A.I.: Beyond Classical Search
Trivial Algorithms

• Random Sampling
  – Generate a state randomly

• Random Walk
  – Randomly pick a neighbor of the current state

• Both algorithms asymptotically complete.
Overview

• Previously we addressed a single category of problems: observable, deterministic, known environments where the solution is a sequence of actions.
• Now we consider what happens when these assumptions are relaxed.
• First we look at purely local search strategies, including methods inspired by statistical physics (simulated annealing) and evolutionary biology (genetic algorithms).
• Later, we examine what happens when we relax the assumptions of determinism and observability. The key idea is that the agent cannot predict exactly what percept it will receive, so it considers a contingency plan.
• Lastly, we discuss online search.
Local Search for Optimization

• Note that for many types of problems, the path to a goal is irrelevant (we simply want the solution – consider the 8-queens).

• If the path to the goal does not matter, we might consider a class of algorithms that don’t concern themselves with paths at all.

• **Local search algorithms** operate using a single current node (rather than multiple paths) and generally move only to neighbors of that node.

• Typically, the paths followed by the search are not retained (the benefit being a potentially substantial memory savings).
Local Search for Optimization

• Local search
  – Keep track of single current state
  – Move only to neighboring states
  – Ignore paths

• Advantages:
  – Use very little memory
  – Can often find reasonable solutions in large or infinite (continuous) state spaces.

• “Pure optimization” problems
  – All states have an objective function
  – Goal is to find state with max (or min) objective value
  – Does not quite fit into path-cost/goal-state formulation
  – Local search can do quite well on these problems.
Local Search for Optimization

- In addition to finding goals, local search algorithms are useful for solving pure optimization problems, in which we aim to find the best state according to an objective function.

- Nature, for example, provides an objective function – reproductive fitness.

- To understand local search, we consider state-space landscapes, as shown next.
Optimization

• So what is optimization?

• Find the minimum or maximum of an objective function (usually: given a set of constraints):

\[
\arg \min_x f_0(x) \\
\text{s.t. } f_i(x) \leq 0, \; i = \{1, \ldots, k\} \\
h_j(x) = 0, \; j = \{1, \ldots, l\}
\]
Why Do We Care?

**Linear Classification**

$$\arg \min_w \sum_{i=1}^{n} ||w||^2 + C \sum_{i=1}^{n} \xi_i$$

s.t. $1 - y_i x_i^T w \leq \xi_i$

$$\xi_i \geq 0$$

**Maximum Likelihood**

$$\arg \max_{\theta} \sum_{i=1}^{n} \log p_\theta(x_i)$$

**K-Means**

$$\arg \min_{\mu_1, \mu_2, \ldots, \mu_k} J(\mu) = \sum_{j=1}^{k} \sum_{i \in C_j} ||x_i - \mu_j||^2$$
Convexity

• We prefer convex problems.
Local Search for Optimization

State-Space Landscape

• If elevation corresponds with cost, then the aim is to find the lowest valley – a global minimum; if elevation corresponds to an objective function, then the aim is to find the highest peak – a global maximum.

• A complete local search algorithm always finds a goal (if one exists); an optimal algorithm always finds a global minimum/maximum.
Hill-Climbing (Greedy Local Search)

