Beyond Classical Search

Beyond Classical Search

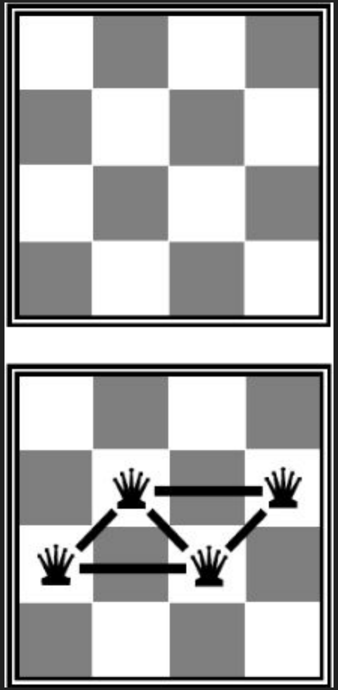
- Chapter 3 covered problems that considered the whole search space and produced a sequence of actions leading to a goal.

- Chapter 4 covers techniques (some developed outside of AI) that don’t try to cover the whole space and only the goal state, not the steps, are important.

- The techniques of Chapter 4 tend to use much less memory and are not guaranteed to find an optimal solution.

Types of Problems

- Planning problems:
  - We want a path to a solution
  - Usually want an optimal path
  - *Incremental formulations*
- Identification problems:
  - We actually just want to know what the goal is
  - Usually want an optimal goal
  - *Complete-state formulations*
  - Iterative improvement algorithms



Iterative improvement

- In many optimization problems, path is irrelevant; the goal state itself is the solution.
- Then, state space = space of “complete”

  configurations.

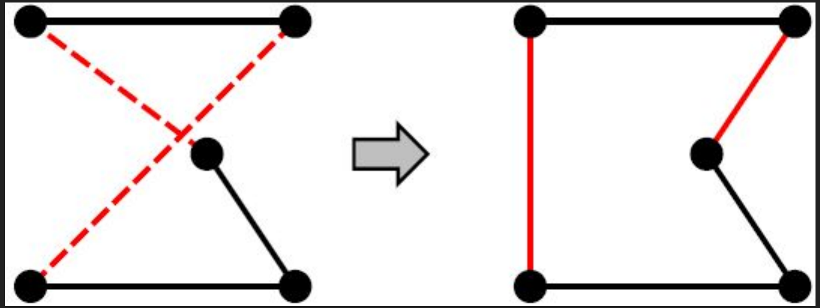
  Algorithm goal:

  - find optimal configuration (e.g., TSP), or,
  - find configuration satisfying constraints

    (e.g., n-queens)
- In such cases, can use iterative improvement algorithms: keep a single “current” state, and try to improve it.

Iterative improvement example: Traveling salesperson problem

- Start with any complete tour, perform pairwise exchanges

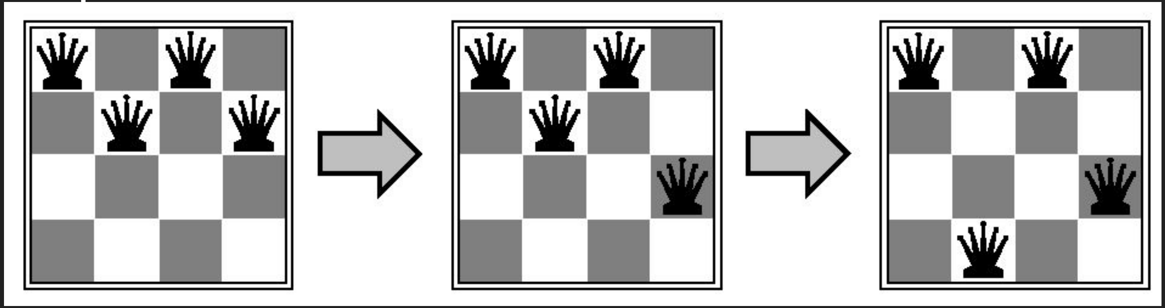


- Variants of this approach get within 1% of optimal very quickly with thousands of cities

Iterative improvement example: n-queens

- Goal: Put n chess-game queens on an n x n board, with no two queens on the same row, column, or diagonal.
  - Here, goal state is initially unknown but is specified by constraints that it must satisfy.

