Local Search

Ch. 4.1-4.2

1

Bookkeeping

- HW 1 in, HW 2 out tonight
  - (for sanity checking)
- Python lecture?
- Piazza notes:
  - Please make a new post with new questions
  - Only use the "respond" feature for stuff in that thread
  - Please tag posts properly

2

Today's Class

- Local Search
- Iterative improvement methods
- Hill climbing
- Simulated annealing
- Local beam search
- Genetic algorithms
- Online search

"If the **path** to the goal does not matter… [we can use] a single **current node** and move to neighbors of that node."

– R&N pg. 121

3

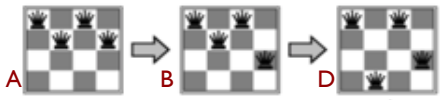
Local Search Algorithms

- Sometimes the path to the goal is irrelevant
  - Goal state itself is the solution
  - ∃ an **objective function** to evaluate states
- In such cases, we can use **local** search algorithms
- Keep a single "current" state, try to improve it

6

Local Search Example: *n*-Queens

- Put *n*-queens on an *n*×*n* board with no two queens on the same row, column, or diagonal



- Does it matter how we got to D?
- We only need the **state** – not the history/path
- Once we reach D, can forget A, B/C

7

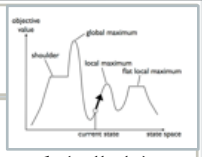
Local Search Algorithms

Very efficient! Why?

- Sometimes the path to the goal is irrelevant
  - Goal state itself is the solution
  - ∃ an **objective function** to evaluate states
- State space = set of "complete" configurations
  - That is, **all elements of a solution are present**
    - E.g., all the queens are on the board in some position
  - Find configuration satisfying constraints
- In such cases, we can use local search algorithms
- Keep a single "current" state, try to improve it

8

Landscapes



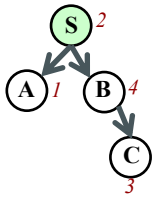
- Search graph can be a **landscape**
- Each node has **successor(s)** it can reach (called *s*)
  - Its children, unless there are loops
- Each successor has some "goodness" (desirability) according to the **objective** function
- $h(n) - h(s)$ is a positive, negative, or 0
- Positive is "uphill" (moving to a more desirable state)

Minor hassle:
Sometimes maximizing,
sometimes minimizing.

9

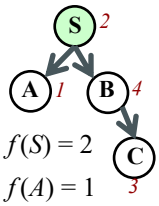
9

State Space (Landscape)



10

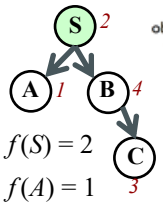
State Space (Landscape)



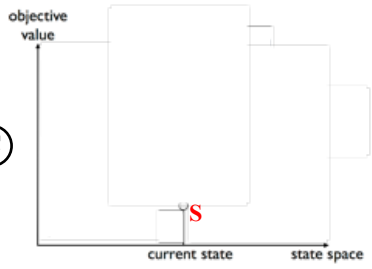
$f(S) = 2$
$f(A) = 1$
$f(B) = 4$
$f(C) = 3$

11

State Space (Landscape)

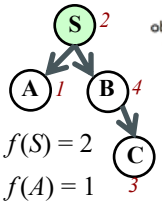


$f(S) = 2$
$f(A) = 1$
$f(B) = 4$
$f(C) = 3$

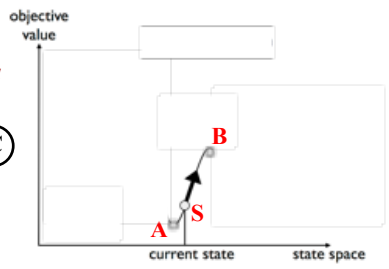


12

State Space (Landscape)

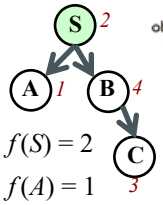


$f(S) = 2$
$f(A) = 1$
$f(B) = 4$
$f(C) = 3$

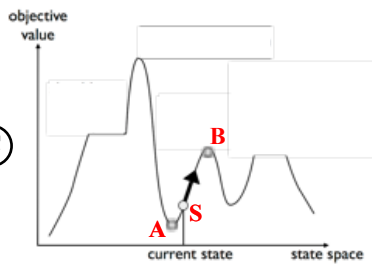


13

State Space (Landscape)

