# CSE 573: Artificial Intelligence

## Winter 2019

## Local Search

Hanna Hajishirzi

With slides from
Dan Weld, Dan Klein, Stuart Russell, Luke Zettlemoyer

# Search for Optimization

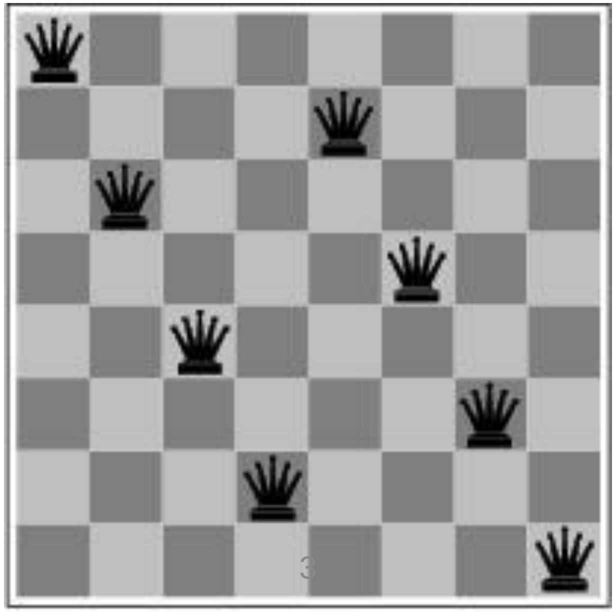
- Assign a utility function to every state
- Goal: find the state with the maximum utility

- Used in machine learning
- Used in neural networks

# Goal **State** *vs.* Path

- Previously: Search to find best path to goal
  - Systematic exploration of search space.

- Now: a state is solution to problem
  - For some problems path is irrelevant.
  - E.g., 8-queens



- Different algorithms can be used
  - Systematic Search
  - Local Search

# 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.
- Very memory efficient
  - only remember current state

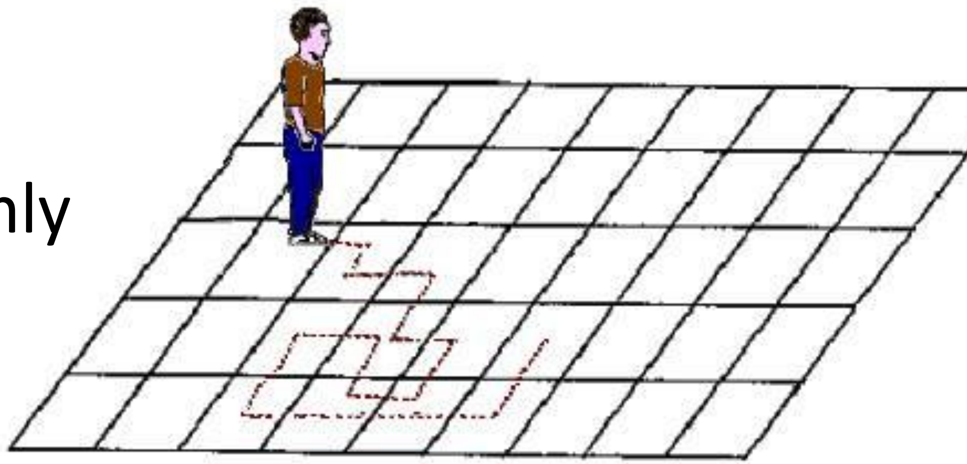
# Local Search and Optimization

- Local search
  - Keep track of single current state
  - Move only to “neighboring” state

    Defined by operators
  - Ignore previous states, path taken
- 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

# Search Methods

- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search

- Heuristic Search Methods
  - Best First / Greedy Search
  - A* Search

- Local Search
  - Hill Climbing
  - Beam Search
  - Gradient descent

# Beam Search

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

k* branch-factor

sorting

# 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)

- 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*
    **will be explained soon!*

- Observe the close analogy to natural selection!