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

More Search Methods

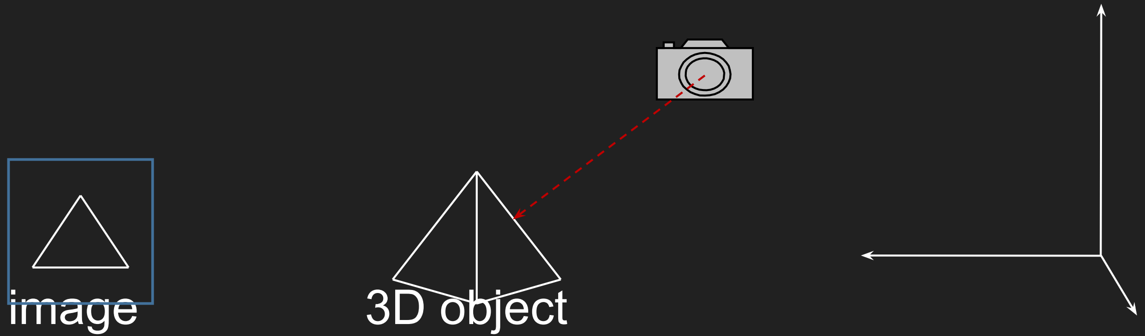
- Local Search
  - Hill Climbing
  - Simulated Annealing
  - Beam Search
  - Genetic Search
- Local Search in Continuous Spaces
- Searching with Nondeterministic Actions
- Online Search (agent is executing actions)

Local Search Algorithms and Optimization Problems

- Complete state formulation
  - For example, for the 8 queens problem, all 8 queens are on the board and need to be moved around to get to a goal state
- Equivalent to optimization problems often found in science and engineering
- Start somewhere and try to get to the solution from there
- Local search around the current state to decide where to go next

Pose Estimation Example

- Given a geometric model of a 3D object and a 2D image of the object.
- Determine the position and orientation of the object wrt the camera that snapped the image.

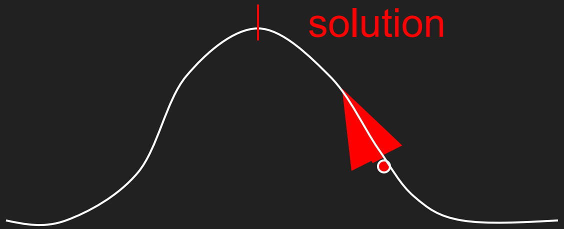


- State (x, y, z, x-angle, y-angle, z-angle)

Hill Climbing

- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit
- Why can this be a terrible idea?
  - Complete?
  - Optimal?
- What’s good about it?

Hill Climbing

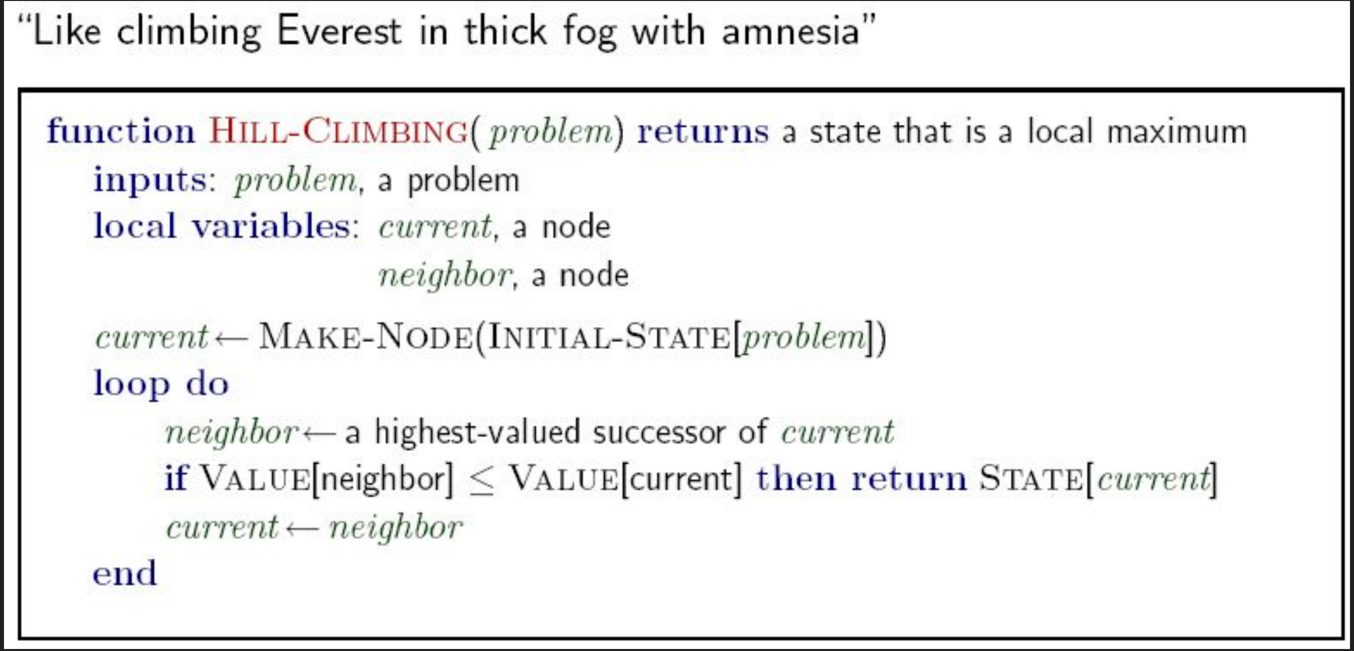


Note: solutions shown here as max not min.

- Often used for numerical optimization problems.
- How does it work?

