CS 331: Artificial Intelligence
Local Search 1

1

Tough 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

2

1

What do these problems have in common?

- These problems are unlike the search problems from last class:
  - 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
- These problems are examples of local search problems

3

Local Search Problems

- Given a set of states $S = \{X_1, \ldots, X_m\}$
- And an objective function $Eval(X_i)$ that returns the "goodness" of a state
- Find the state X^* that maximizes the objective function

Note: Sometimes $Eval(X_i)$ is a cost function instead of an objective function. In this case, we want to find the state X^* that minimizes the cost function. We will deal with objective functions in these slides but it's easy to just flip the signs and think of cost functions.

4

1D State-Space Landscape

Objective Function

Global maximum

Local maximum

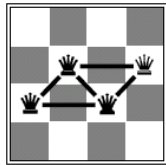Flat local maximum

State Space
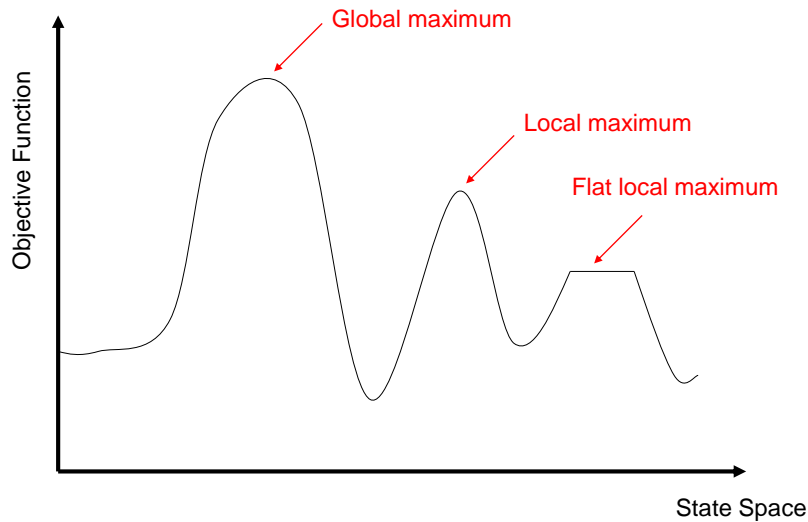
5

Why is this hard?

- Lots of states (sometimes an infinite number)
- Most of these problems are NP-complete
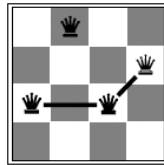- Objective function might be expensive

But:

- Use very little memory (usually constant)
- Find reasonable (but usually not optimal) solutions in large or infinite state spaces
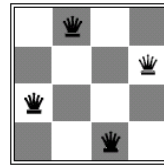
6

Example: n-queens

Eval(X) = 5   Eval(X) = 2   Eval(X) = 0

- State X = placement of the queens on the board
- Eval(X) = # of pairs of queens attacking each other
- Want to minimize Eval(X)

7

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

   How you define the neighborhood is important to the performance of the algorithm.

3. Select one of its neighbors X*

   Which neighbor you select is also very important

4. Move to X* and repeat until the current configuration is satisfactory

   Common stopping criteria:
   - Run for specified # of iterations
   - Run for specified time
   - Run until you can't move uphill

4

Outline

1. Hill-climbing
2. Simulated Annealing
3. Beam Search
4. Genetic Algorithms
5. Gradient Descent

9

1. Hill-climbing

10

Hill-climbing (Intuitively)

- "…resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia."

- Starting at initial state X, keep moving to the neighbor with the highest objective function value greater than X's.

11

Hillclimbing Pseudocode

$X \leftarrow$ 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

12

Example: 8-queens

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

- State: a board with 8 queens
- Successor function: move a single queen to any other square in same column
- Heuristic cost function (h): # of pairs of queens attacking each other