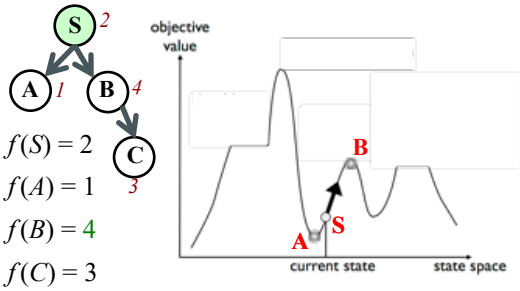


$f(S) = 2$
$f(A) = 1$
$f(B) = 4$
$f(C) = 3$



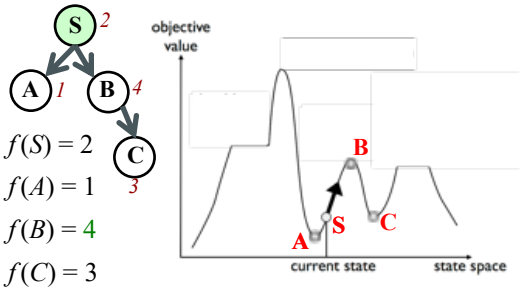
14

State Space (Landscape)



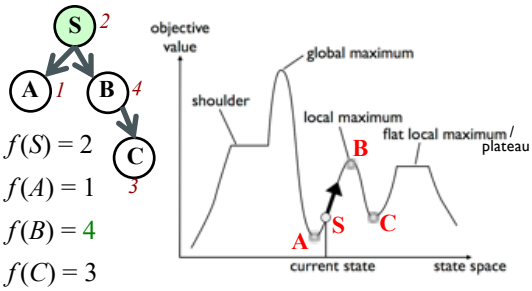
15

State Space (Landscape)



16

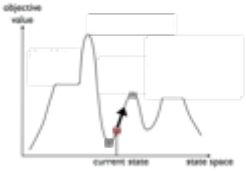
State Space (Landscape)



17

Iterative Improvement Search

- Start with an initial guess
- Gradually improve it until it is legal or optimal
- Some examples:
  - Hill climbing
  - Simulated annealing
  - Constraint satisfaction

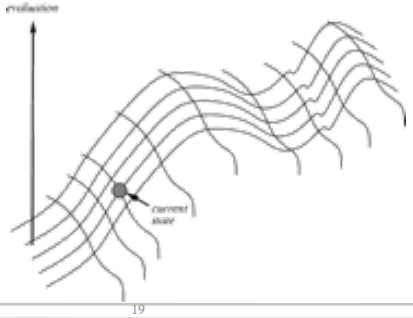


18

18

Hill Climbing on State Surface

- Concept: trying to reach the "highest" (most desirable) point (state)
- "Height" Defined by Evaluation Function



19

19

Hill Climbing Search

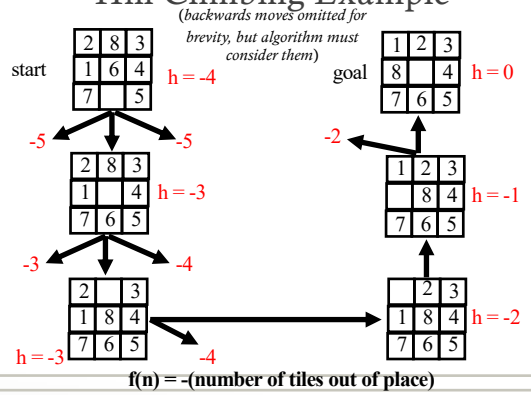
- Looks **one** step ahead to determine if any successor is "better" than current state, then moves to best choice
- If there exists a successor *s* for the current state *n* such that
  - *h*(*s*) > *h*(*n*) – it's better than where we are now
  - *h*(*s*) >= *h*(*t*) for all the successors *t* of *n* – and better than other choices

  then move from *n* to *s*. Otherwise, halt at *n*.
- A kind of Greedy search in that it uses *h*
  - But, does not allow backtracking or jumping to an alternative path
  - **Doesn't "remember" where it has been**
- Not *complete*
  - Search will terminate at local minima, plateaux, ridges.

20

20

Hill Climbing Example



21

Exploring the Landscape

- **Local Maxima**:
  - Peaks that aren't the highest point in the whole space
- **Plateaus:**
  - A broad flat region that gives the search algorithm no direction (do a random walk)
- **Ridges:**
  - Flat like a plateau, but with drop-offs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.