Hill climbing (or gradient ascent/descent)

- Iteratively maximize "value" of current state, by replacing it by successor state that has highest value, as long as possible.

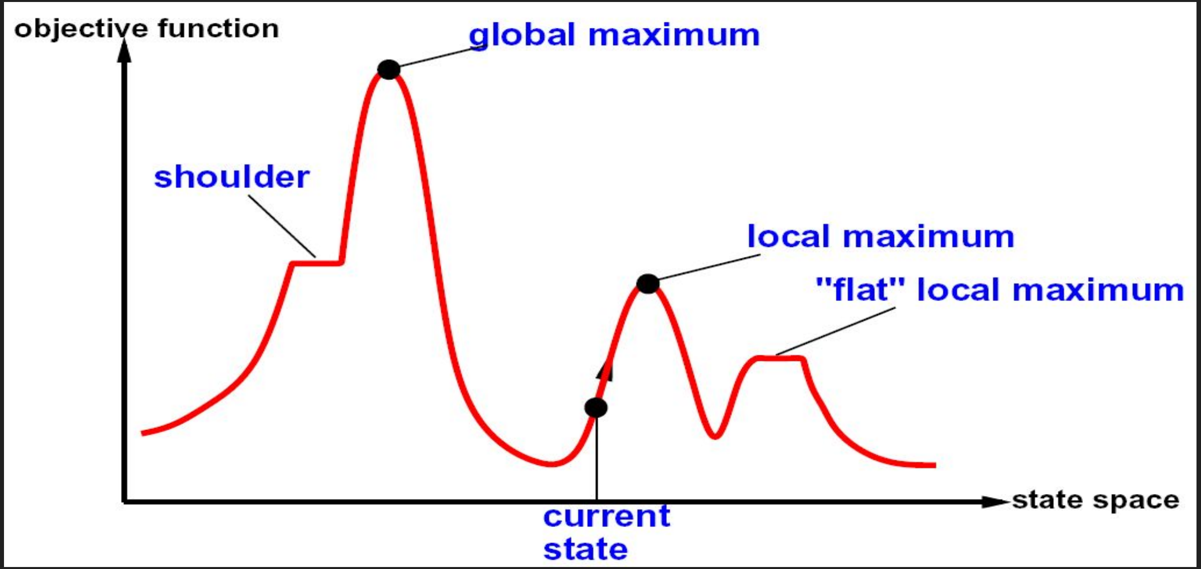


"Like climbing Everest in thick fog with amnesia"

```
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
```

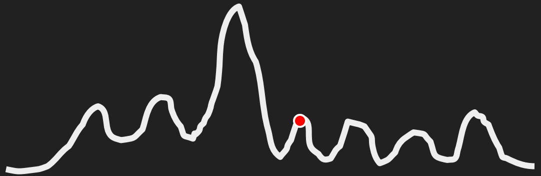
Hill Climbing diagram



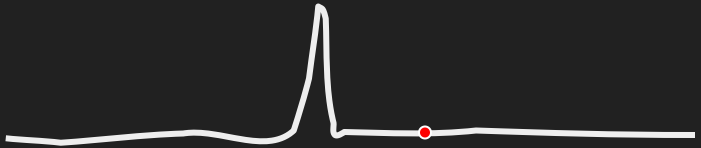
- Random restarts – overcomes local maxima – trivially complete
- Random sideways moves – escape shoulders, loop when flat

Hill Climbing Problems

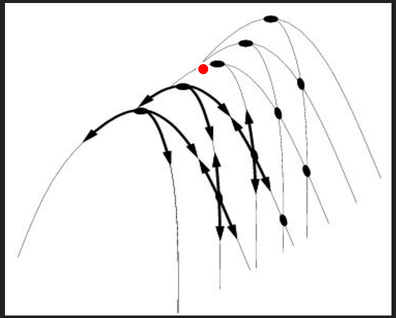
Local maxima



Plateaus



Diagonal ridges



What is it sensitive to?
Does it have any advantages?

Solving the problems

- Allow backtracking (What happens to complexity?)
- Stochastic hill climbing: choose at random from uphill moves, using steepness for a probability
- Random restarts: “If at first you don’t succeed, try, try again.”
- Several moves in each of several directions, then test
- Jump to a different part of the search space

Simulated annealing: basic idea

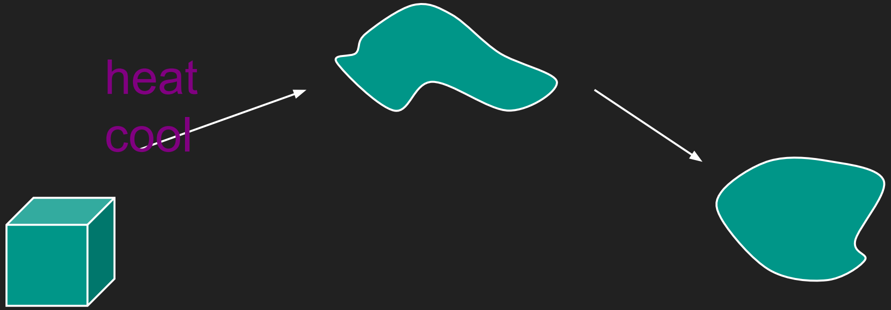
- Idea: Escape local extremes by allowing “bad moves,” but gradually decrease their size and frequency.