These numbers are the value of the heuristic cost function if you move the queen in that column to that square
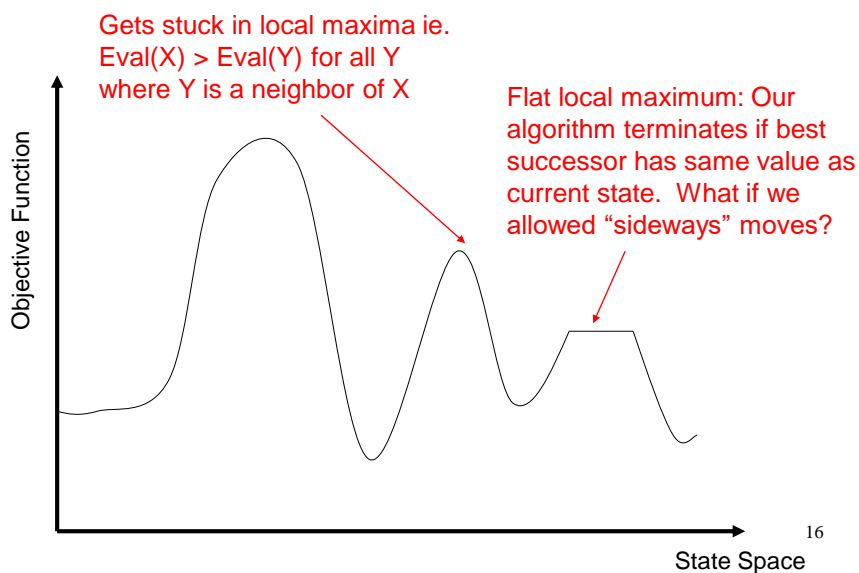
Example: 8-queens
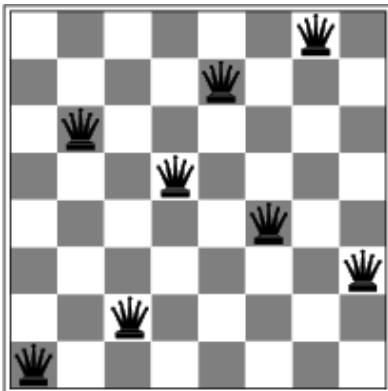
- Local minimum with h=1
- Note: at global minimum, h=0

14

More on hill-climbing

- Hill-climbing also called greedy local search
- Greedy because it takes the best immediate move
- Greedy algorithms often perform quite well

15

Problems with Hill-climbing

Gets stuck in local maxima ie.
Eval(X) > Eval(Y) for all Y
where Y is a neighbor of X

Flat local maximum: Our
algorithm terminates if best
successor has same value as
current state. What if we
allowed "sideways" moves?

Objective Function

State Space

16

8

Neighborhoods

- Recall that we said that defining the neighborhood correctly is critical to the performance of the algorithm
- Large neighborhood: Lots of evaluations to do at each state but better chance of finding a good maximum
- Small neighborhood: Fewer evaluations at each state but can potentially get stuck in more local maxima

17

Variants of Hill-climbing

- Stochastic hill climbing:
  - Chooses at random among the uphill moves
  - Probability of selection varies with steepness
- First-choice hill climbing:
  - Generates successors randomly until one is generated that is better than the current state
  - Good when state has many successors
- Random-restart hill-climbing
  - Good for dealing with local maxima
  - Conduct a series of hill-climbing searches from randomly generated initial states
  - Stop when a goal state is found (or until time runs out, in which case return the best state found so far)

Which variant?

- Depends on the state-space landscape (which is difficult to know a priori)
- Typical real-world problems have an exponential number of local maxima
- But hill-climbing still works reasonably well

What if a state-space landscape looked like this guy's back?

19

Simulated Annealing

- Hill-climbing never makes a downhill move
- What if we added some random moves to hill-climbing to help it get out of local maxima?
- This is the motivation for simulated annealing

If you're curious, **annealing** refers to the process used to harden metals by heating them to a high temperature and then gradually cooling them

20

2. Simulated Annealing

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'

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'

This part is different from hillclimbing

23

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'

How do you set p?

24

Setting p

- What if p is too low?
  - We don't make many downhill moves and we might not get out of many local maxima
- What if p is too high?
  - We may be making too many suboptimal moves
- Should p be constant?
  - We might be making too many random moves when we are near the global maximum

25

Setting p

- **Decrease p as iterations progress**
  - Accept more downhill moves early, accept fewer as search goes on
  - Intuition: as search progresses, we are moving towards more promising areas and quite likely toward a global maximum
- **Decrease p as E-E' increases**
  - Accept fewer downhill moves if slope is high
  - See next slide for intutition

26

Decreasing p as E-E' increases

E

E'
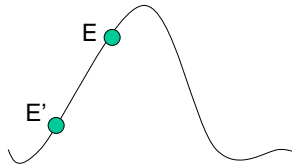
E-E' is large: we are likely moving towards a sharp (interesting) maximum so don't move downhill too much