**function** HILL-CLIMBING( *problem*) return a state that is a local maximum

**input:** *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
```

Minimum version will reverse inequalities and look
for lowest valued successor
Hill-Climbing

• “A loop that continuously moves towards increasing value”
  – terminates when a peak is reached
  – Aka greedy local search
• Value can be either
  – Objective function value
  – Heuristic function value (minimized)
• Hill climbing does not look ahead of the immediate neighbors
• Can randomly choose among the set of best successors
  – if multiple have the best value
• “Climbing Mount Everest in a thick fog with amnesia”
Hill-Climbing (8-queens)

- Need to convert to an optimization problem!
Hill-Climbing (8-queens)

• State
  – All 8 queens on the board in some configuration

• Successor function
  – move a single queen to another square in the same column.

• Example of a heuristic function $h(n)$:
  – the number of pairs of queens that are attacking each other
  – (so we want to minimize this)
Hill-Climbing (8-queens)

- $h = \text{number of pairs of queens that are attacking each other}$
- $h = 17$ for the above state
Hill-Climbing (8-queens)

- Randomly generated 8-queens starting states...
- 14% the time it solves the problem
- 86% of the time it get stuck at a local minimum

However...

- Takes only 4 steps on average when it succeeds
- And 3 on average when it gets stuck
- (for a state space with $8^8 \approx 17$ million states)
Hill Climbing Drawbacks

- Local maxima
- Plateaus
- Diagonal ridges
Escaping Shoulders: Sideways Move

• If no downhill (uphill) moves, allow sideways moves in hope that algorithm can escape
  – Need to place a limit on the possible number of sideways moves to avoid infinite loops

• For 8-queens
  – Now allow sideways moves with a limit of 100
  – Raises percentage of problem instances solved from 14 to 94%

  – However....
    • 21 steps for every successful solution
    • 64 for each failure
Tabu Search

• Prevent returning quickly to the same state
• Keep fixed length queue (“tabu list”)
• Add most recent state to queue; drop oldest
• Never make the step that is currently tabu’ed

• Properties:
  – As the size of the tabu list grows, hill-climbing will asymptotically become “non-redundant” (won’t look at the same state twice)
  – In practice, a reasonable sized tabu list (say 100 or so) improves the performance of hill climbing in many problems
Escaping Shoulders/local Optima
Enforced Hill Climbing

• Perform breadth first search from a local optima
  – to find the next state with better h function

• Typically,
  – prolonged periods of exhaustive search
  – bridged by relatively quick periods of hill-climbing

• Middle ground b/w local and systematic search
Hill-climbing: stochastic variations

• Stochastic hill-climbing
  – Random selection among the uphill moves.
  – The selection probability can vary with the steepness of the uphill move.

• To avoid getting stuck in local minima
  – Random-walk hill-climbing
  – Random-restart hill-climbing
  – Hill-climbing with both
Hill Climbing: stochastic variations

→ When the state-space landscape has local minima, any search that moves only in the greedy direction cannot be complete

→ Random walk, on the other hand, is asymptotically complete

Idea: Put random walk into greedy hill-climbing
Hill-climbing with random restarts

- If at first you don’t succeed, try, try again!
- Different variations
  - For each restart: run until termination vs. run for a fixed time
  - Run a fixed number of restarts or run indefinitely
- Analysis
  - Say each search has probability \( p \) of success
    - E.g., for 8-queens, \( p = 0.14 \) with no sideways moves
  - Expected number of restarts?
  - Expected number of steps taken?
- If you want to pick one local search algorithm, learn this one!!
Hill-climbing with random walk

• At each step do one of the two
  – Greedy: With prob $p$ move to the neighbor with largest value
  – Random: With prob $1-p$ move to a random neighbor

Hill-climbing with both

• At each step do one of the three
  – Greedy: move to the neighbor with largest value
  – Random Walk: move to a random neighbor
  – Random Restart: Resample a new current state
Simulated Annealing

- Simulated Annealing = physics inspired twist on random walk
- Basic ideas:
  - like hill-climbing identify the quality of the local improvements
  - instead of picking the best move, pick one randomly
  - say the change in objective function is $d$
  - if $d$ is positive, then move to that state
  - otherwise:
    - move to this state with probability proportional to $d$
    - thus: worse moves (very large negative $d$) are executed less often
  - however, there is always a chance of escaping from local maxima
  - over time, make it less likely to accept locally bad moves
  - (Can also make the size of the move random as well, i.e., allow “large” steps in state space)
Physical Interpretation of Simulated Annealing

• A Physical Analogy:
  
  • imagine letting a ball roll downhill on the function surface
  
    – this is like hill-climbing (for minimization)
  
  • now imagine shaking the surface, while the ball rolls, gradually reducing the amount of shaking
  