# Search Methods

- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
- Heuristic Search Methods
  - Best First / Greedy Search
  - A* Search
- Local Search
  - Beam Search
  - Hill Climbing
  - Gradient descent

# Hill Climbing (Greedy Local Search)

- Idea
  - Always choose best child; no backtracking
  - Similar to beam-search (with queue =1)

# Hill-climbing (Greedy Local Search)

(minimum)

**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** (lowest)

*neighbor* ← a highest ≥ valued successor of *current*

**if** VALUE [*neighbor*] ≤ VALUE[*current*] **then return** STATE[*current*]

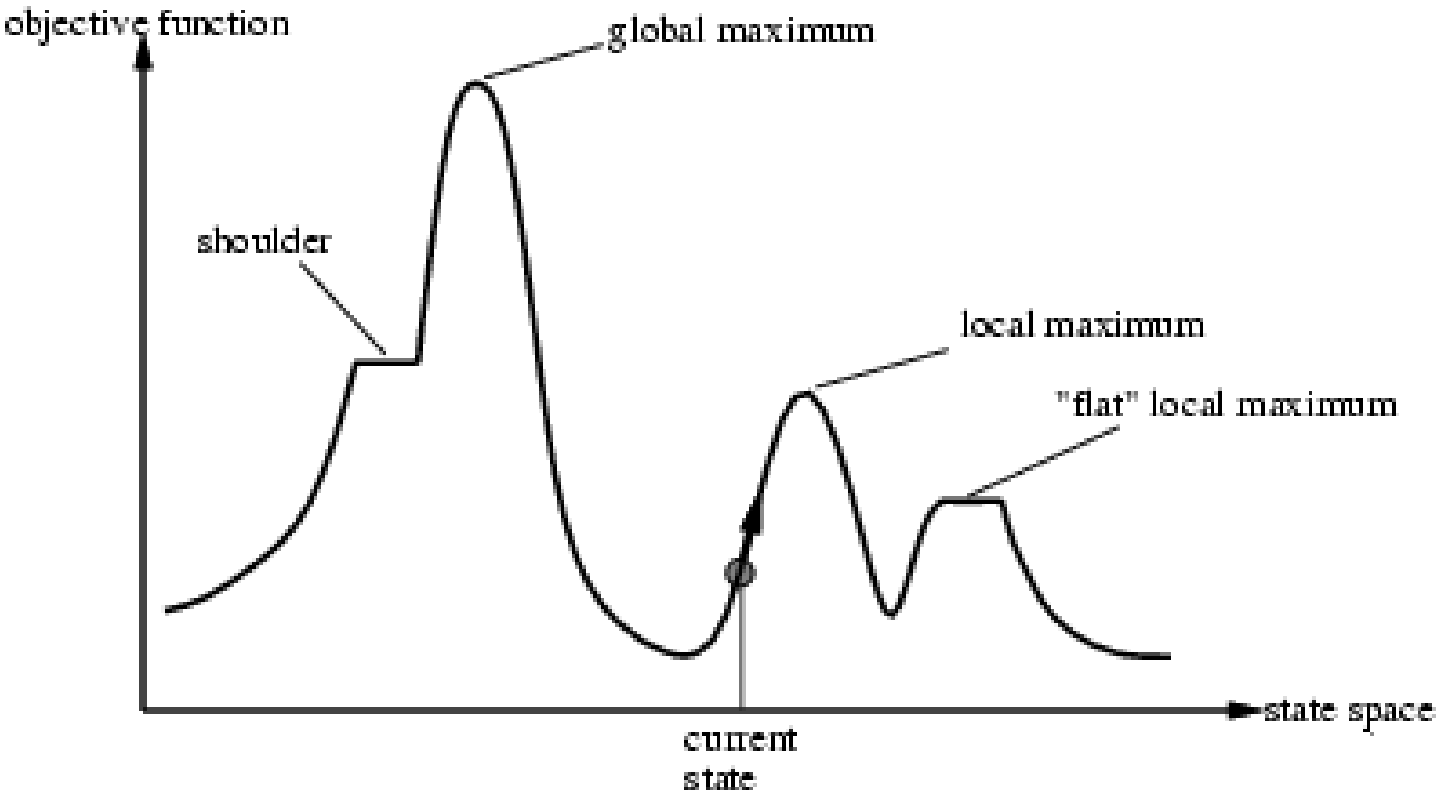
