Local Search
Ch. 4.1-4.2

Based on slides by Dr. Marie desJardin. Some material also adapted from slides by Dr. Rebecca Hutchinson @ Oregon State, and Dr. Matuszek @ Villanova University, which are based on Hwee Tou Ng at Berkeley, which are based on Russell at Berkeley. Some diagrams are based on AIMA.

1

Bookkeeping

- Upcoming: homework 1 due 9/16 at 11:59 PM
- Last time: informed (heuristic) search
  - Greedy search
  - A* and its variants
- Today:
  - Local search
  - Beginnings of constraint satisfaction?

2

Today's Class

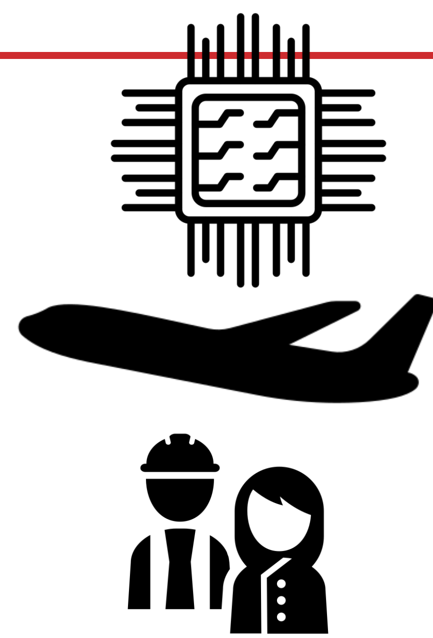
- Local Search
  - Search as "landscape"
  - Iterative improvement methods
  - Hill climbing
  - Simulated annealing
  - Local beam search
  - Genetic algorithms
  - Online search
- Intro to Constraint Satisfaction

> "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

Real-World Problems

- Suppose you had to solve VLSI layout problems (minimize distance between components, unused space, etc.)...
- Or schedule airlines...
- Or schedule workers with specific skill sets to do tasks that have resource and ordering constraints



Slide from Dr. Rebecca Hutchinson @ Oregon State

4

Local Search

- These problems are unlike the search problems previously:
  - The path to the goal is irrelevant
  - All you care about is the final configuration
  - These are often optimization problems in which you find the best state according to an **objective function** applied to a node (state)
- These problems are examples of **local search problems**
  - We care about the current state of the world

Slide from Dr. Rebecca Hutchinson @ Oregon State

5

Why Is This Hard?

- Lots of states (sometimes infinite)
- Most problems are NP-complete
- Objective function might be expensive
- But:
  - Use very little memory (usually constant)
  - Find reasonable (not usually optimal) solutions in large state spaces

Slide from Dr. Rebecca Hutchinson @ Oregon State

6

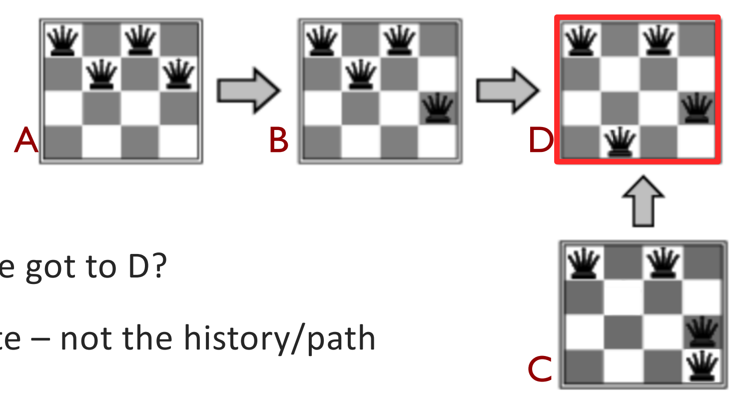
Local Search Algorithms

- 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

7

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

8

Local Search Algorithms

- ∃ an **objective function** to evaluate states
- State space = set of "complete" configurations
  - **All elements of a solution are present**
    - All the queens are on the board
    - All sudoku squares are filled
- Find configurations that satisfy constraints
- In such cases, we can use local search algorithms
  - Keep a single "current" state, try to improve it

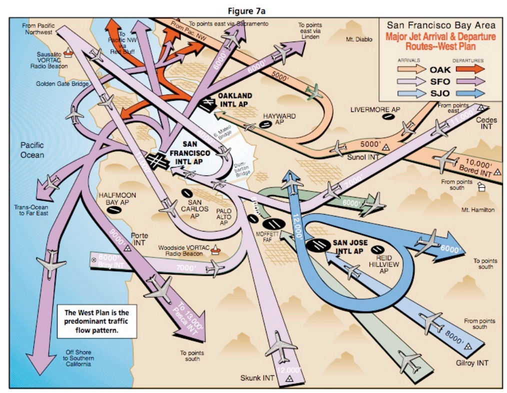


Image: telstarlogistics.typepad.com/telstarlogistics/2008/10/a-roadmap-to-our-highways-in-the-sky.html

9

Slide from Dr. Rebecca Hutchinson @ Oregon State

Local Search Algorithm Recipe

1. Start with initial configuration X
2. Evaluate its neighbors, i.e., the set of all states reachable in one move from X
3. Select one of its neighbors X*
4. Move to X* and repeat until the current configuration is satisfactory

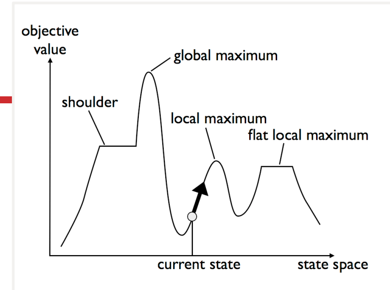
How you define the neighborhood is important.

