Beyond Classical Search

Chapter 4, Sections 4.1-4.2

Outline

◇ Hill-climbing

◇ Simulated annealing

◇ Genetic algorithms (briefly)

◇ Local search in continuous spaces (briefly)

Iterative improvement algorithms

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

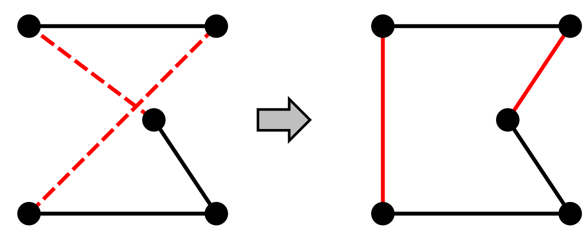
Then state space = set of "complete" configurations;
find **optimal** configuration, e.g., TSP
or, find configuration satisfying constraints, e.g., timetable

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: Traveling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

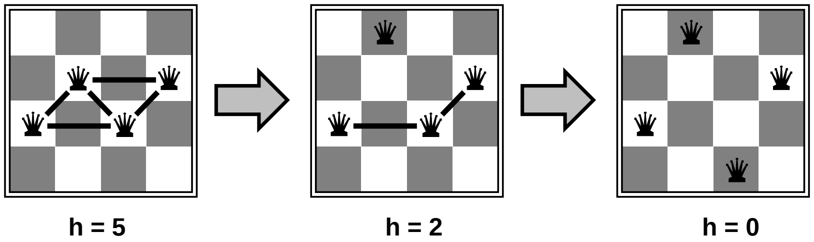


Variants of this approach get within 1% of optimal very quickly with thousands of cities

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)

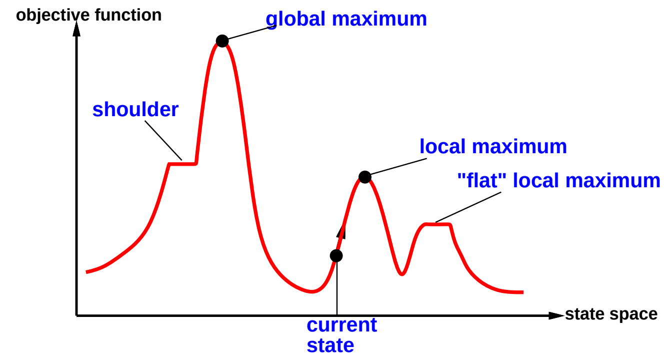
"Like climbing Everest in thick fog with amnesia"

```
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.

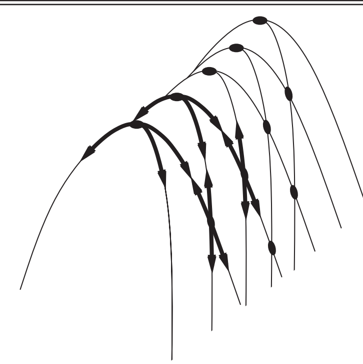
Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima (eventually a good initial state)

Random sideways moves escape from shoulders loop on flat maxima

Ridges



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^(Δ E/T)
```


Properties of simulated annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T decreased slowly enough $\Longrightarrow$ always reach best state x^*
because $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T

Is this necessarily an interesting guarantee??

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

Widely used in VLSI layout, airline scheduling, etc.

Local beam search

Idea: k random initial states; choose and keep top k of all their successors

◇ Not the same as k hill climbing searches run in parallel!

◇ Searches that find good states recruit other searches to join them

◇ However, if the successors from an initial state are not selected, the search starting from that state is effectively abandoned.

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

Idea: ?

Local Beam Search

Idea: k random initial states; choose and keep top k of all their successors

◇ Not the same as k hill climbing searches run in parallel!

◇ Searches that find good states recruit other searches to join them

◇ However, if the successors from an initial state are not selected, the search starting from that state is effectively abandoned.

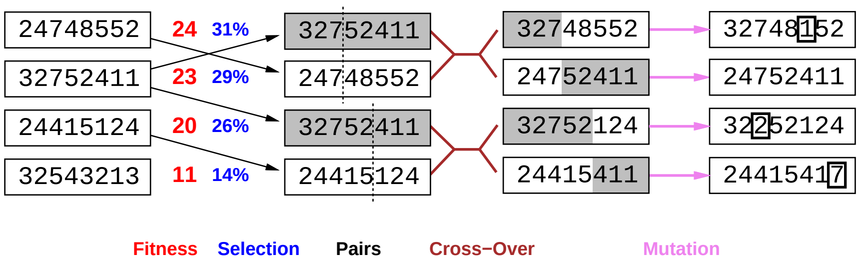
Problem: quite often, all k states end up on same local hill

Idea: choose k successors randomly, biased towards good ones (Stochastic Beam Search)

Observe the close analogy to natural selection!