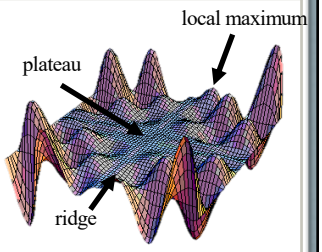


Image from: http://classes.yale.edu/fractals/CA/GA/Fitness/Fitness.html

22

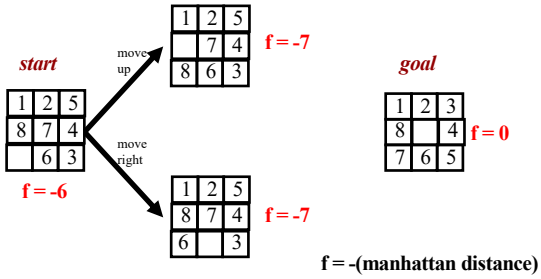
Drawbacks of Hill Climbing

- Problems: local maxima, plateaus, ridges
- Remedies:
  - **Random restart:** keep restarting the search from random locations until a goal is found.
  - **Problem reformulation:** reformulate the search space to eliminate these problematic features
- Some problem spaces are great for hill climbing; others are terrible

23

23

Example of a Local Optimum



24

24

Some Extensions of Hill Climbing

- Simulated Annealing
  - Escape local maxima by allowing *some* "bad" moves but gradually decreasing their frequency
- Local Beam Search
  - Keep track of *k* states rather than just one
  - At each iteration:
    - All successors of the *k* states are generated and evaluated
    - Best *k* are chosen for the next iteration

25

25

Some Extensions of Hill Climbing

- Stochastic (probabilistic) Beam Search
  - Chooses semi-randomly from "uphill" possibilities
  - "Steeper" moves have a higher probability of being chosen
- Random-Restart Climbing
  - Can actually be applied to any form of search
  - Pick random starting points until one leads to a solution
- Genetic Algorithms
  - Each successor is generated from two predecessor (parent) states

26

26

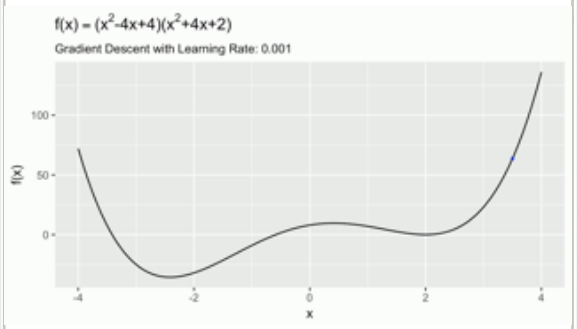
Gradient Descent (or Ascent)

- Length of downward "steps" proportional to negative of the gradient (slope) at the current state
  - "Steepest descent" → long "steps"
  - Jump to a node that is "farther away" if $f(\cdot)$ difference is large
- Gradient descent procedure for finding the arg_x *min f(x)*
  - choose initial x_0 randomly
  - repeat: $x_{i+1} \leftarrow x_i - \eta f'(x_i)$
  - until the sequence $x_0, x_1, \ldots, x_i, x_{i+1}$ converges
- Step sizeη(eta) is small (~0.1–0.05)
- Good for **differentiable, continuous** spaces

27

27

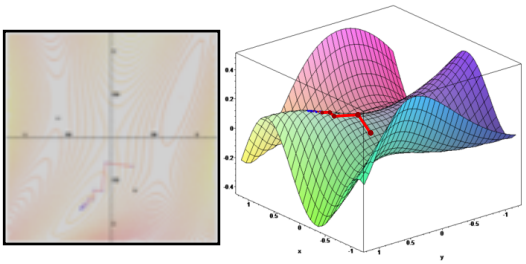
Gradient Descent



28 https://www.youtube.com/watch?v=ClotAJHZ3oE

28

Gradient Ascent / Descent

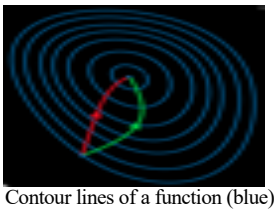


29 Images from http://en.wikipedia.org/wiki/Gradient_descent

29

Gradient Methods vs. Newton's Method

- Newton's method (calculus): $x_{i+1} \leftarrow x_i - \eta f'(x_i) / f''(x_i)$
- Newton's method uses 2nd order information (the second derivative, or, **curvature**) to take a more direct route to the minimum.
- The second-order information is more expensive to compute, but converges more quickly.