E

E'

E-E' is small: we are likely moving towards a smooth (uninteresting) maximum so we want to escape this local maximum

27

Setting p

- Needs 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 (less downhill moves allowed)
- Annealing schedule describes how T is decreased at each step

28

Complete Simulated Annealing Pseudocode

X ← Initial configuration
Iterate:
   Do K times:
      $E \leftarrow Eval(X)$
      X' ← Randomly selected neighbor of X
      $E' \leftarrow Eval(X')$
      If $E' \geq E$
         X ← X'
         E ← E'
      Else with probability $p = e^{-(E-E')/T}$
         X ← X'
         E ← E'
   $T = \alpha T$

29

Complete Simulated Annealing Pseudocode

X ← Initial configuration
Iterate:
   Do K times:

> We keep the temperature fixed and we iterate K times

      $E \leftarrow Eval(X)$
      X' ← Randomly selected neighbor of X
      $E' \leftarrow Eval(X')$
      If $E' \geq E$
         X ← X'
         E ← E'
      Else with probability $p = e^{-(E-E')/T}$
         X ← X'
         E ← E'
   $T = \alpha T$

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

Convergence

- If the schedule lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1
- In practice, reaching the global optimum could take an enormous number of iterations

31

The fine print…

- Design of neighborhood is critical
- Lots of parameters to tweak e.g. α, K, initial temperature
- Simulated annealing is usually better than hillclimbing if you can find the right parameters

32

3. Beam Search

33

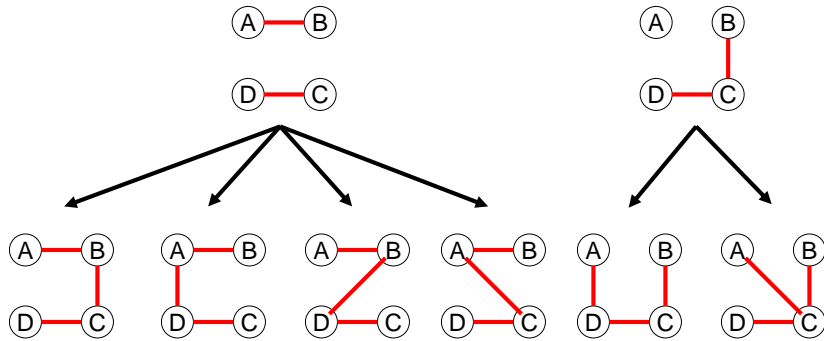# Local Beam Search

Travelling Salesman Problem

```
(A)——(B)          (A)   (B)
                         |
(D)——(C)          (D)——(C)
```

Keeps track of k states rather than just 1.
k=2 in this example. Start with k randomly
generated states.

34

17

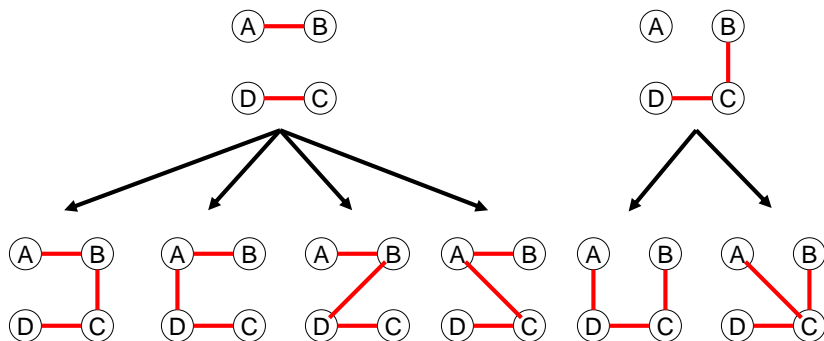# Local Beam Search Example

Travelling Salesman Problem (k=2)