Which neighbor you choose is important.

Some # of iterations, *or* some time, *or* until you can't move uphill

10

Landscapes



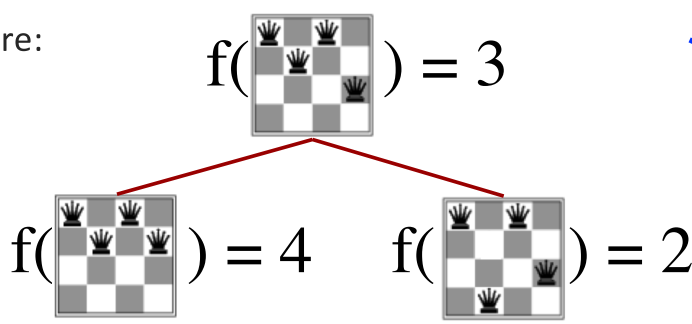
- Search graph can be a **landscape**
- Each node has **successor(s)** it can reach (called *s*)
  - Its children, unless there are loops
- Each successor has some "goodness" (desirability) according to the **objective** function
- $h(n) - h(s)$ is a positive, negative, or 0
- Want to go "uphill" (moving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

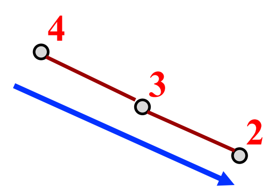
Minor hassle:
Sometimes maximizing,
sometimes minimizing.

11

N-Queens example

- Evaluation function: number of queens in conflict
- We are here:
- Some possible moves:



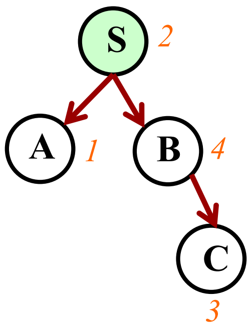


- We want to traverse the graph "downward" (minimize $f(n)$), so we choose the right-hand choice

12

State Space (Landscape)

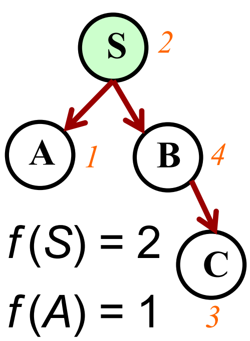
Maximizing (higher h(n) is better)



13

State Space (Landscape)

Maximizing (higher h(n) is better)



$f(S) = 2$

$f(A) = 1$

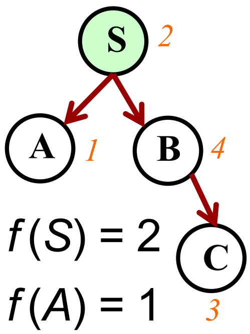
$f(B) = 4$

$f(C) = 3$

14

State Space (Landscape)

Maximizing (higher f(n) is better)

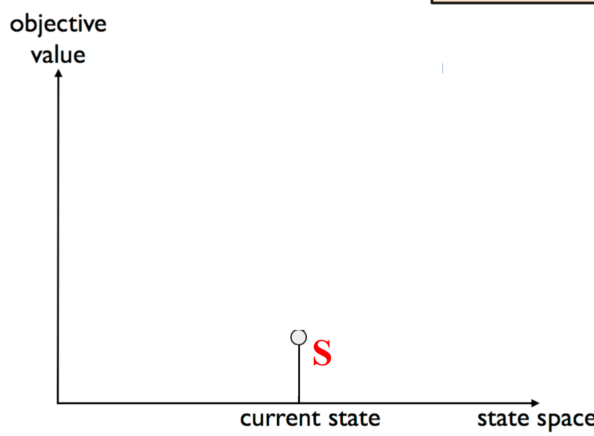


$f(S) = 2$

$f(A) = 1$

$f(B) = 4$

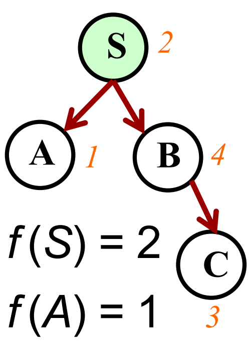
$f(C) = 3$



15

State Space (Landscape)

Maximizing (higher h(n) is better)

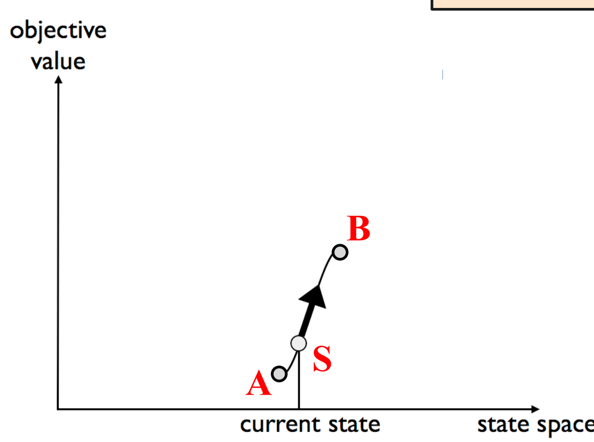


$f(S) = 2$

$f(A) = 1$

$f(B) = 4$

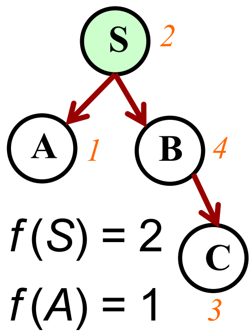
$f(C) = 3$



16

State Space (Landscape)

