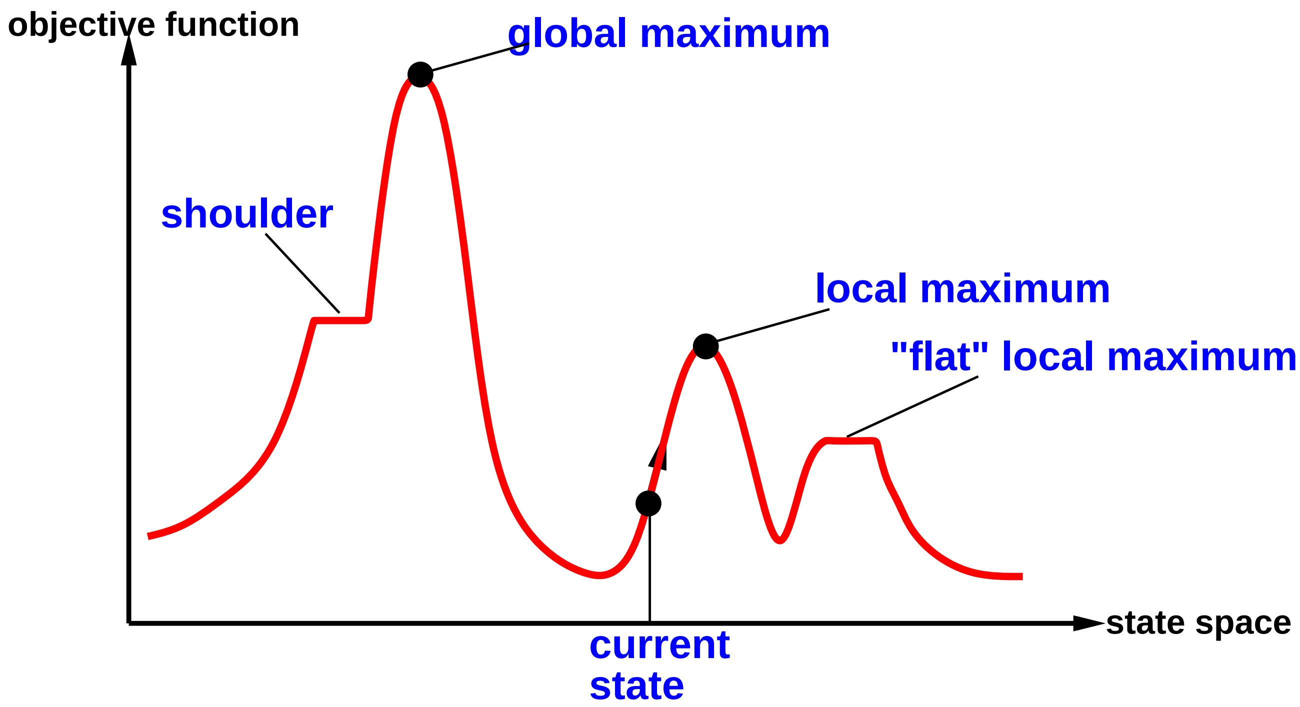
"Like climbing Everest in thick fog with amnesia"

```
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
    end
```

steepest-ascent version

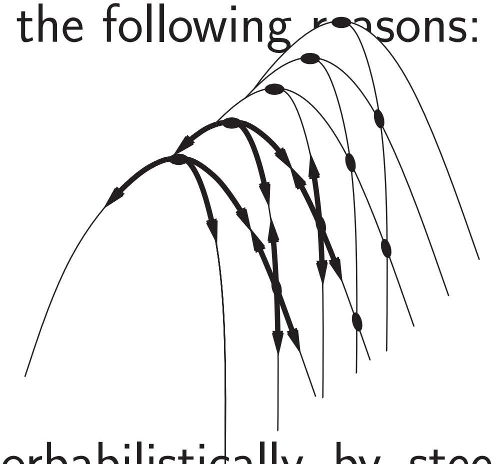
Hill-climbing contd.

Useful to consider state space landscape



Hill-climbing contd.

Unfortunately, hill climbing often get stuck for the following reasons:



- ♢ Local maxima
- ♢ Ridges
- ♢ Plateaux

Stochastic hill climbing choose at random(porbabilistically by steepness) from among the uphill moves – coverges more slowely but can find better solutions. Still gets stuck in the local minimal/maximal

Random-restart hill climbing try many restart from different start states and choose the best one - trivially complete as probability approaching 1.

NP-hard problems typically have an exponential number of local maximas. However, reasonalby good local maximum can often be found after a small number of restarts.

Simulated annealing

Idea: Pick a random move,

if the move that improves the objective function always accept the move,

otherwise accept the “bad” move with some probabilty less than 1.

Gradually decreasing “bad” move frequency.

Probability decreases exponentially with the “badness” of the move and as the “temperature” T goes down

If the $schedule$ lowers the T slowly enough, the algorithm will fund a global optimum with probabilty approaching 1.

Simulated annealing algorithm

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 $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[t]
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta\ E/T}$

Properties of simulated annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution* (= Gibbs Distribution): Distribution function f(E) probability that a particle is in energy state E.

$$f(E(x)) = \frac{1}{Ae^{\frac{E(x)}{kT}}}$$

T decreased slowly enough $\Longrightarrow$ always reach best state x^*

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

Widely used in *ab initio* protein structure prediction, Very-large-scale integration (VLSI) for creating integrated circuits layout, airline scheduling, etc.