Generate all successors of all the k states

35

Local Beam Search Example
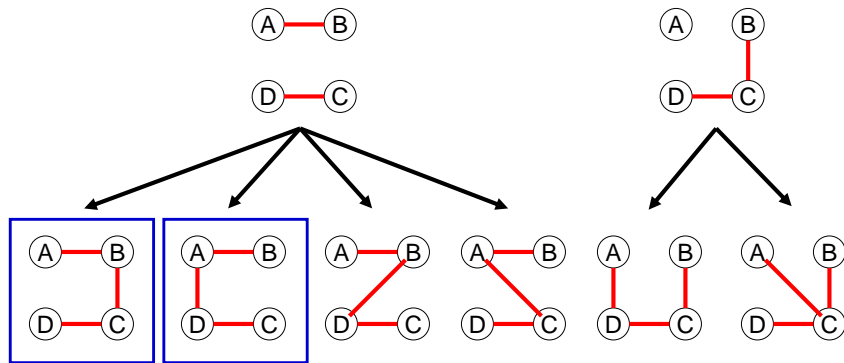
Travelling Salesman Problem (k=2)

None of these is a goal state so we continue

36

Local Beam Search Example

Travelling Salesman Problem (k=2)

Select the best k successors from the **c**omplete list

Local Beam Search Example

Travelling Salesman Problem (k=2)

Repeat the process until goal found

38

19

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 the more promising states are expanded
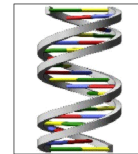
39

Local Beam Search

- 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

40

4. Genetic Algorithms

41

Genetic Algorithms

- Like natural selection in which an organism creates offspring according to its fitness for the environment
- Essentially a variant of stochastic beam search that combines two parent states (just like sexual reproduction)
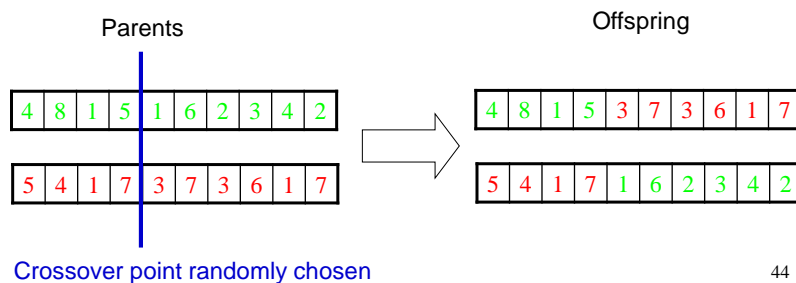- Over time, population contains individuals with high fitness

42

Definitions

- Fitness function: Evaluation function in GA terminology
- Population: k randomly generated states (called individuals)
- Individual: String over a finite alphabet

"gene"

| 4 | 8 | 1 | 5 | 1 | 6 | 2 | 3 | 4 | 2 |

"chromosome"
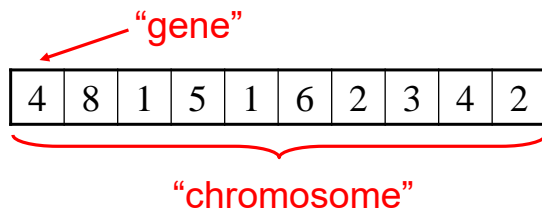
43

Definitions

- Selection: Pick two random individuals for reproduction
- Crossover: Mix the two parent strings at the crossover point

Parents

Offspring

| 4 | 8 | 1 | 5 | 1 | 6 | 2 | 3 | 4 | 2 |

| 5 | 4 | 1 | 7 | 3 | 7 | 3 | 6 | 1 | 7 |

| 4 | 8 | 1 | 5 | 3 | 7 | 3 | 6 | 1 | 7 |

| 5 | 4 | 1 | 7 | 1 | 6 | 2 | 3 | 4 | 2 |

Crossover point randomly chosen

44

Definitions

- Mutation: randomly change a location in an individual's string with a small independent probability

4	8	1	5	1	6	2	3	4	2

⬇

4	8	1	5	1	6	0	3	4	2

Randomness aids in avoiding small local extrema

45

GA Overview

Population = Initial population
Iterate until some individual is fit enough or enough time has elapsed:
    NewPopulation = Empty
    For 1 to size(Population)
        Select pair of parents (P_1, P_2) using Selection(P,Fitness Function)
        Child C = Crossover(P_1, P_2)
        With small random probability, Mutate(C)
        Add C to NewPopulation
    Population = NewPopulation
Return individual in Population with best Fitness Function

46

GA Overview

Population = Initial population

Iterate until some individual is fit enough or enough time has elapsed:

    NewPopulation = Empty

    For 1 to size(Population)

        Select pair of parents (P_1, P_2) using Selec

        Child C = Crossover(P_1, P_2)

