Local search algorithms

Chapter 4, Sections 3–4

Outline

♦ Hill-climbing
♦ Simulated annealing
♦ Genetic algorithms (briefly)
♦ Local search in continuous spaces (very 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

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
```

Chapter 4, Sections 3–4

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

Chapter 4, Sections 3–4

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)
```

Chapter 4, Sections 3–4

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
Idea: choose k successors randomly, biased towards good ones
Observe the close analogy to natural selection!

Chapter 4, Sections 3–4
Genetic algorithms

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

Fitness Selection Pairs Cross-Over Mutation

24748552 32752411 32748552 32744512
32752411 24748552 24752411 24752411
24415124 32752411 3275124 32752411
32543213 24415124 24415411 24415411

Continental state spaces

Suppose we want to site one airport in Romania:
– 2-D state space defined by \((x, y)\)
– objective function \(f(x, y) = \sum (x_i - x)^2 + (y_i - y)^2\)

\[
\frac{\partial f}{\partial x} = -2\sum (x_i - x)
\]

\[
\frac{\partial f}{\partial y} = -2\sum (y_i - y)
\]

Genetic algorithms contd.

GAs require states encoded as strings (GPs use programs)

Crossover helps iff substrings are meaningful components

Continuous state spaces

Suppose we want to site one airport in Romania:
– 2-D state space defined by \((x, y)\)
– objective function \(f(x, y) = \sum (x_i - x)^2 + (y_i - y)^2\)

Discretization methods turn continuous space into discrete space, e.g., empirical gradient considers ±δ change in each coordinate
Continuous state spaces

Suppose we want to site three airports in Romania:
- 6-D state space defined by \((x_1, y_1), (x_2, y_2), (x_3, y_3)\)
- objective function \(f(x_1, y_1, x_2, y_2, x_3, y_3) = \) 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 \(x \leftarrow x + \alpha \nabla f(x)\)

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