CSEP 573: Artificial Intelligence
Winter 2019

Local Search

Dan Weld

With slides from
Dan Klein, Stuart Russell, Andrew Moore, Luke Zettlemoyer

Previous Search Methods

Systematic
- Blind Search
  - Depth first search
  - Breadth first search
  - Iterative deepening search
  - Uniform cost search
- Informed Search
  - Best First
  - A*
  - Beam Search
  - Hill Climbing

Heuristic = Estimate of solution cost

Local (Randomized)
Constraint Satisfaction (Factored)

Beam Search

- Idea
  - Best first but only keep N best items on priority queue
- Evaluation
  - Complete?
  - Time Complexity?
  - Space Complexity?

Hill Climbing

"Gradient ascent"

- Idea
  - Always choose best child; no backtracking
  - Beam search with |queue| = 1
- Problems?
  - Local maxima

"climbing Mount Everest in a thick fog with amnesia"

  - Plateaus

  - Diagonal ridges

But…

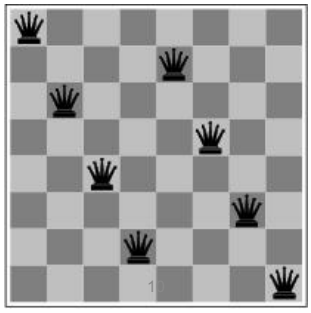
- Simple
- Fast
- O(1) memory

In convex spaces, … hill climbing can be fantastic

8

Goal **State** *vs.* Path

- Previously: Search to find best path to goal
- For some problems path is irrelevant.
  - E.g., 8-queens
- Different algorithms can be used
  - Systematic Search
  - Local Search
  - Constraint Satisfaction



N Queens Problem

Place N queens so they don't attack each other
(i.e. not on same row, same col, same diagonal)

- States

  Chess board with 0 or more columns having a queen, not attacking each other

- Operators

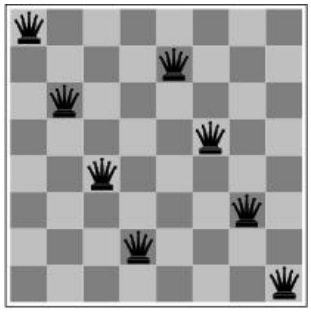
  Add a nonattacking queen

- Initial

  No queens

- Goal

  N queens



N Queens Problem

Place N queens so they don't attack each other
(i.e. not on same row, same col, same diagonal)

- States

  Chess board with exactly N queens, one per column – possibly attacking each other

- Operators

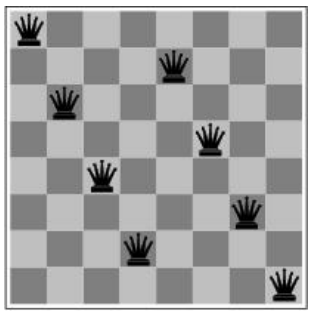
  Move a queen

- Initial

  Random assignment of N

- Goal

  No attacking queens

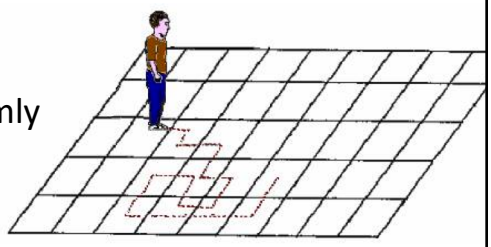


Local search algorithms

- State space = set of "complete" configurations
- Find configuration satisfying constraints,
  - e.g., all n-queens on board, no attacks
- In such cases, we can use local search algorithms
- Keep a single "current" state, try to improve it.
  - E.g., by hill climbing
- Very memory efficient
  - *duh* - only remember current state

Trivial Algorithms

- Random Sampling
  - Generate a state randomly



- Random Walk
  - Randomly pick a neighbor of the current state
- Why even mention these?
  - Both algorithms asymptotically complete.
  - http://projecteuclid.org/download/pdf_1/euclid.aop/1176996718 for Random Walk

16

Need Heuristic Function

What can we relax?