- Kirkpatrick et al. 1983:

Simulated annealing is a general method for making likely the escape from local minima by allowing jumps to higher energy states.

Simulated annealing: basic idea

- Comes from the physical process of annealing in which substances are raised to high energy levels (melted) and then cooled to solid state.



- The probability of moving to a higher energy state, instead of lower is

$$p = e^{(-\Delta E/kT)}$$

where ΔE is the positive change in energy level, T is the temperature, and k is Bolzmann’s constant.

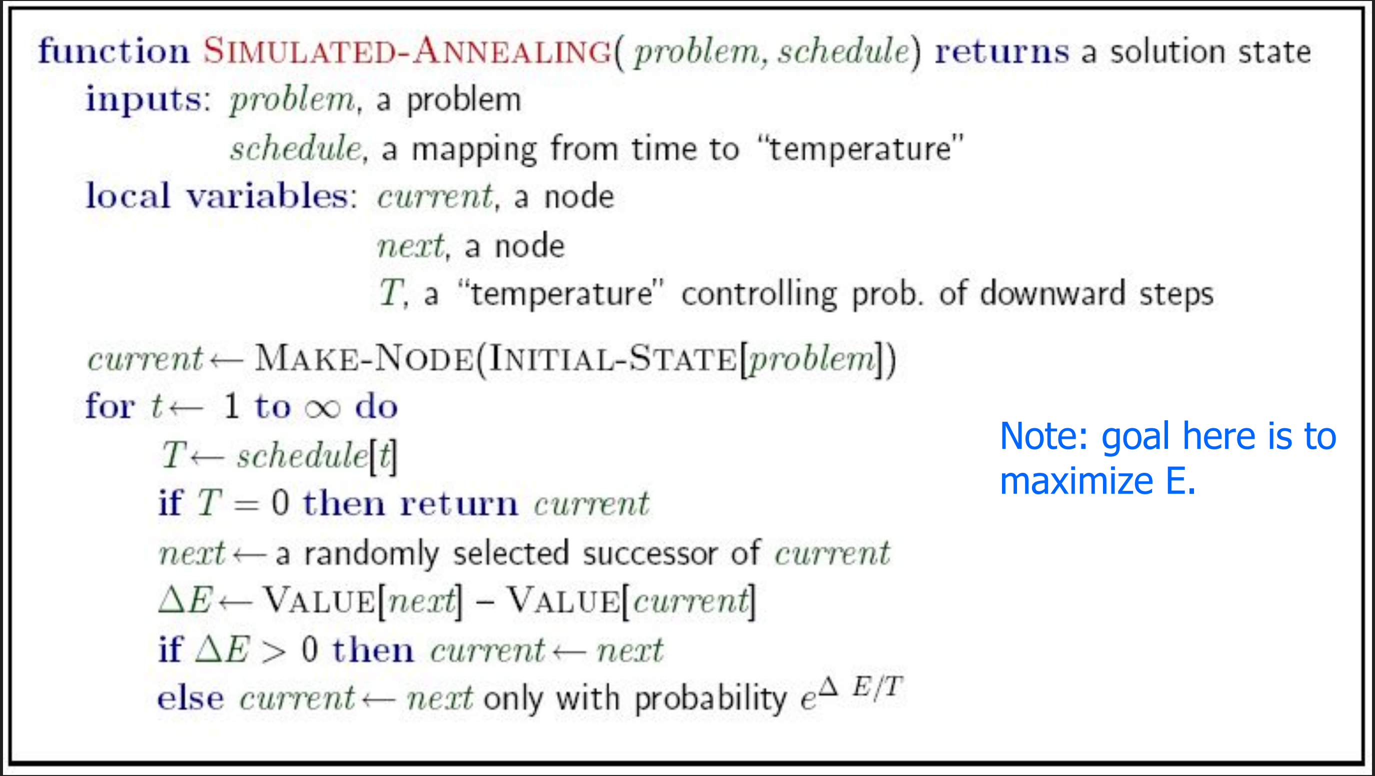
Simulated annealing

- At the beginning, the temperature is high.
- As the temperature becomes lower
  - kT becomes lower
  - ΔE/kT gets bigger
  - (-ΔE/kT) gets smaller
  - e^(-ΔE/kT) gets smaller
- As the process continues, the probability of a downhill move gets smaller and smaller.