        With small random probability, Mutate(C)

        Add C to NewPopulation

    Population = NewPopulation

Return individual in Population with best Fitness Function

> This pseudocode only produces one child. Could also do a variant like before where we produce 2 children

47

Lots of variants
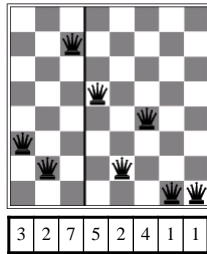
- Variant 1: Culling - individuals below a certain threshold are removed

- Variant 2: Selection based on:

$$P(X \; selected) = \frac{Eval(X)}{\sum_{Y \in Population} Eval(Y)}$$
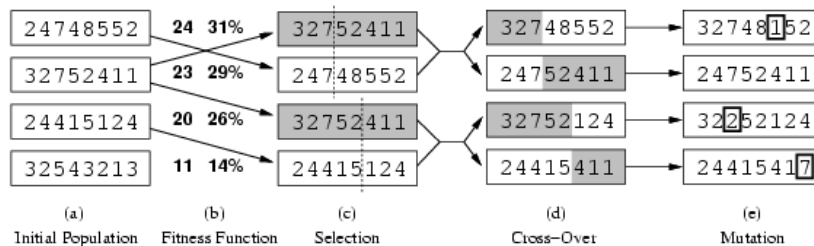
48

Example: 8-queens

- Fitness Function: number of nonattacking pairs of queens (28 is the value for the solution)
- Represent 8-queens state as an 8 digit string in which each digit represents position of queen
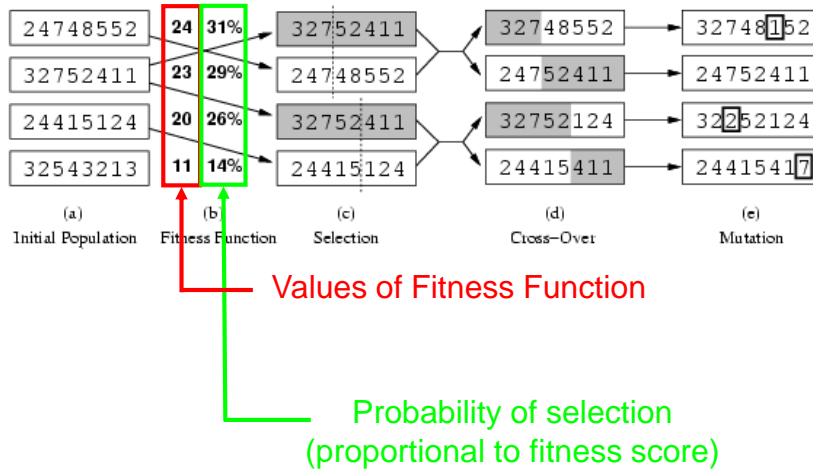
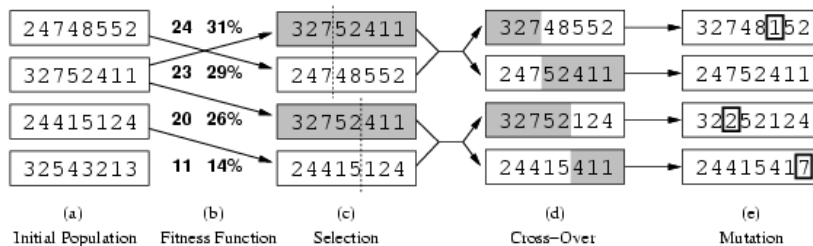| 3 | 2 | 7 | 5 | 2 | 4 | 1 | 1 |

49

Example: 8-queens

24748552	24 31%	32752411	32748552	32748152
32752411	23 29%	24748552	24752411	24752411
24415124	20 26%	32752411	32752124	32252124
32543213	11 14%	24415124	24415411	24415417
(a) Initial Population	(b) Fitness Function	(c) Selection	(d) Cross-Over	(e) Mutation

50

25

Example: 8-queens (Fitness Function)

24748552	**24**	**31%**	32752411	32748552	32748**1**52
32752411	**23**	**29%**	24748552	24752411	24752411
24415124	**20**	**26%**	32752411	32752124	32**2**52124
32543213	**11**	**14%**	24415124	24415411	244154**7**

| (a) | (b) | (c) | (d) | (e) |
| Initial Population | Fitness Function | Selection | Cross-Over | Mutation |