Maximizing (higher h(n) is better)

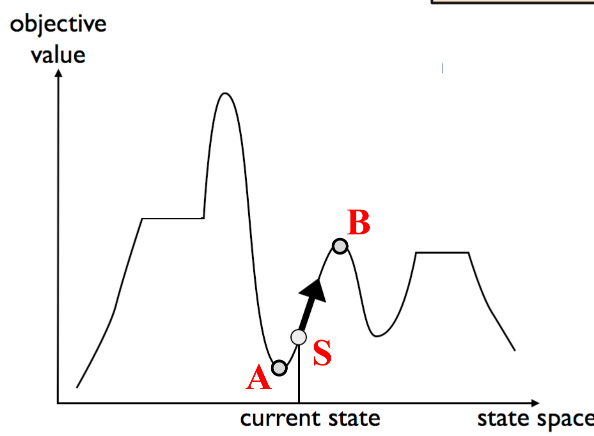


$f(S) = 2$

$f(A) = 1$

$f(B) = 4$

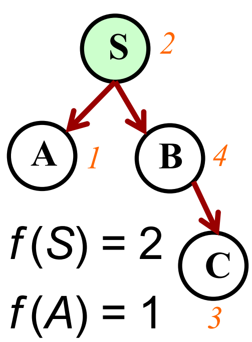
$f(C) = 3$



17

State Space (Landscape)

Maximizing (higher h(n) is better)

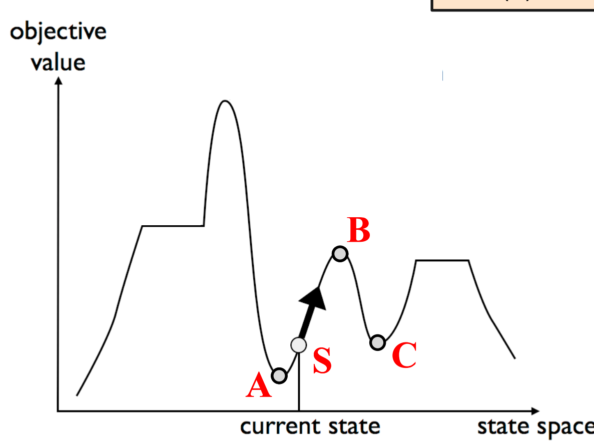


$f(S) = 2$

$f(A) = 1$

$f(B) = 4$

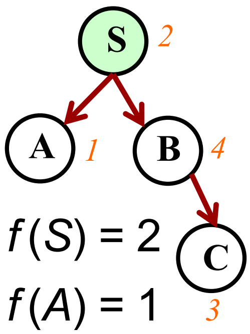
$f(C) = 3$



18

State Space (Landscape)

Maximizing (higher h(n) is better)