For Simulated Annealing

- ΔE represents the change in the value of the objective function.
- Since the physical relationships no longer apply, drop k. So p = $e^{(-\Delta E/T)}$
- We need an annealing schedule, which is a sequence of values of T: $T_0, T_1, T_2, \ldots$

Simulated annealing algorithm



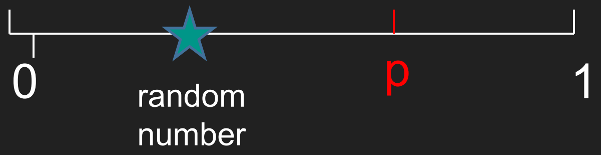
```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        $\Delta E$ ← VALUE[next] – VALUE[current]
        if $\Delta E > 0$ then current ← next
        else current ← next only with probability $e^{\Delta E/T}$
```

Note: goal here is to maximize E.

Probabilistic Selection

- Select *next* with probability p



- Generate a random number
- If it’s <= p, select *next*

Simulated Annealing Properties

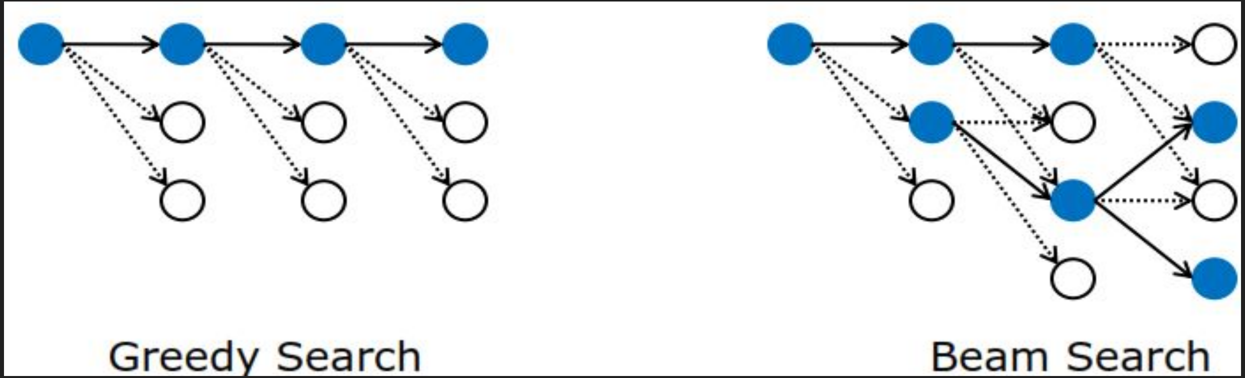
- Theoretical guarantee:
  - Stationary distribution: $$p(x) \propto e^{\frac{E(x)}{kT}}$$
  - If T decreased slowly enough, will converge to optimal state!
- Is this an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape, the less likely you are to every make them all in a row
  - People think hard about *ridge operators which let you* jump around the space in better way

Simulated Annealing Applications

- Basic Problems
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling
- Engineering
  - VLSI design
    - Placement
    - Routing
    - Array logic minimization
    - Layout
  - Facilities layout
  - Image processing
  - Code design in information theory

Local Beam Search

- Idea: Like greedy search but keep k states instead of 1; choose top k of all their successors

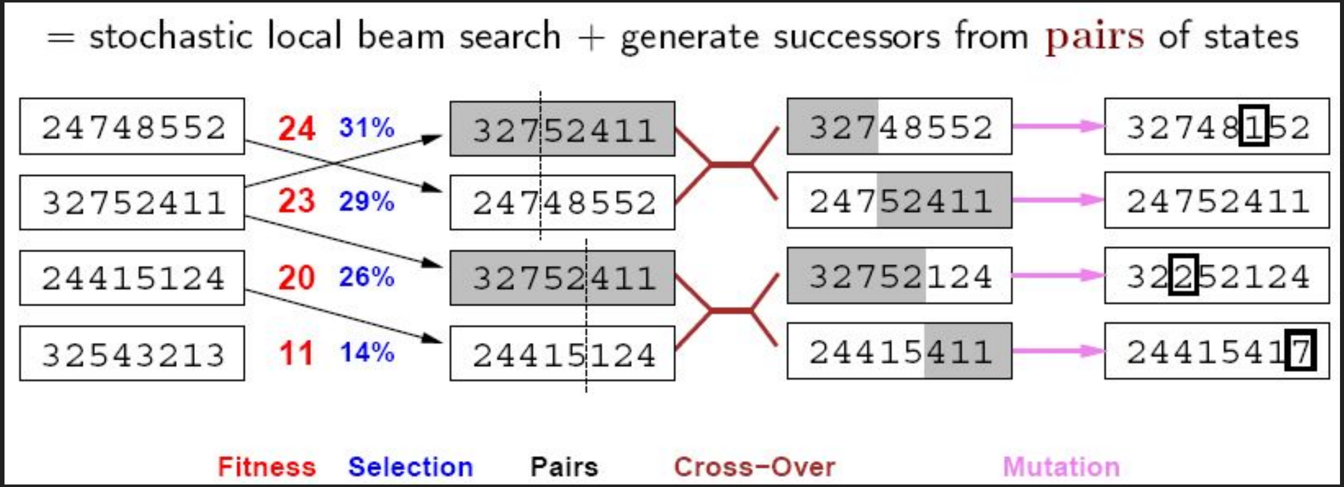


- Not the same as k searches run in parallel!
- Searches that find good states recruit other searches to join them

Local Beam Search

- 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!
- Variables: beam size, encourage diversity?
    - The best choice in MANY practical settings
    - Complete? Optimal?
    - Why do we still need optimal methods?