Local beam search

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

Not the same as k searches run in parallel!
Searches that find good states recruit other searches to join them

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

Stochastic beam search:

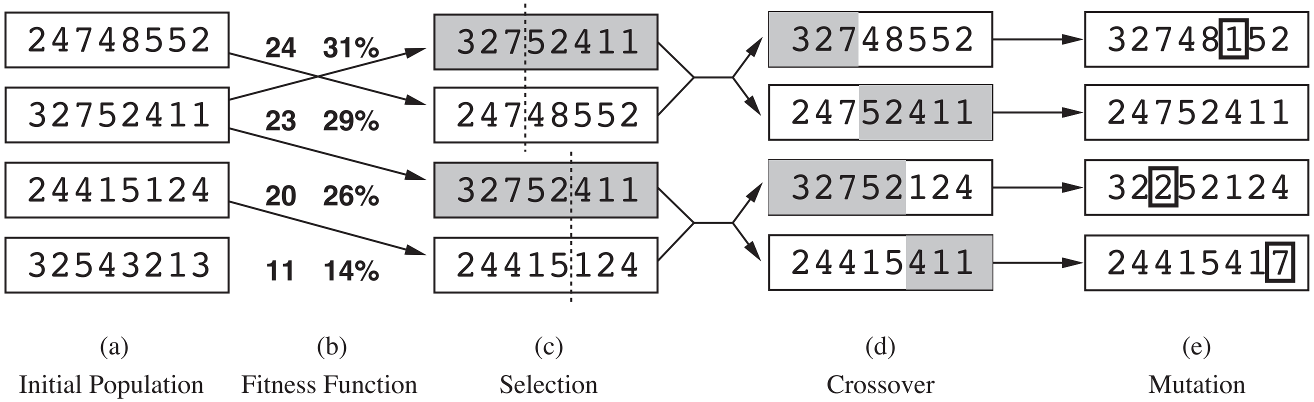
Idea: choose k successors randomly, biased towards good ones

Observe the close analogy to natural selection!

Genetic algorithms

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

GAs are **search** and **optimization** techniques based on Darwin's Principle of Natural Selection.



Population: k randomly generated states
Indivisual: states prepresented as a string of finite alphabets
Fitness function: objective function (higher values for better states)
Crossover point: random position in the state string
Mutation: each location in a state string is subjected to random mutation with small independent probability

Genetic algorithms contd.

```
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
```

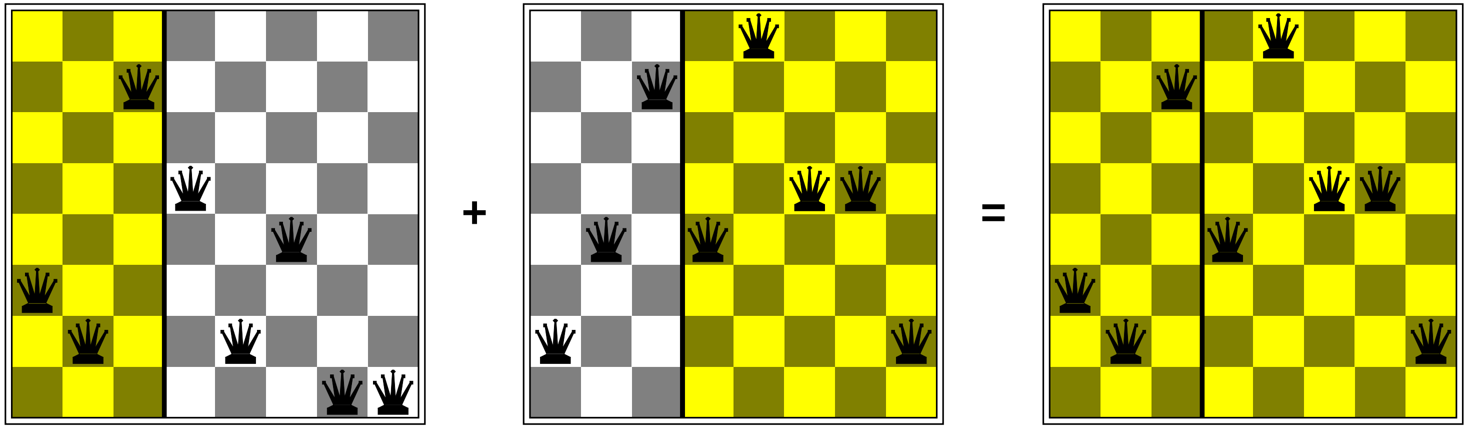
```
function REPRODUCE(x, y) returns an individual
   inputs: x, y, parent individuals

   n ← LENGTH(x); c ← random number from 1 to n
   return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Figure 4.8 A genetic algorithm. The algorithm is the same as the one diagrammed in Figure **??**, with one variation: in this more popular version, each mating of two parents produces only one offspring, not two.

Genetic algorithms contd.

GAs require states encoded as strings (GPs use programs)

Crossover helps **iff substrings are meaningful components**



GAs $\neq$ evolution: e.g., real genes encode replication machinery!

Local search in continuous state spaces

Suppose we want to site three airports in Romania:
- 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)
- objective function $f(x_1, y_2, x_2, y_2, x_3, y_3) =$
  sum of squared distances from each city to nearest airport

Discretization methods turn continuous space into discrete space,
e.g., empirical gradient considers $\pm\delta$ change in each coordinate

Gradient methods compute

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

to increase/reduce f, e.g., by $\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$

Sometimes can solve for $\nabla f(\mathbf{x}) = 0$ exactly (e.g., with one city).
Newton–Raphson (1664, 1690) iterates $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$
to solve $\nabla f(\mathbf{x}) = 0$, where $\mathbf{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$