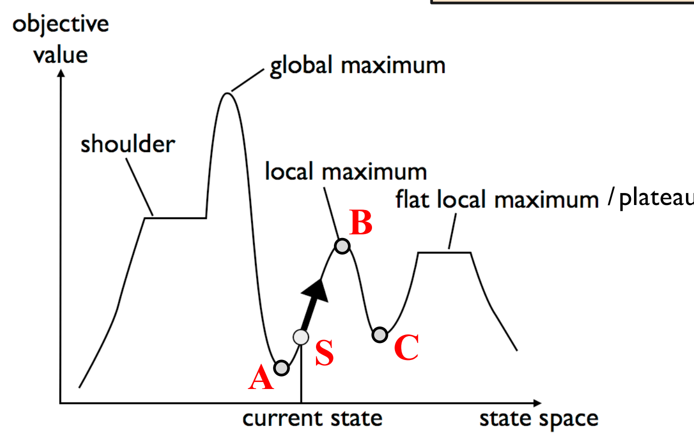


$f(S) = 2$

$f(A) = 1$

$f(B) = 4$

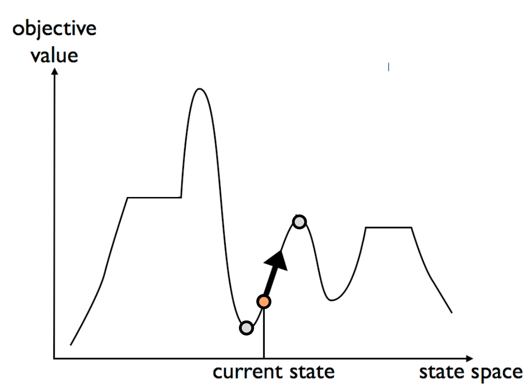
$f(C) = 3$



19

Iterative Improvement Search

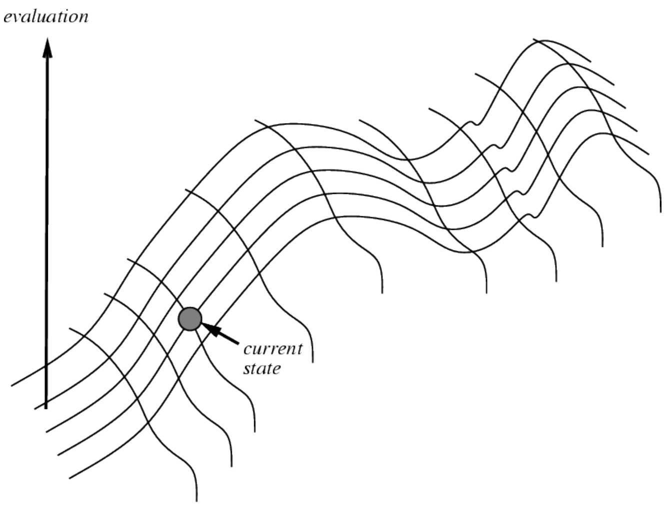
- Start with an initial guess
- Gradually improve it until it is legal or optimal
- Some examples:
  - Hill climbing
  - Simulated annealing
  - Constraint satisfaction



20

Hill Climbing on State Surface

- Starting at initial state X, keep moving to the neighbor with the highest objective function value greater than X's
- Concept: trying to reach the "highest" (most desirable) point (state)
- "Height" Defined by Evaluation Function
- Use the negative of heuristic cost function as the objective function



21

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* or *optimal*
  - Search will terminate at local minima, plateaus, ridges.

22

Hill Climbing Pseudocode

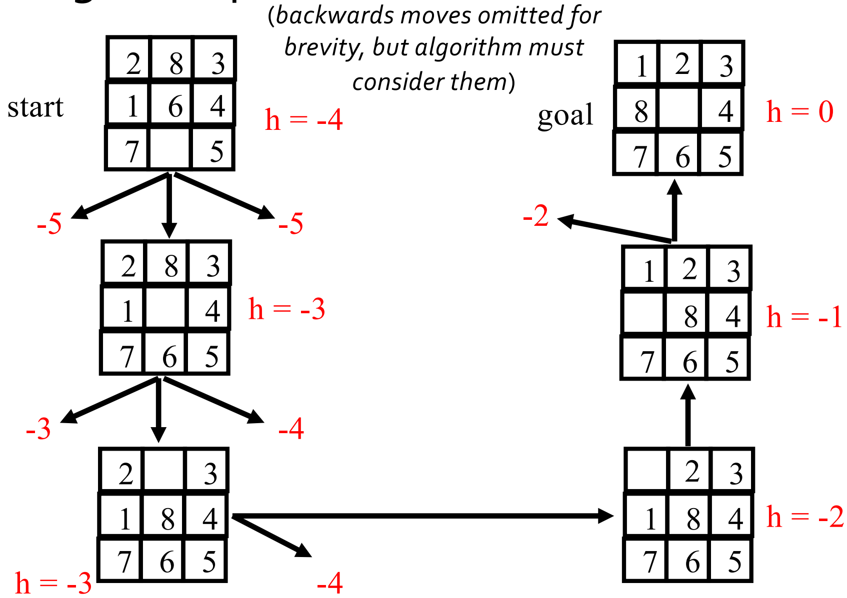
$X \leftarrow$ Initial configuration
Iterate:
  $E \leftarrow \text{Eval}(X)$
  $N \leftarrow \text{Neighbors}(X)$
  For each X_i in N
    $E_i \leftarrow \text{Eval}(X_i)$
  $E^* \leftarrow$ Highest E_i
  $X^* \leftarrow X_i$ with highest E_i
  If $E^* > E$
    $X \leftarrow X^*$
  Else
    Return X

Pretty simple, but will help us later…

23

Hill Climbing Example

(backwards moves omitted for brevity, but algorithm must consider them)



f(n) = -(number of tiles out of place)

24

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 and South may go down, but a step to the East and West is stable