Contour lines of a function (blue)
- Gradient descent (green)
- Newton's method (red)

Images from http://en.wikipedia.org/wiki/Newton's_method_in_optimization

30

Simulated Annealing

- Conceptually: Escape **local maxima** by allowing some "bad" (locally counterproductive) moves but gradually decreasing their frequency
- Simulated annealing (SA): analogy between the way metal cools into a minimum-energy crystalline structure and the search for a minimum generally
  - In very hot metal, molecules can move fairly freely
  - They are slightly less likely to move out of a **stable structure**
  - As metal cools, molecules are more likely to stay

31

31

Simulated Annealing (II)

- Can avoid becoming trapped at local minima.
- Uses a random local search that:
  - Accepts "moves" that increase objective function *f*
  - **As well as** some that **decrease** it
- Uses a control parameter T
  - By analogy with the original application
  - Is known as the system "**temperature**"

  } freedom to make "bad" moves
- T starts out high and gradually decreases toward 0

32

32

Simulated Annealing (IV)

- $f(s)$ represents the quality of state n (high is good)
- A "bad" move from A to B is accepted with probability
  $P(\text{move}_{A\to B}) \approx e^{(f(B)-f(A))\ /\ T}$
  - $f(B) - f(A)$ is negative – 'bad' moves have relative probability <1
  - $f(B) - f(A)$ is positive – 'good' have relative probability >1
- Temperature
  - Higher temperature = more likely to make a "bad" move
  - As T tends to zero, this probability tends to zero
    - SA becomes more like hill climbing
  - If T is lowered slowly enough, SA is complete and admissible.
    - domain-specific
    - sometimes hard to determine

33

The Simulated Annealing Algorithm

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  static: *current*, a node
    next, a node
    T, a "temperature" controlling the probability 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*
  Δ*E* ← VALUE[*next*] – VALUE[*current*]
  if Δ*E* > 0 **then** *current* ← *next*
  else *current* ← *next* only with probability $e^{\Delta E/T}$

no history – we only keep 'current'. Local!

35

35

Local Beam Search

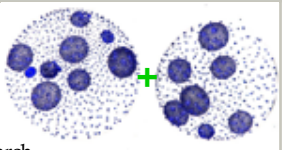
- Begin with k random states
  - k, instead of one, current state(s)
- Generate all successors of these states
- Keep the k best states across **all** successors
- Stochastic beam search
  - Probability of keeping a state is a function of its heuristic value
  - **More likely to keep "better" successors**

36

36

Genetic Algorithms

- The Idea:
  - New states generated by "mutating" a single state or "reproducing" (combining) two parent states
  - Selected for their **fitness**



- Similar to stochastic beam search
- Start with k random states (the **initial population**)
  - Encoding used for the "genome" of an individual strongly affects the behavior of the search
  - Must have some combinable representation of state spaces
  - Genetic algorithms / genetic programming are a research area

37

37

Tabu *Meta*-Search

- Problem: Local search "stuck" at local maxima
- Solution: relax search rules
  1. At each step, make a "bad" move if there's no good one
  2. Add "prohibitions" to discourage search from returning to recently visited areas of the search space ("cycling")
- How: Maintain a list of k previously visited states, and prevent the search from revisiting them
  - Why not always do this?

Russell & Norvig, en.wikipedia.org/wiki/Tabu_search

38

"Online" Search

- Interleave computation and action (search some, act some)
  - Exploration: Don't know outcomes of actions
  - So agent must try them!
- Competitive ratio = Path cost found* / Path cost that could be found**
  * On average, or in an adversarial scenario (worst case)
  ** If the agent knew transition functions and could use offline search
- Relatively easy **if** actions are reversible
- LRTA* (Learning Real-Time A*): Update $h(s)$ (in a state table) as new nodes are found

More about online search and nondeterministic actions next time…

39

39

Summary: Local Search (I)

- State space can be treated as a "landscape" of movement through connected states
- We're trying to find "high" (good) points
- **Best-first search**: a class of search algorithms where minimum-cost nodes are expanded first
- **Greedy search**: uses minimal estimated cost $h(n)$ to the goal state as measure of goodness
  - Reduces search time, but is neither complete nor optimal

40

40

Summary: Local Search (II)

- **Hill-climbing algorithms** keep only a single state in memory, but can get stuck on local optima
- **Simulated annealing** escapes local optima, and is complete and optimal given a "long enough" cooling schedule
- **Genetic algorithms** search a space by modeling biological evolution
- **Online search** algorithms are useful in state spaces with partial/no information

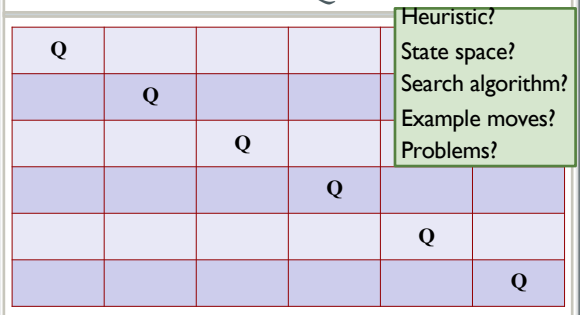
Questions?

41

41

Class Exercise: Local Search for *n*-Queens

Heuristic?
State space?
Search algorithm?
Example moves?
Problems?



(more on constraint satisfaction heuristics next time...)

42