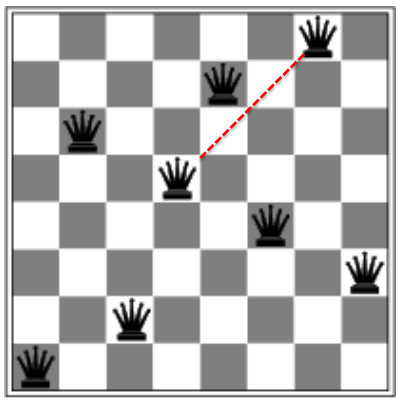
- *Assume each time we move a queen, we don't create new problems*
- *h* = number of ***pairs*** of queens attacking each other
- *h = 17* for the above state

19

Hill-climbing search: 8-queens

Result of hill-climbing in this case…

Bummer



A local minimum with *h = 1*

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 =~17 million states)

21

Escaping Shoulders: Sideways Move

- If no downhill (uphill) moves, allow sideways moves in hope that algorithm can escape
  - Must limit the number of possible sideways moves to avoid infinite loops
- For 8-queens
  - Allow sideways moves with limit of 100
  - Raises percentage of problems solved from 14 to 94%

  - However….
    - 21 steps for every successful solution
    - 64 for each failure

22

Tabu Search

- Prevent returning quickly to the same state
- Keep fixed length queue ("tabu list")
- Add most recent state to queue; drop oldest
- Never move to a tabu state

- 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

23

Escaping 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

24

Can we do better?

25

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: Combine random walk & greedy hill-climbing

At each step do one of the following:

- Greedy: With prob p move to the neighbor with largest value
- Random: With prob 1-p move to a random neighbor

26

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?

Restarts	0	2	4	8	16	32	64
Success?	14%	36%	53%	74%	92%	99%	99.994%

  - Expected number of steps taken?

27

Hill-Climbing with Both Random Walk & Random Sampling

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: Start over from a new, random state

Use this algorithm!

28

Simulated Annealing

written to find minimum value solutions

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 $e^{-\Delta E/T}$

30

Physical Interpretation of Simulated Annealing

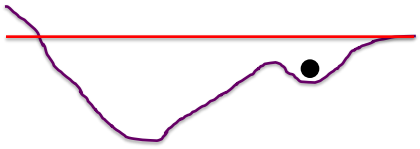
- A Physical Analogy:
  - Imagine letting a ball roll downhill on the function surface
  - Now shake the surface, while the ball rolls,
  - Gradually reducing the amount of shaking

Minimization (not max)

31

Physical Interpretation of Simulated Annealing

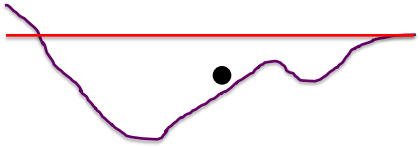
- A Physical Analogy:
  - Imagine letting a ball roll downhill on the function surface
  - Now shake the surface, while the ball rolls,
  - Gradually reducing the amount of shaking



32

Physical Interpretation of Simulated Annealing

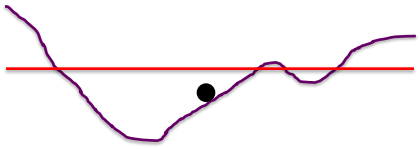
- A Physical Analogy:
  - Imagine letting a ball roll downhill on the function surface
  - Now shake the surface, while the ball rolls,
  - Gradually reducing the amount of shaking



33

Physical Interpretation of Simulated Annealing

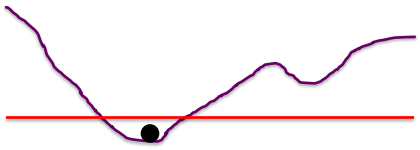
- A Physical Analogy:
  - Imagine letting a ball roll downhill on the function surface
  - Now shake the surface, while the ball rolls,
  - Gradually reducing the amount of shaking



34

Physical Interpretation of Simulated Annealing

- A Physical Analogy:
  - Imagine letting a ball roll downhill on the function surface
  - Now shake the surface, while the ball rolls,
  - Gradually reducing the amount of shaking

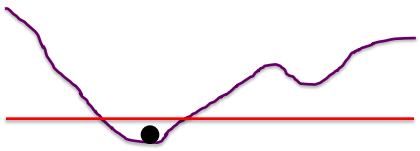


- Annealing = physical process of cooling a liquid → frozen
  - simulated annealing:
    - free variables are like particles
    - seek "low energy" (high quality) configuration
    - slowly reducing temp. T with particles moving around randomly

35

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"



36

Simulated Annealing in Practice

- Method proposed in 1983 by IBM researchers for solving VLSI layout problems (Kirkpatrick et al, *Science*, 220:671-680, 1983).