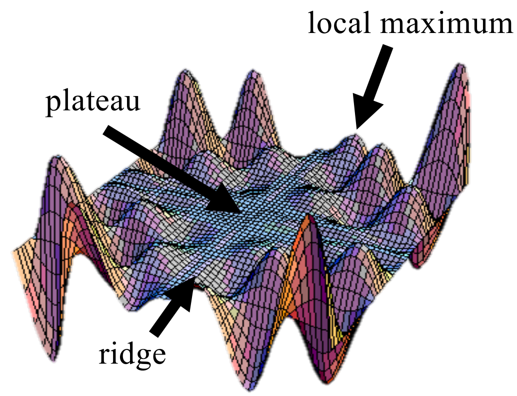
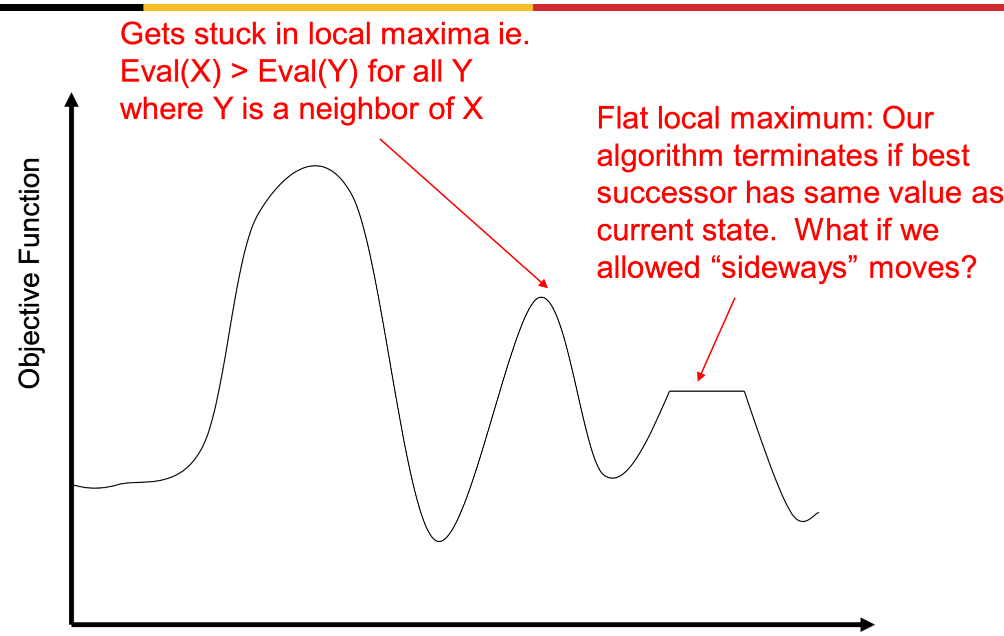


Image from: http://classes.yale.edu/fractals/CA/GA/Fitness/Fitness.html

25

Slide from Dr. Rebecca Hutchinson @ Oregon State

Local Maxima



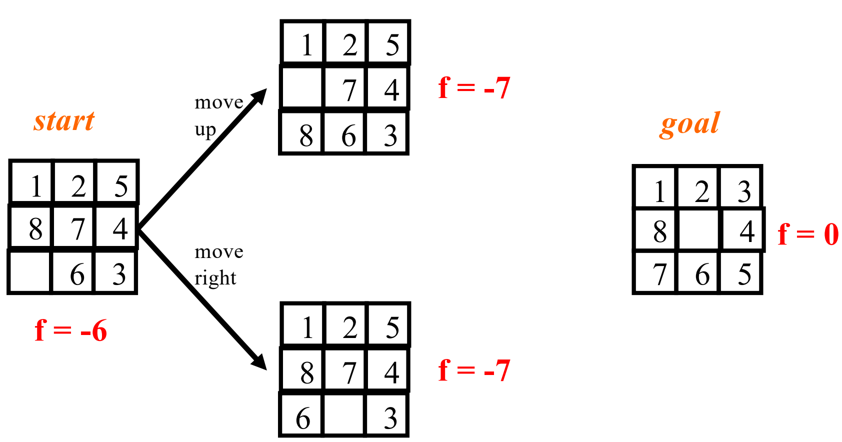
26

Drawbacks of Hill Climbing

- Problems: local maxima, plateaus, ridges
- Remedies:
  - **Random restart:** keep restarting the search from random locations until the 'best' goal is found
    - How do you know when to stop restarting?
  - **Problem reformulation:** reformulate the search space to eliminate these problematic features
    - Sometimes feasible, often not
- Some problem spaces are great for hill climbing; others are terrible
- Hill climbing is also **greedy local search** because you are greedily choosing the best-choice option in the neighborhood

27

Example of a Local Optimum



28

Some Extensions of Hill Climbing

- Random-Restart Climbing
  - Can actually be applied to any form of search
  - Pick random starting points until one leads to a solution
- First-choice hill climbing
  - Generate successors randomly until one is better than the current state
    - Our original n-queens example!
  - Good when state has many successors
- 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

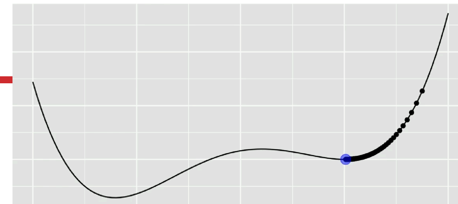
29

Some Extensions of Hill Climbing

- Simulated Annealing
  - Escape local maxima by allowing some “bad” moves but gradually decreasing their frequency
- Stochastic (probabilistic) Beam Search
  - Chooses semi-randomly from “uphill” possibilities
  - “Steeper” (better) moves have a higher probability of being chosen
- Genetic Algorithms
  - Each successor is generated from two predecessor (parent) states

30

The Problem



- Typical real-world problems have many (possibly an exponential number of) local maxima
- A hill-climbing algorithm that never makes "downhill" moves is vulnerable to getting stuck in a local maximum
  - Imagine a ball trying to reach the lowest state – it can get stuck in a "dip" that's above the lowest point
- A purely random walk that moves to a successor state whether it's "up" or "down" will eventually stumble on the global maximum, but is extremely inefficient

31

A possible solution

- Let's **combine hill climbing with random walk**
- Hill-climbing never makes a downhill move
  - What if we added **occasional** non-positive moves to hill-climbing to help it get out of local maxima?
  - This is the motivation for simulated annealing
- Conceptually: Escape **local maxima** by allowing some "bad" (locally counterproductive) moves but gradually decreasing their frequency
  - Our "ball" is allowed to bounce "up" occasionally, getting it out of "dips"

If you're curious, annealing is the process of hardening metals by heating them to a high temperature and then gradually cooling them. In very hot metal, molecules can move fairly freely; they are slightly less likely to move out of a stable structure, so as metal cools, molecules are more likely to stay in a strong matrix. So now you know.