Values of Fitness Function

Probability of selection (proportional to fitness score)

51

Example: 8-queens (Selection)

24748552	**24**	**31%**	32752411	32748552	32748**1**52
32752411	**23**	**29%**	24748552	24752411	24752411
24415124	**20**	**26%**	32752411	32752124	32**2**52124
32543213	**11**	**14%**	24415124	24415411	244154**7**

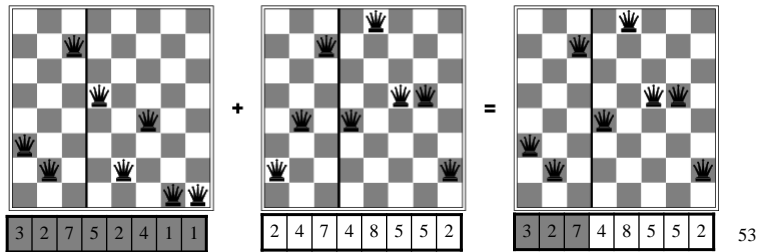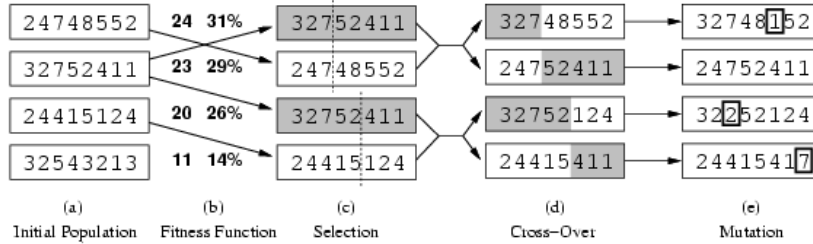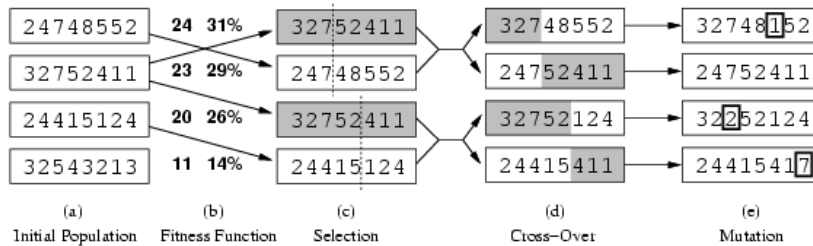| (a) | (b) | (c) | (d) | (e) |
| Initial Population | Fitness Function | Selection | Cross-Over | Mutation |

Notice 3 2 7 5 2 4 1 1 is selected twice while
3 2 5 4 3 2 1 3 is not selected at all

52

Example: 8-queens (Crossover)

Example: 8-queens (Mutation)

Mutation corresponds to randomly selecting a queen and randomly moving it in its column

Implementation details on Genetic Algorithms

- Initially, population is diverse and crossover produces big changes from parents
- Over time, individuals become quite 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
    - E.g. Putting first 3 queens in positions 2, 4, and 6 is a useful block

55

Schemas

- A substring in which some of the positions can be left unspecified eg. 246*****
- Instances: strings that match the schema
- If the average fitness of the instances of a schema is above the mean, then the number of instances of the schema within the population will grow over time

56

Schemas

- Schemas are important if contiguous blocks provide a consistent benefit
- Genetic algorithms work best when schemas correspond to meaningful components of a solution

57

The fine print…

- The representation of each state is critical to the performance of the GA
- Lots of parameters to tweak but if you get them right, GAs can work well
- Limited theoretical results (skeptics say it's just a big hack)

58

5. Gradient Descent

Discrete Environments

Hillclimbing pseudocode

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

- In discrete state spaces, the # of neighbors is finite.

- What if there is a continuum of possible moves leading to an infinite # of neighbors?