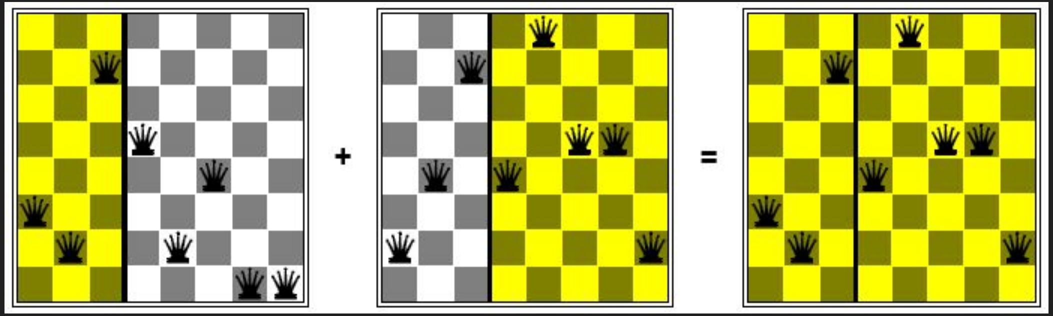
Genetic Algorithms



- Genetic algorithms use a natural selection metaphor
  - Like beam search (selection), but also have pairwise crossover
- operators, with optional mutation
  - Probably the most misunderstood, misapplied (and even maligned) technique around!

Genetic Algorithms

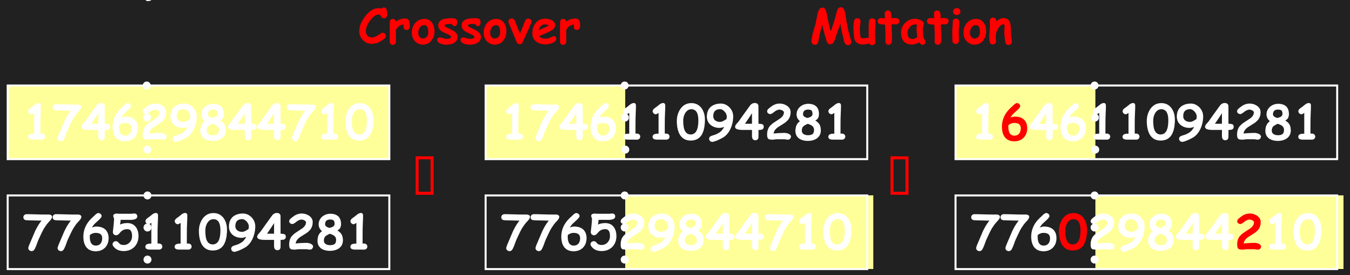
- GAs require states encoded as strings (GPs use programs)
- Crossover helps iff substrings are meaningful components



- GAs ≠ evolution: e.g., real genes encode replication machinery!
- Why does crossover make sense here?
  - When wouldn’t it make sense?
  - What would mutation be?
  - What would a good fitness function be?

Genetic Algorithms

- Start with random population of states
  - Representation serialized (ie. strings of characters or bits)
  - States are ranked with “fitness function”
- Produce new generation
  - Select random pair(s) using probability:
    - probability ~ fitness
  - Randomly choose “crossover point”
    - Offspring mix halves
  - Randomly mutate bits



Genetic Algorithm

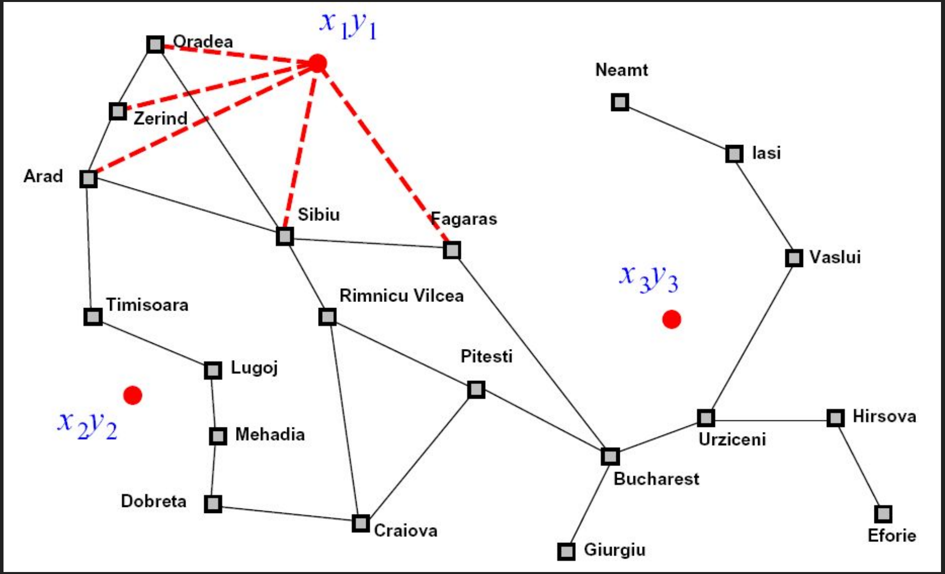
```
● Given: population P and fitness-function f
● repeat
    ○ newP □ empty set
    ○ for i = 1 to size(P)
        x □ RandomSelection(P,f)
        y □ RandomSelection(P,f)
        child □ Reproduce(x,y)
        if (small random probability) then child □ Mutate(child)
        add child to newP
    ○ P □ newP
● until some individual is fit enough or enough time has elapsed
● return the best individual in P according to f
```