32

Simulated Annealing

- Can avoid becoming trapped at local minima.
- Uses a random local search that:
  - Accepts "moves" that decrease objective function f
  - **As well as some that increase 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

33

Simulated Annealing Pseudocode

```
X ← Initial configuration
Iterate:
    E ← Eval(X)
    X' ← Randomly selected neighbor of X
    E' ← Eval(X')
    If E' ≥ E
        X ← X'
        E ← E'
    Else with probability p
        X ← X'
        E ← E'
```

So what's *p*?

Slide partially drawn from Dr. Rebecca Hutchinson @ Oregon State

34

Choosing *p*

- If *p* is too low, we don't make many 'downhill' moves
  - We might not get out of many local maxima
- If *p* is too high, we may be making too **many** suboptimal moves
- If *p* is constant, we might be making too many random moves when we are near the global maximum
- Solution: Decrease *p* over time
  - More counterproductive moves early, fewer as search goes on
  - Intuition: as search progresses, we are moving towards more promising areas and hopefully toward a global maximum

35

Choosing *p*

- Use a temperature parameter **T**
- If $E' \leq E$, accept the downhill move with probability $p = e^{-(E-E')/T}$
- Start with high temperature T
  - More downhill moves allowed at the start
- Decrease T gradually as iterations increase
  - Fewer downhill moves as we progress
- "Annealing schedule" describes how T decreases over time

36

Slide partially drawn from Dr. Rebecca Hutchinson @ Oregon State

Actual Simulated Annealing Pseudocode

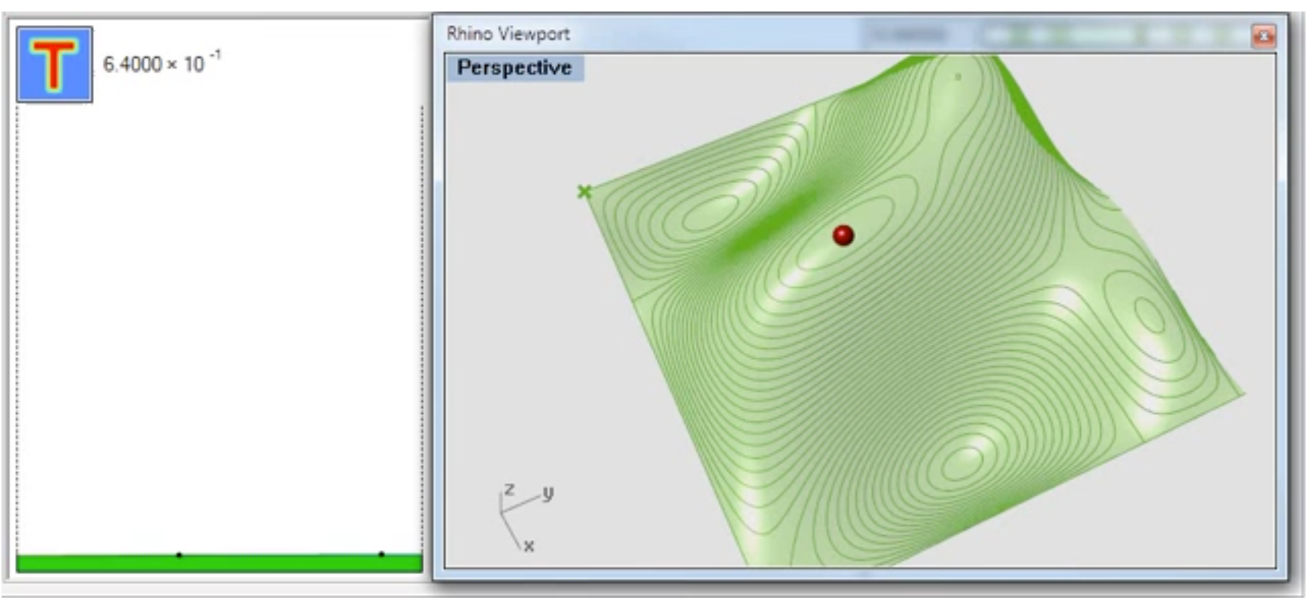
```
X ← Initial configuration
Iterate:
    Do K times:
        E ← Eval(X)
        X' ← Randomly selected neighbor of X
        E' ← Eval(X')
        If E' ≥ E
                X ← X'
                E ← E'
        Else with probability p = e^(-(E-E')/T)
                X ← X'
                E ← E'
    T = αT
```

Exponential cooling schedule
$T(n) = \alpha T(n-1)$ with $0 < \alpha < 1$

37

www.youtube.com/watch?v=VWtYLv-4oP0

Simulated Annealing: Examples



38

Simulated Annealing Summary

- $f(n)$ 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 low probability
  - $f(B) - f(A)$ is positive – 'good' moves have higher probability
- 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.

