Artificial Intelligence

Local Search

Vibhav Gogate

The University of Texas at Dallas

Some material courtesy of Luke Zettlemoyer, Dan Klein, Dan Weld, Alex Ihler, Stuart Russell, Mausam
Systematic Search: Review

• Be able to formulate a problem as a Search problem
• Tree Search vs Graph Search
• How they operate and expand nodes
  – Priority queue
    • Breadth first: $f(n)=d(n)$
    • Depth first: $f(n)=-d(n)$
    • Uniform cost search: $f(n)=g(n)$
    • Best first search: $f(n)=h(n)$
    • $A^*$: $f(n)=g(n)+h(n)$
Systematic Search: Properties

• Completeness?
• Optimality?
  – Depth-first search (graph search/path checking)
  – A* search (admissibility vs consistency)
• Time Complexity?
• Space Complexity?
• How to construct heuristics
• Pros and Cons of various search methods, which method to use when
Outline

• Local search techniques and optimization
  – Hill-climbing
  – Simulated annealing
  – Genetic Algorithms (read the book)
  – Issues with local search
Local search and optimization

- Previous lecture: path to goal is solution to problem
  - systematic exploration of search space.

- This lecture: a state is solution to problem
  - for some problems path is irrelevant.
  - E.g., 8-queens
Local search and 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.
Trivial Algorithms

• Random Sampling
  – Generate a state randomly

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

• Both algorithms asymptotically complete.
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])
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
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
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

• Min version will reverse inequalities and look for lowest valued successor
Hill-climbing search

• “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”
“Landscape” of search

objective function

shoulder

global maximum

local maximum

“flat” local maximum

current state

state space
“Landscape” of search

Hill Climbing gets stuck in local minima
“Landscape” of search

Objective function

Global maximum

Shoulder

Local maximum

“Flat” local maximum

Current state

State space

Hill Climbing gets stuck in local minima depending on?
Example: $n$-queens

- Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
Hill-climbing search: 8-queens problem
Hill-climbing search: 8-queens problem

- Formulate it as an optimization problem
Hill-climbing search: 8-queens problem

• Formulate it as an optimization problem
• $h =$ number of pairs of queens that are attacking each other
Hill-climbing search: 8-queens problem

• *Formulate it as an optimization problem*
• $h = \text{number of pairs of queens that are attacking each other}$
• $h = 17$ for the above state
Hill-climbing search: 8-queens problem

- Formulate it as an optimization problem
- $h =$ number of pairs of queens that are attacking each other
- $h = 17$ for the above state
  - $3+4+2+3+2+2+1+0$ (from left to right)
Search Space

• 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 search: 8-queens problem

• Is this a solution?
• What is h?
Hill-climbing on 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 $8^8 = \sim 17$ million states)
Hill Climbing Drawbacks

- Local maxima
- Plateaus
Escaping Plateaus: 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, success comes at a cost
    • 21 steps for every successful solution
    • 64 for each failure
Hill-Climbing: Review

• Solution=Goal state; path to goal is irrelevant

• Algorithm:
  – Assign a cost to each state
  – Current state=randomly selected state \textit{(Initialization)}
  – Loop
    • If current state is the goal state or local maxima
      – Return current state
    • Current state = the best successor of the current state in terms of cost

• Problems: Local maxima, plateaus, initialization
Hill Climbing: Stochastic variations

• How to escape local maxima/plateaus
• Stochastic variations
  – Restart hill-climbing with different initializations
  – Choose each uphill move with some probability
  – Sometimes even pick a downward move

• What we will cover?
  – Random-restart hill climbing
  – Random-walk hill climbing
  – Combination
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? = $1/p$ (why?)
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? = $1/p$ (why?)

• 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 idea:
  – instead of picking the best move, pick one randomly
  – say the change in objective function is $\delta$
  – if $\delta$ is positive (i.e. uphill move), then move to that state
  – otherwise:
    • move to this state with probability proportional to $\delta$
    • thus: worse moves (very large negative $\delta$) are executed less often
  – over time, make it less likely to accept locally bad moves using a parameter “$T$” called the temperature!
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

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
    ΔE ← VALUE[next] - VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability $e^{ΔE/T}$
Temperature $T$

- $current \leftarrow next$ only with probability $e^{\Delta E / T}$
  - 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”
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
Hey! Perhaps sex can improve search?
Sure!
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: Read the book

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

• Positive points
  – Appealing connection to human evolution
    • “neural” networks, and “genetic” algorithms are metaphors!
  – Probabilistically complete just like random walk

• Negative points and that is why we won’t cover it!!!
  – 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