CMPU-365, Spring 2019

Russell & Norvig, Section 4.1
Outline

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

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

Then the state space can be the set of “complete” configurations:
- e.g., for 8-queens, a configuration can be any board with 8 queens.
- e.g., for TSP, a configuration can be any complete tour.

In such cases, we can use iterative improvement algorithms; we keep a single “current” state, and try to improve it:
- e.g., for 8-queens, we gradually move some queen to a better place.
- e.g., for TSP, we start with any tour and gradually improve it.

The goal would be to find an optimal configuration:
- e.g., for 8-queens, an optimal config. is where no queen is threatened.
- e.g., for TSP, an optimal configuration is the shortest route.

This takes constant space, and is suitable for online as well as offline search.
Example: Travelling Salesperson Problem

Start with any complete tour, and 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* × *n* board, with no two queens on the same column.

Move a queen to reduce the number of conflicts; repeat until we cannot move any queen anymore.

– then we are at a local maximum, hopefully it is global too.

This 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.

It is useful to consider the state space landscape:

Random-restart hill climbing overcomes local maxima
  – trivially complete, given enough time

Random sideways moves
  😊 escapes from shoulders 😞 loops 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”

    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
        ∆E ← Value[next] – Value[current]
        if ∆E > 0 then current ← next
        else current ← next only with probability e^{∆E/T}
```

Note: The schedule should decrease the temperature T
so that it gradually goes to 0
Local beam search

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

This is not the same as $k$ searches run in parallel!

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

Idea: choose $k$ successors randomly, biased towards good ones

(“Stochastic local beam search”)
Genetic algorithms (briefly)

Idea:
– a variant of stochastic local beam search
– generate successors from **pairs** of states
– the states have to be encoded as strings

Note: \( 24 / (24+23+20+11) = 31\% \)
Genetic algorithms contd.

GAs require that the states are encoded as strings

The ‘crossover helps iff substrings are meaningful components

\[
\begin{array}{c}
3 & 2 & 7 & 5 & 2 & 4 & 1 & 1 \\
2 & 4 & 7 & 4 & 8 & 5 & 5 & 2
\end{array}
\cdot \begin{array}{c}
3 & 2 & 7 & 4 & 8 & 5 & 5 & 2
\end{array}
\]
Continuous state spaces (very briefly)

Suppose we want to site three airports in Romania:
- 6-D state space is defined by \((x_1, y_2), (x_2, y_2), (x_3, y_3)\)
- objective function \(f(x_1, y_2, x_2, y_2, x_3, y_3) = \) the 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

\[
\nabla f = \begin{pmatrix}
\frac{\partial f}{\partial x_1}, & \frac{\partial f}{\partial y_1}, & \frac{\partial f}{\partial x_2}, & \frac{\partial f}{\partial y_2}, & \frac{\partial f}{\partial x_3}, & \frac{\partial f}{\partial y_3}
\end{pmatrix}
\]

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

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