Lots of parameters to tweak ☹

- domain-specific
- sometimes hard to determine

39

Slide from Dr. Rebecca Hutchinson @ Oregon State

Local Beam Search

- Always keep *k*, instead of one, current state(s)
- Begin with *k* randomly chosen states
- 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

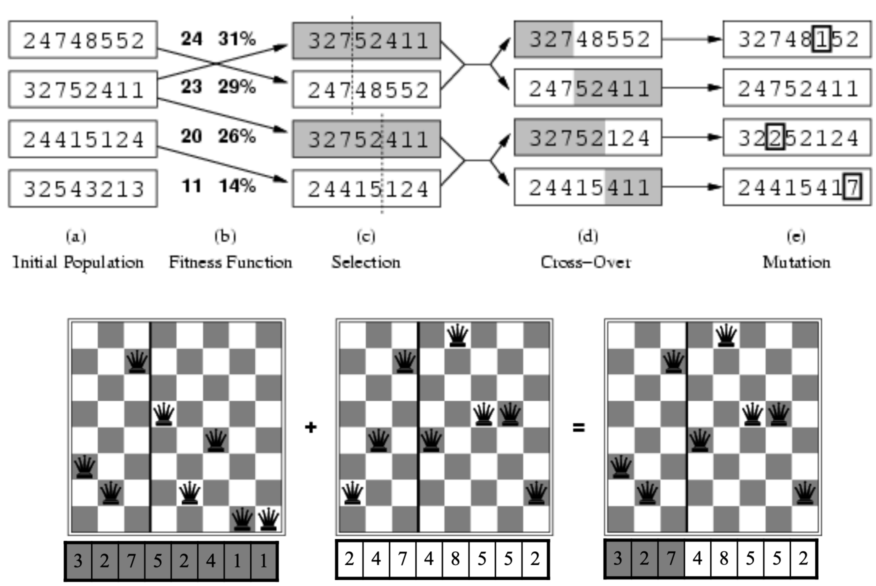
40

Local Beam Search

- How is this different from k random restarts in parallel?
  - Random-restart search: each search runs independently of the others
  - Local beam search: useful information is passed among the k parallel search threads
  - E.g. One state generates good successors while the other k-1 states all generate bad successors, then only the more promising states are expanded
- Disadvantage: all k states can become stuck in a small region of the state space
  - To fix this, use stochastic beam search
  - Stochastic beam search:
    - Doesn't pick best k successors
    - Chooses k successors at random, with probability of choosing a given successor being an increasing function of its value

41

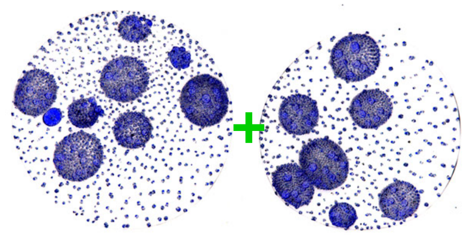
Genetic Algorithms



42

Genetic Algorithms

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

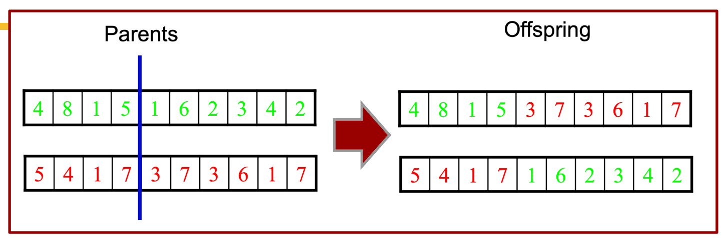


- Like natural selection in which an organism creates offspring according to its fitness for the environment
- Over time, population contains individuals with high fitness

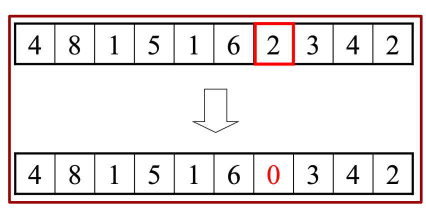
43

Genetic Algorithms

- 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



reproduction



mutation

Slide partially drawn from Dr. Rebecca Hutchinson @ Oregon State

44

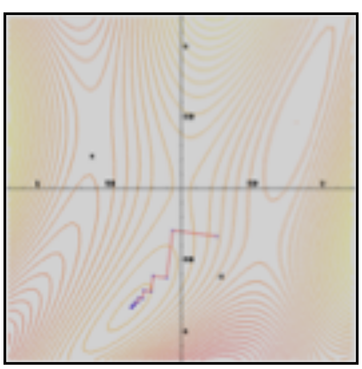
GA Implementation

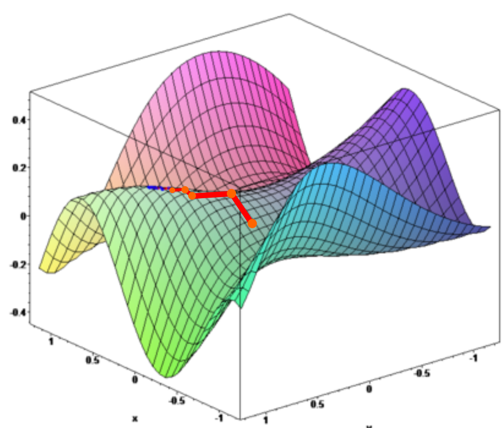
- Initially, population is diverse, crossover produces big changes from parents
- Over time, individuals become similar and crossover doesn't produce such a big change
- Crossover is the big advantage
    - Preserves a big block of "genes" that have evolved independently to perform useful functions

Slide partially drawn from Dr. Rebecca Hutchinson @ Oregon State

45

Gradient Ascent / Descent





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

46

Hill climbing: Discrete spaces

$X \leftarrow$ Initial configuration
Iterate:
  $E \leftarrow$ Eval(X)
  $N \leftarrow$ Neighbors(X)
  For each X_i in N
    $E_i \leftarrow$ Eval(X_i)
  $E^* \leftarrow$ Highest E_i
  $X^* \leftarrow X_i$ with highest E_i
  If $E^* > E$
    $X \leftarrow X^*$
  Else
    Return X

- In discrete spaces, the number of neighbors is finite.
- What if there is a continuous space of possible moves leading to an infinite number of neighbors?