  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

  …Because T must be decreased very gradually in order to assure optimality

37

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 → finished
  - Else select *k* best from successors and repeat

38

Local Beam Search (contd)

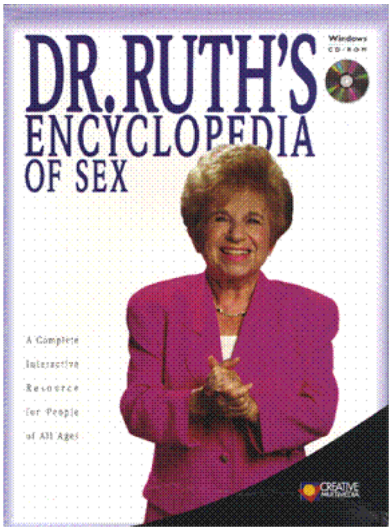
- Not the same as *k random-start searches run in parallel!*
- Searches that find good states recruit other searches to join them

- 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!

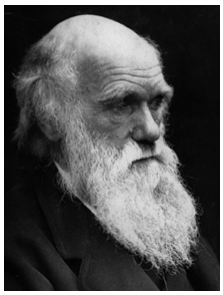
39

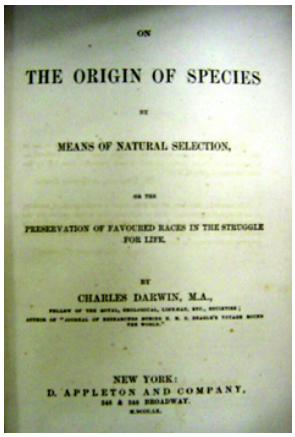
How to Make Search More … *Exciting?*



40

…And Be Scholarly!




ON
THE ORIGIN OF SPECIES
BY
MEANS OF NATURAL SELECTION,
OR THE
PRESERVATION OF FAVOURED RACES IN THE STRUGGLE FOR LIFE.
BY
CHARLES DARWIN, M.A.,
NEW YORK:
D. APPLETON AND COMPANY,
200 & 202 BROADWAY.

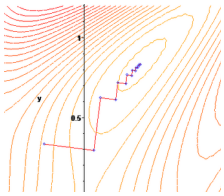
41

Genetic algorithms

- Local beam search, but…
  - A successor state is generated by ***combining two parent states***
- Start with *k* randomly generated states (population)
- A state is represented as a ***string*** over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher = better
- Produce the next generation of states by selection, crossover, and mutation

Gradient Descent

Assume we have a continuous function: $f(x_1,x_2,\ldots,x_N)$
and we want minimize over continuous variables X1,X2,..,Xn

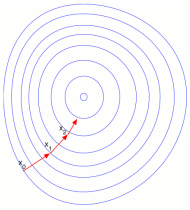


1. Compute the *gradients* for all *i:* $\partial f(x_1,x_2,\ldots,x_N) / \partial x_i$
2. Take a small step downhill in the direction of the gradient:

$$x_i \leftarrow x_i - \lambda \partial f(x_1,x_2,\ldots,x_N) / \partial x_i$$

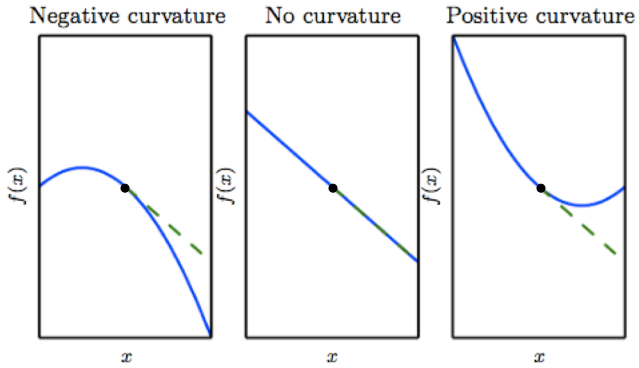
3. Repeat.