Local Search in Continuous State Spaces

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

61

Gradient Descent

- What is the gradient of a function f(x)?
  - Usually written as

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

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

62

Gradient Descent

Suppose we want to find a local minimum of a function f(x). We use the gradient descent rule:

$$x \leftarrow x - \alpha \nabla f(x)$$

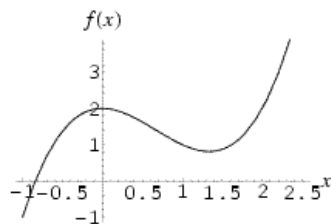α is the learning rate, which is usually a small number like 0.05

Suppose we want to find a local maximum of a function f(x). We use the gradient ascent rule:
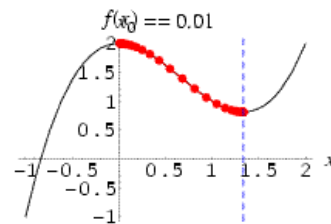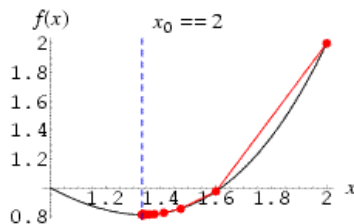
$$x \leftarrow x + \alpha \nabla f(x)$$

63

Gradient Descent Examples

$$f(x) = x^3 - 2x^2 + 2$$

These pictures were taken from Wolfram Mathworld

64

Question of the Day

- 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)

65

Multivariate Gradient Descent

- What happens if your function is multivariate eg. $f(x_1, x_2, x_3)$?
- Then

$$\nabla f(x_1, x_2, x_3) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right)$$
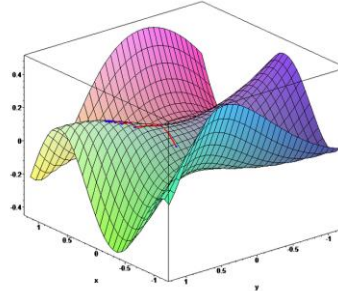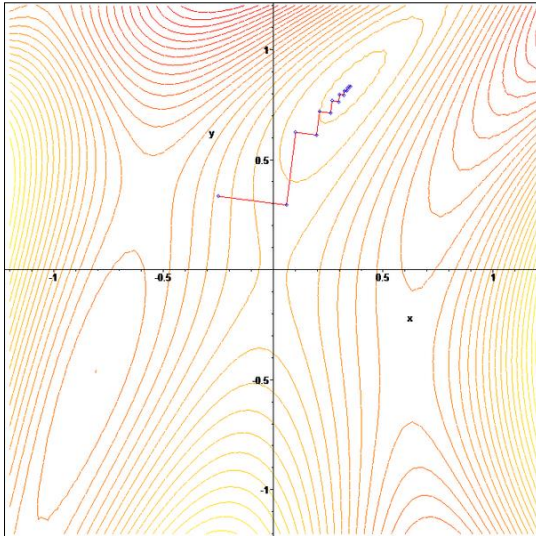
- The gradient descent rule becomes:

$$x_1 \leftarrow x_1 - \alpha \frac{\partial f}{\partial x_1} \qquad x_3 \leftarrow x_3 - \alpha \frac{\partial f}{\partial x_3}$$

$$x_2 \leftarrow x_2 - \alpha \frac{\partial f}{\partial x_2}$$

66

Multivariate Gradient Ascent

Pictures taken from
Wikipedia entry on
gradient descent

67

More About the Learning Rate

- If α is too large
  - Gradient descent overshoots the optimum point
- If α is too small
  - Gradient descent requires too many steps and
    will take a very long time to converge

68

Weaknesses of Gradient Descent

1. Can be very slow to converge to a local optimum, especially if the curvature in different directions is very different
2. Good results depend on the value of the learning rate α
3. What if the function f(x) isn't differentiable at x?

69

What You Should Know

- Be able to formulate a problem as a local search problem
- Know the difference between local search and uninformed and informed search
- Know how hillclimbing works
- Know how simulated annealing works
- Know the pros and cons of both methods

70

What you should know

- Be able to formulate a problem as a Genetic Algorithm
- Understand what crossover and mutation do and why they are important
- Differences between hillclimbing, simulated annealing, local beam search, and genetic algorithms
- Understand how gradient descent works, including its strengths and weaknesses
- Understand how to derive the gradient descent rule

71

Exercise

You have to move from your old apartment to your new one. You have the following:

- A list $L = \{a_1, a_2, \ldots, a_n\}$ of n items, each with a size $s(a_i) > 0$.
- There are M moving boxes available, each with a box capacity C (assume MC far exceeds the sum of the sizes of your items). You can put as many items into a box as long as the sum of their sizes does not 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.

72

Exercise (continued)

- States?

- Neighborhood?

- Evaluation function?

- How to avoid local maxima?

73