47

Local Search in Continuous Spaces

- Almost all real world problems involve continuous state spaces
- The main technique to find a local minimum is called **gradient descent** (or gradient ascent if you want to find the maximum)
- To perform local search in continuous state spaces, you need calculus
  - What is the gradient of a function f(x)?

$$\nabla f(x) = \frac{\partial}{\partial x} f(x)$$

  - $\nabla f(x)$ (the gradient) represents the direction of the steepest slope
  - $|\nabla f(x)|$ (the magnitude of the gradient) tells you how big the steepest slope is

Slide partially drawn from Dr. Rebecca Hutchinson @ Oregon State

48

Gradient Descent

- Suppose we want to find a local minimum of a function f(x)
  - (Which we do—the continuous-space analog of a minimum)
- We use the gradient descent rule:

$$x \leftarrow x - \alpha \nabla \mathrm{f}(x)$$

- 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 $f(\bullet)$ difference is large

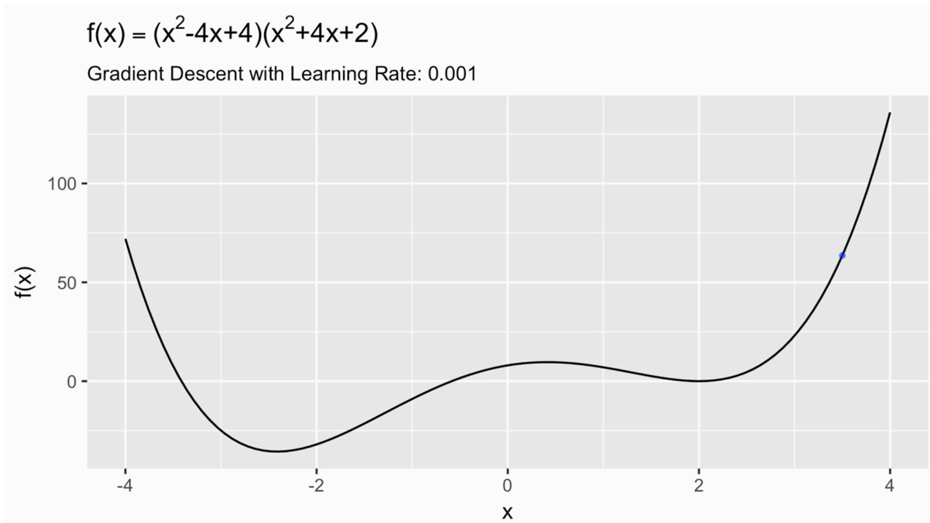
49

Gradient Descent (or Ascent)

- 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 size η (eta) is small (~0.1–0.05)
- Good for **differentiable, continuous** spaces
- Why not just calculate the global optimum using $\nabla f(x) = 0$?
  - May not be able to solve this equation in closed form
  - If you can't solve it globally, you can still compute the gradient locally (like we are doing in gradient descent)

50

Gradient Descent



https://www.youtube.com/watch?v=CIotAJHZ3oE

51

Weaknesses of Gradient Descent

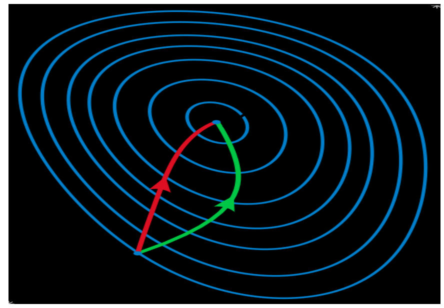
- Must pick α
  - If too large, gradient descent overshoots the optimum point
  - If too small, gradient descent requires too many steps and will take a very long time to converge
- Can be very slow to converge to a local optimum, especially if the curvature in different directions is very different
- Good results depend on the value of the learning rate α
- What if the function f(x) isn't differentiable at x?

Slide partially drawn from Dr. Rebecca Hutchinson @ Oregon State

52

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

53

"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

55

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

56

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?

57

Class Exercise: Local Search for *n*-Queens

Q					
	Q				
		Q			
			Q		
				Q	
					Q

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

58

Class Exercise: Moving

- You have to move from your old apartment to your nev the following:
  - A list $L = \{a_1, a_2, \ldots, a_n\}$ of n items, each with a size $s(a_i) > 0$.
  - M moving boxes, each with a box capacity C (assume MC exc sizes of your items)
  - You can put arbitrary items into a box as long as the sum of t exceed the box capacity C
- Your job is to pack your stuff into as few boxes as possible
- Formulate this as a **local search problem**

States?
Neighborhood?
Evaluation function?
How to avoid local maxima?

59