- How to select step size, λ
  - Line search: successively double
    - until *f* starts to increase again



46

Higher Order Derivatives

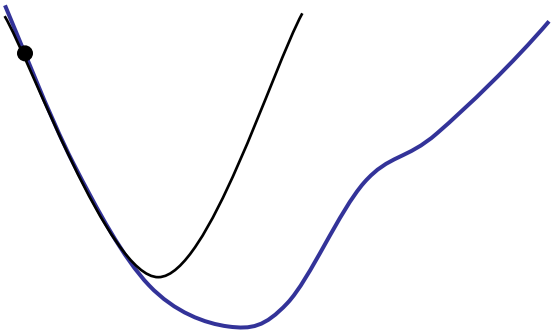


‹#›

Newton's Method

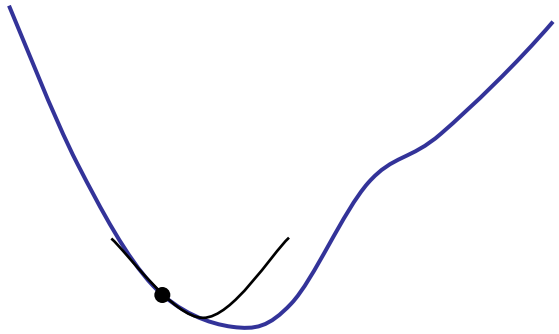
Assume function can be locally approximated with quadratic

Use both first & second derivatives



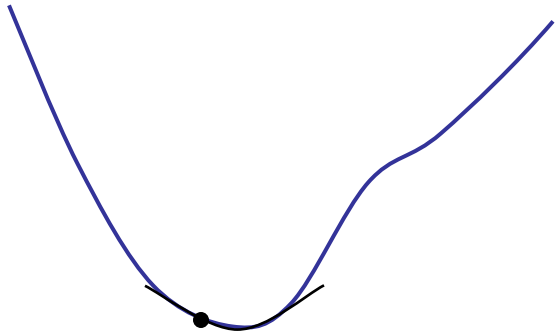
Slide from Princeton COS323 / Szymon Rusinkiewicz

Newton's Method



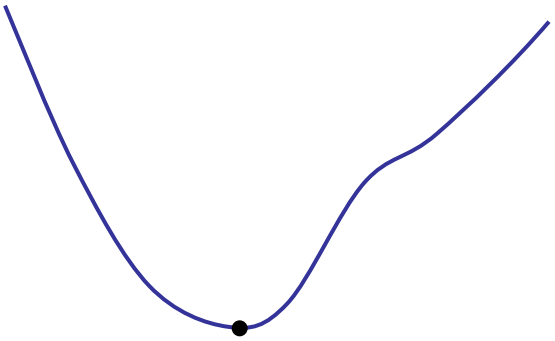
Slide from Princeton COS323 / Szymon Rusinkiewicz

Newton's Method



Slide from Princeton COS323 / Szymon Rusinkiewicz

Newton's Method



Slide from Princeton COS323 / Szymon Rusinkiewicz

Newton's Method

- At each step:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

- Requires 1st and 2nd derivatives
- Quadratic convergence

Slide from Princeton COS323 / Szymon Rusinkiewicz

Newton's Method in Multiple Dimensions

- Replace 1st derivative with gradient, 2nd derivative with Hessian

$$f(x,y)$$

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}$$

Slide from Princeton COS323 / Szymon Rusinkiewicz

Newton's Method in Multiple Dimensions

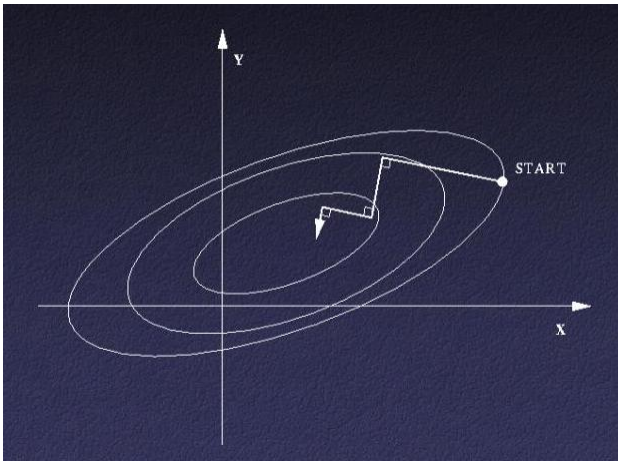
- Replace 1st derivative with gradient, 2nd derivative with Hessian
- So,

$$\vec{x}_{k+1} = \vec{x}_k - H^{-1}(\vec{x}_k)\nabla f(\vec{x}_k)$$

- Tends to be extremely fragile unless function very smooth and starting close to minimum

Slide from Princeton COS323 / Szymon Rusinkiewicz

Problem With Steepest Descent



Slide from Princeton COS323 / Szymon Rusinkiewicz