Using Genetic Algorithms

- 2 important aspects to using them
  - 1. How to encode your real-life problem
  - 2. Choice of fitness function
- Research Example
  - We want to generate a new operator for finding interesting points on images
  - The operator will be some function of a set of values $v_1, v_2, \dots v_K$.
  - Idea: weighted sums, weighted mins, weighted maxs. $a_1,\dots a_K, b_1,\dots b_K, c_1,\dots c_k$

Continuous Problems

- Placing airports in Romania
  - States: (x1,y1,x2,y2,x3,y3)
  - Cost: sum of squared distances to closest city



Local Search in Continuous Spaces

- Given a continuous state space

  $S = \{(x_1, x_2, \ldots, x_N) \mid x_i \varepsilon \mathbb{R}\}$
- Given a continuous objective function $f(x_1, x_2, \ldots, x_N)$
- The gradient of the objective function is a vector

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

- The gradient gives the magnitude and direction of the steepest slope at a point.

Local Search in Continuous Spaces

- To find a maximum, the basic idea is to set $\nabla f = 0$
- Then updating of the current state becomes

  x □ $x + \alpha \nabla f(x)$

  where α is a small constant.
- Theory behind this is taught in numerical methods classes.
- Your book suggests the Newton-Raphson method. Luckily there are packages…..

Searching with Nondeterministic Actions

- Vacuum World (actions = {left, right, suck})

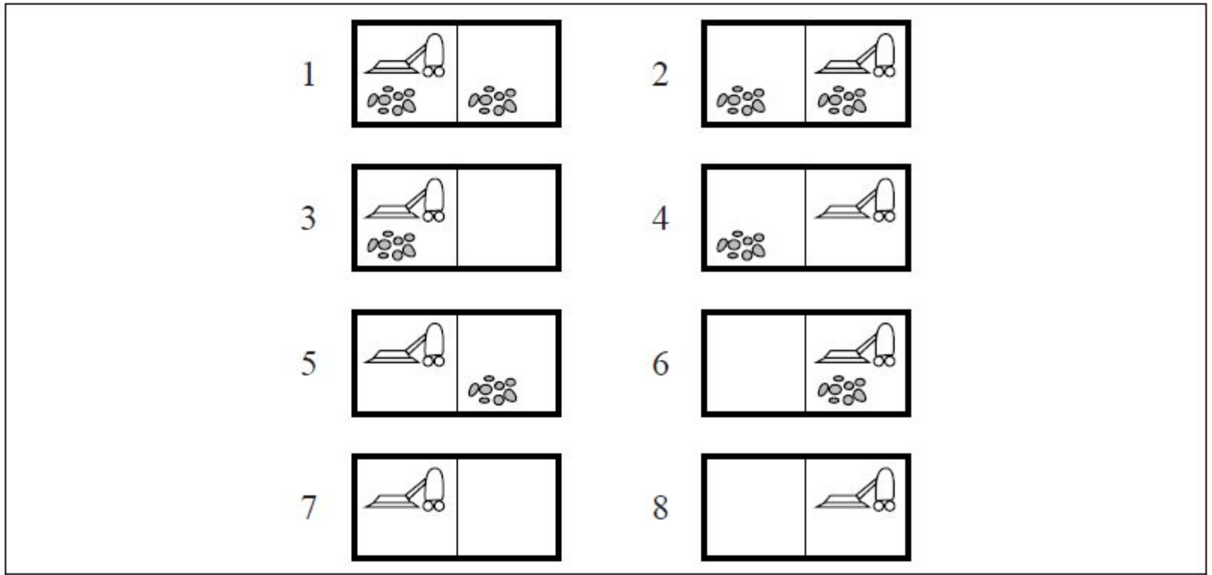


Figure 4.9 The eight possible states of the vacuum world; states 7 and 8 are goal states.

Searching with Nondeterministic Actions

In the nondeterministic case, the result of an action can vary.