    – this is like simulated annealing

• Annealing = physical process of cooling a liquid or metal until particles achieve a certain frozen crystal state

  • simulated annealing:
    
    – free variables are like particles
    
    – seek “low energy” (high quality) configuration
    
    – slowly reducing temp. T with particles moving around randomly
Simulated Annealing

Convergence of simulated annealing

- AT INIT_TEMP: Unconditional Acceptance
- COST FUNCTION, C
- NUMBER OF ITERATIONS
- AT FINAL_TEMP
- Move accepted with probability $e^{-\Delta C/\text{Temp}}$
Simulated annealing

function SIMULATED-ANNEALING( problem, schedule) return a solution state

input: problem, a problem  
schedule, a mapping from time to temperature

local variables: current, a node.  
next, a node.  
T, a “temperature” controlling the 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
  ∆E ← VALUE[next] - VALUE[current]
  if ∆E > 0 then current ← next
  else current ← next only with probability $e^{\Delta E / T}$
Temperature T

• High T: probability of “locally bad” move is higher.
• Low T: probability of “locally bad” move is lower.
• Typically, T is decreased as the algorithm runs longer, i.e., there is a “temperature schedule”.

• In statistical mechanics, the **Boltzmann distribution** is a probability distribution that gives the probability of a certain state as a function of that state’s energy and temperature of the system to which the distribution is applied. It is given as:

\[
p_i = \frac{e^{-\varepsilon_i/kT}}{\sum_{j=1}^{M} e^{-\varepsilon_j/kT}}
\]
Simulated Annealing in Practice

  • theoretically will always find the global optimum

– Other applications: Traveling salesman, Graph partitioning, Graph coloring, Scheduling, Facility Layout, Image Processing, …

– useful for some problems, but can be very slow
  • slowness comes about because T must be decreased very gradually to retain optimality
Local beam search

• Idea: Keeping only one node in memory is an extreme reaction to memory problems.

• Keep track of $k$ states instead of one
  – Initially: $k$ randomly selected states
  – Next: determine all successors of $k$ states
  – If any of successors is goal $\rightarrow$ finished
  – Else select $k$ best from successors and repeat
Local Beam Search (contd)

• Not the same as *k* random-start 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: Stochastic beam search
  – Choose *k* successors randomly, biased towards good ones

• Observe the close analogy to natural selection!
Genetic algorithms

- Twist on Local Search: successor is generated by combining two parent states

- A state is represented as a string over a finite alphabet (e.g. binary)
  - 8-queens
    - State = position of 8 queens each in a column

- Start with \( k \) randomly generated states (population)

- Evaluation function (fitness function):
  - Higher values for better states.
  - Opposite to heuristic function, e.g., \# non-attacking pairs in 8-queens

- Produce the next generation of states by “simulated evolution”
  - Random selection
  - Crossover
  - Random mutation
Genetic algorithms & 8-queens

Can we evolve 8-queens through genetic algorithms?

String representation
16257483
Genetic algorithms

- Fitness function: number of non-attacking pairs of queens (min = 0, max = \(8 \times \frac{7}{2} = 28\))
- \(\frac{24}{24+23+20+11} = 31\%\)
- \(\frac{23}{24+23+20+11} = 29\%\) etc

4 states for 8-queens problem
2 pairs of 2 states randomly selected based on fitness. Random crossover points selected
New states after crossover
Random mutation applied
Genetic algorithms

Has the effect of “jumping” to a completely different new part of the search space (quite non-local)
Comments on Genetic Algorithms

• Genetic algorithm is a variant of “stochastic beam search”

• Positive points
  – Random exploration can find solutions that local search can’t
    • (via crossover primarily)
  – Appealing connection to human evolution
    • “neural” networks, and “genetic” algorithms are metaphors!

• Negative points
  – Large number of “tunable” parameters
    • Difficult to replicate performance from one problem to another
  – Lack of good empirical studies comparing to simpler methods
  – Useful on some (small?) set of problems but no convincing evidence that GAs are better than hill-climbing w/random restarts in general
Optimization of Continuous Functions

• Discretization
  – use hill-climbing

• Gradient descent
  – make a move in the direction of the gradient
    • gradients: closed form or empirical
Objective Function in Continuous Search Space

\[ f(x,y) = e^{-(x^2+y^2)} + 2e^{-(x-1.7)^2 + (y-1.7)^2} \]
Gradient Descent

Assume we have a continuous function: $f(x_1, x_2, \ldots, x_N)$ and we want to minimize over continuous variables $X_1, X_2, \ldots, X_n$

1. Compute the gradients for all $i$: $\frac{\partial f(x_1, x_2, \ldots, x_N)}{\partial x_i}$

2. Take a small step downhill in the direction of the gradient:

   $$x_i \leftarrow x_i - \lambda \frac{\partial f(x_1, x_2, \ldots, x_N)}{\partial x_i}$$

3. Repeat.

   - **How to select $\lambda$**
     - Line search: successively double until $f$ starts to increase again