Genetic algorithms

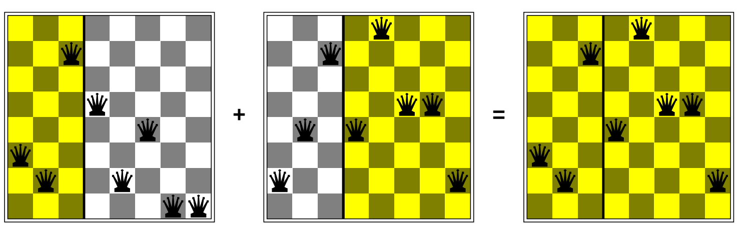
= stochastic beam search + generate successors from **pairs** of states



Genetic algorithms contd.

GAs require states encoded as strings (GPs use programs)

Crossover helps **iff substrings are meaningful components**



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

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

Continuous state spaces–Discretization

◇ 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

◇ each state has six discrete variables (e.g. $\pm\delta$ miles, where δ is a constant) [or grid cells]

◇ each state has how many possible successors?

Continuous state spaces–Discretization

◇ 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

◇ each state has six discrete variables (e.g. $\pm\delta$ miles, where δ is a constant) [or grid cells]

◇ each state has how many possible successors?

- 12 [book] (action: change only one variable—x or ("xor") y of one airport)
- $3^6 - 1$ (action: change at least one variable)

◇ what is the algorithm?

Continuous state spaces–No Discretization

◇ Gradient (of the objective function) 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., only one airport).

◇ Otherwise, 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 $\mathbf{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$

Contrast and Summary

◇ Ch. 3

◇ Ch. 4.1-2

◇ What is the key difference?

Contrast and Summary

◇ Ch. 3: "It is the journey, not the destination." (optimize the path)

◇ Ch. 4.1-2: "It is the destination, not the journey" (optimize the goal)

◇ Different problem formulation, do we still need:

- Initial state (state space): ?
- Successor function (actions): ?
- Step (path) cost: ?
- Goal test: ?

Contrast and Summary

♢ Ch. 3: "It is the journey, not the destination." (optimize the path)

♢ Ch. 4.1-2: "It is the destination, not the journey" (optimize the goal)

♢ Different problem formulation, do we still need:

- Initial state (state space): yes [but different kind of states]
- Successor function (actions): yes [but different kind of actions]
- Step (path) cost: no [not the journey]
- Goal test: no [optimize objective function]

♢ The n-queen and TSP problems can be forumluated in either way, how?

Skipping the rest

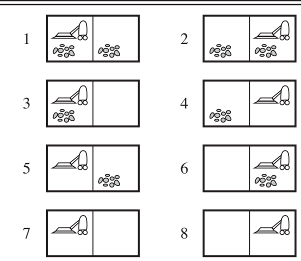
Searching with Non-deterministic Actions

♢ performing an action might not yield the expected successor state

♢ Suck can clean one dirty square, but sometimes an adjacent dirty square as well

♢ Suck on a clean square can sometimes make it dirty

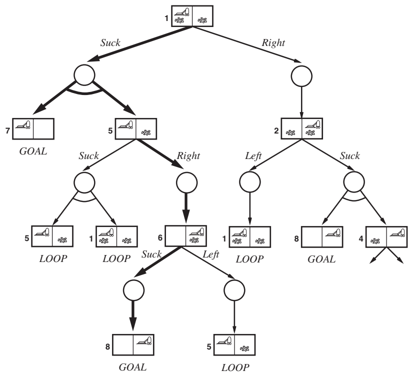
Erratic Vacuum World



♢ not just a sequence of actions, but backup/contingency plans

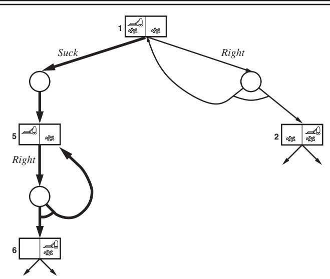
♢ from State 1: [Suck, if State = 5 then [Right, Suck] else []]

And-Or Search Tree



◇ every path reaches a goal, a repeated state, or a dead end

Slippery floor



◇

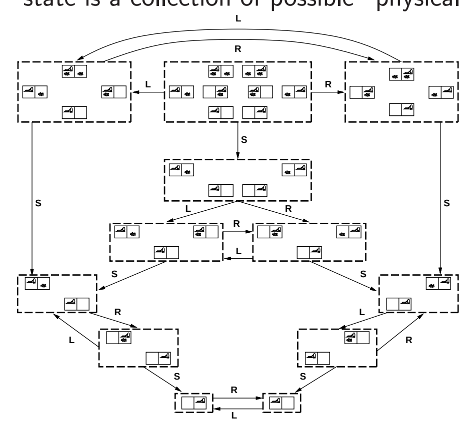
Sensorless problems

◇ No sensor—the agent does not know which state it is in

◇ Is it hopeless?

Belief States

◇ Each “belief” state is a collection of possible “physical” states.



◇ 12 “reachable” belief states (out of 255 possible belief states)

◇ If the actions have uncertain outcomes, how many belief states are there?

Contingency problems

- ♢ Environment is partially observable
- ♢ Fixed sequence: [Suck, Right, Suck]
- ♢ Actions have uncertain outcomes
- ♢ Addtional percepts during execution: [Suck, Right, if [R Dirty] then Suck]
- ♢ More in Chapter 12 (Planning)
- ♢ Adversarial environment (e.g., games): Chapter 6