*current* ← *neighbor*

# 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

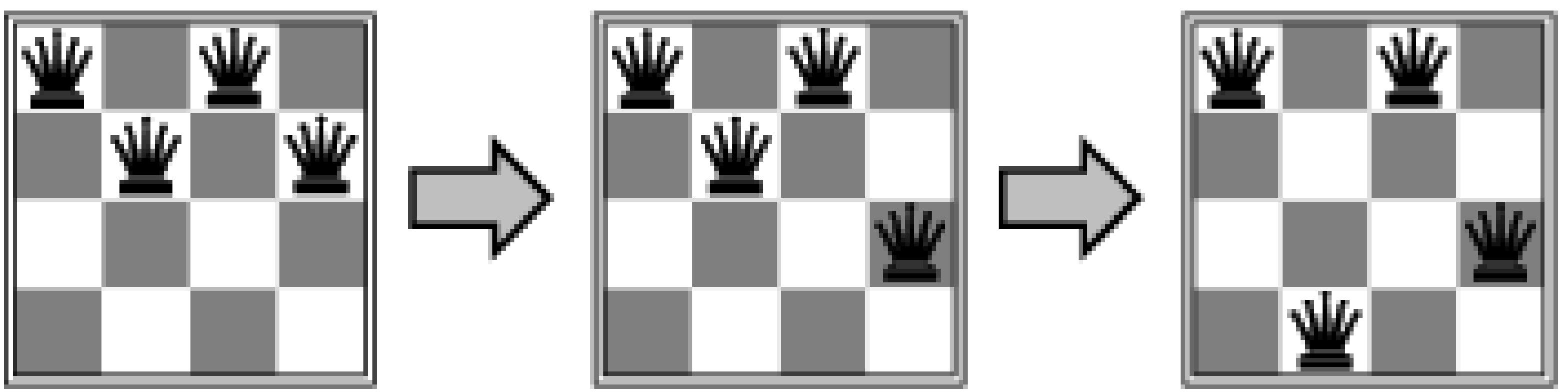
# “Landscape” of search




Hill Climbing gets stuck in local maxima

# Example: *n*-Queens

Objective: Put *n* queens on an *n* x *n* board with no two queens on the same row, column, or diagonal



Formulate the problem as an optimization.

# Our n-Queens (Local) Search Space

- State
  - All N queens on the board in some configuration
  - But each in a different column
- Successor function
  - Move single queen to another square in same column.