Erratic Vacuum World:

- When sucking a dirty square, it cleans it and sometimes cleans up dirt in an adjacent square.
- When sucking a clean square, it sometimes deposits dirt on the carpet.

Generalization of State-Space Model

1. Generalize the transition function to return a set of possible outcomes.

   oldf: $S \times A \rightarrow S$ newf: $S \times A \rightarrow 2^S$

2. Generalize the solution to a contingency plan.

   if state=s then action-set-1 else action-set-2

3. Generalize the search tree to an AND-OR tree.

AND-OR Search Tree

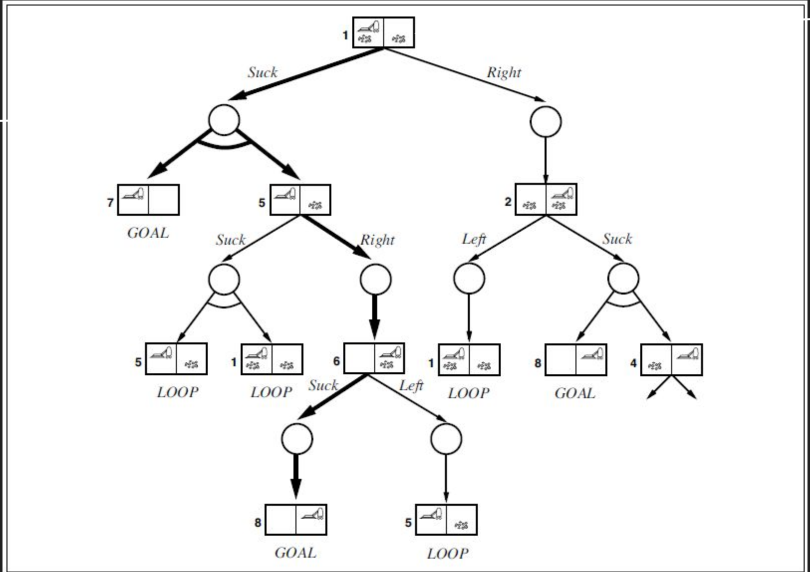


Figure 4.10 The first two levels of the search tree for the erratic vacuum world. State nodes are OR nodes where some action must be chosen. At the AND nodes, shown as circles, every outcome must be handled, as indicated by the arc linking the outgoing branches. The solution found is shown in bold lines.

Searching with Partial Observations

- The agent does not always know its state!
- Instead, it maintains a belief state: a set of possible states it might be in.
- Example: a robot can be used to build a map of a hostile environment. It will have sensors that allow it to “see” the world.

Belief states for sensor less agents

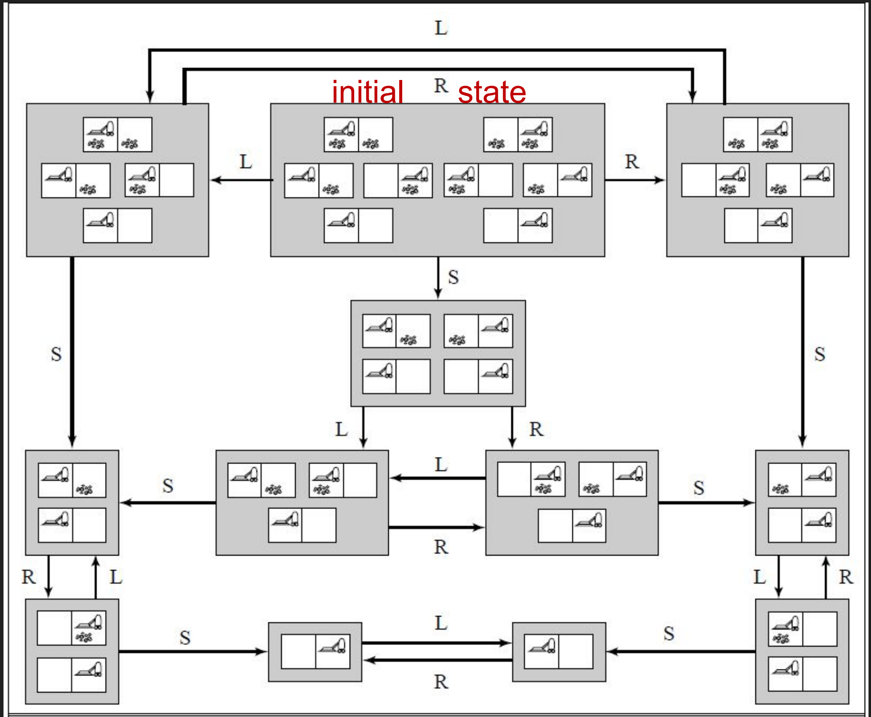


Figure 4.14 The reachable portion of the belief-state space for the deterministic, sensorless vacuum world. Each shaded box corresponds to a single belief state. At any given point, the agent is in a particular belief state but does not know which physical state it is in. The initial belief state (complete ignorance) is the top center box. Actions are represented by labeled links. Self-loops are omitted for clarity.