# Need Heuristic Function

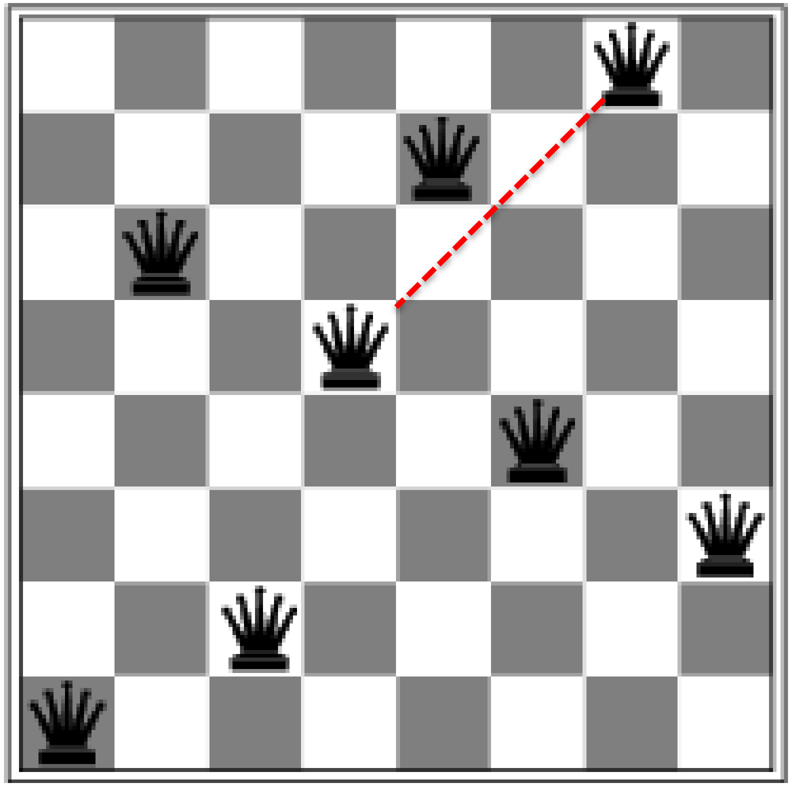
## Convert to Optimization Problem


- *h* = number of ***pairs*** of queens attacking each other
- *h = 17* for the above state

# Hill-climbing search: 8-queens

Result of hill-climbing in this case…

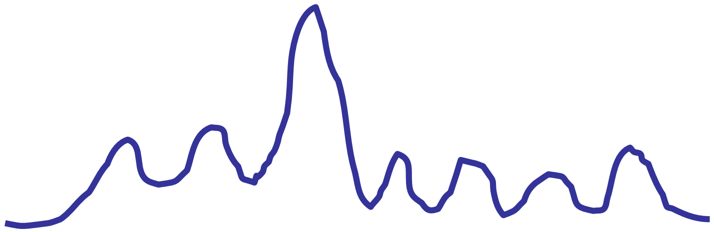
Bummer



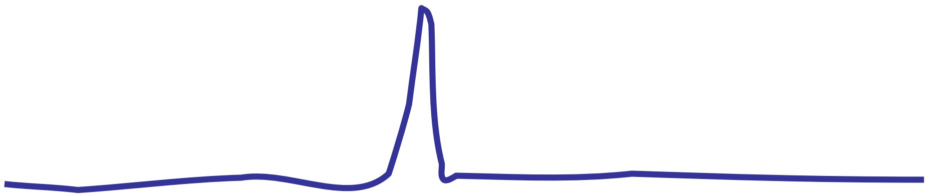
A local minimum with $h = 1$

# Hill Climbing Drawbacks

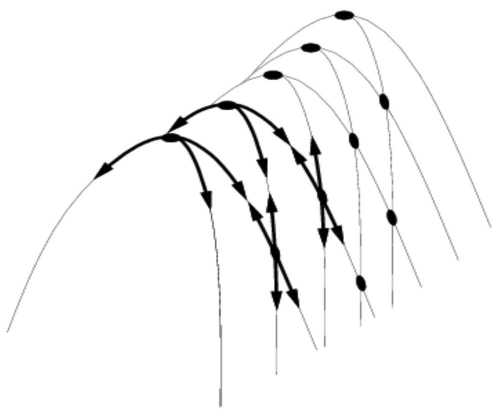
- Local minima



- Plateaus



- Diagonal ridges



# Hill Climbing Properties

- Not Complete
- Worst Case Exponential Time
- Simple, O(1) Space & Often Very Fast!

# 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)

# 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

# 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

# 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

# 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

Use this algorithm!

- 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?

# 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

# Application

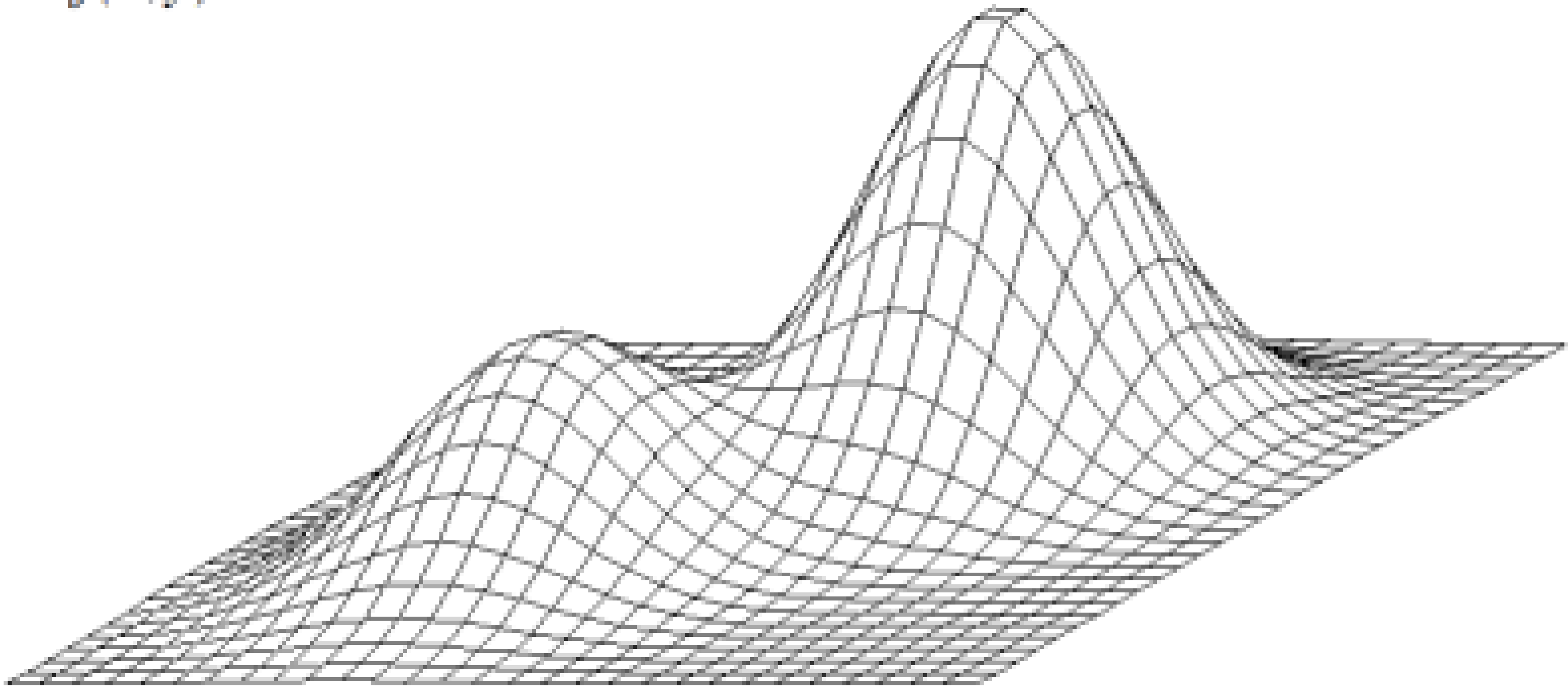
- In many machine learning algorithms:
  - Sentence generation
    - Generate one token at a time,
    - Keep n-best generated sentences

# Optimization of Continuous Functions

- Discretization
  - use hill-climbing
- Gradient descent
  - make a move in the direction of the gradient
    - gradients: closed form or empirical

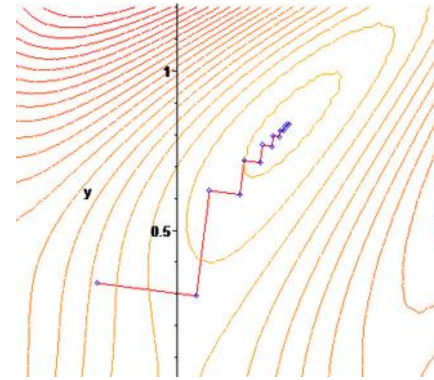
Is essential in most neural models

$$f(x,y)=e^{-(x^2+y^2)}+2e^{-((x-1.7)^2+(y-1.7)^2)}$$



# Gradient Descent

Assume we have a continuous function: $f(x_1,x_2,\ldots,x_N)$
and we want 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 λ
  - Line search: successively double
    - until *f* starts to increase again