CS 331: Artificial Intelligence
Local Search

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

What do these problems have in common?

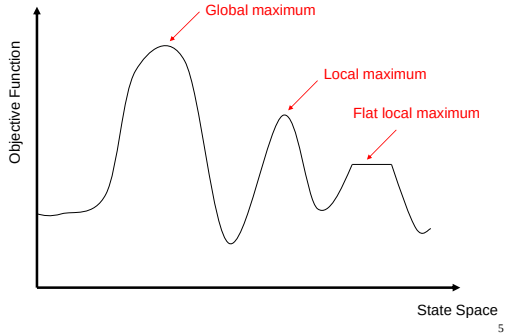
- These problems are unlike the search problems from last class:
  - The path to the goal is irrelevant -- all you care about is the final configuration
  - These are often optimization problems in which you find the best state according to an objective function
- These problems are examples of local search problems

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.

1D State-Space Landscape



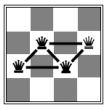
Why is this hard?

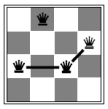
- Lots of states (sometimes an infinite number)
- Most of these problems are NP-complete
- Objective function might be expensive

But:

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

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
3. Select one of its neighbor X*
4. Move to X* and repeat until the current configuration is satisfactory

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

Which neighbor you select is also very important

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

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.

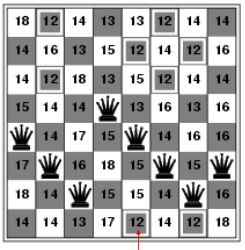
11

Hillclimbing Pseudocode

```
X ← 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
```

12

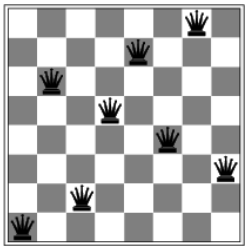
Example: 8-queens



- 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

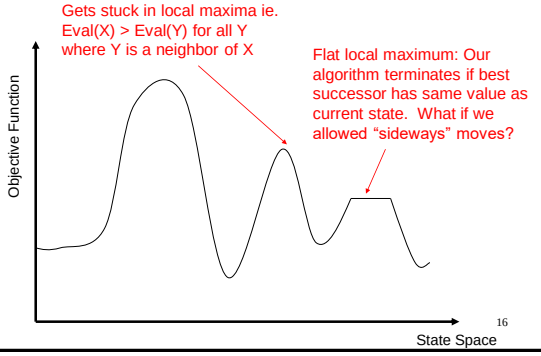


- Local minimum with h=1
- Note: at global minimum, h=0

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

Problems with Hill-climbing



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

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

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

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

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?

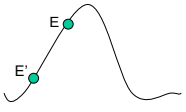
Setting p

- 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

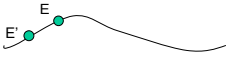
Setting p

- **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

Decreasing p as E-E' increases



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



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

Setting p

- Needs a temperature parameter T
- If E' ≤ 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

Complete 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
```


Complete 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
```

We keep the temperature fixed and we iterate K times

Exponential cooling schedule T(n) = α T(n-1) with 0 < α < 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

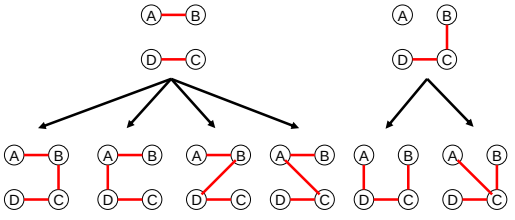
Travelling Salesman Problem

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

34

Local Beam Search Example

Travelling Salesman Problem (k=2)

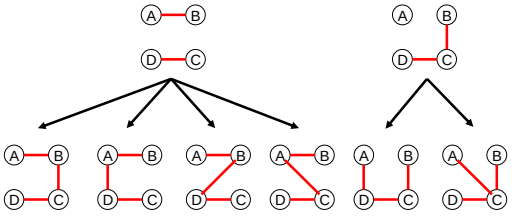


Generate all successors of all the k states

35

Local Beam Search Example

Travelling Salesman Problem (k=2)

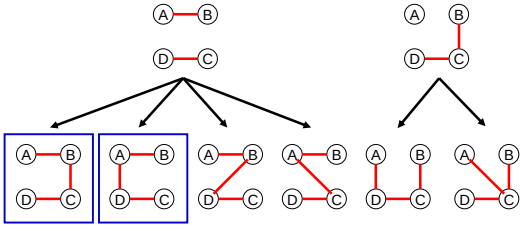


None of these is a goal state so we continue

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

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

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

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.

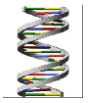
Your job is to pack your stuff into as few boxes as possible. Formulate this as a local search problem.

Exercise (continued)

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

4. Genetic Algorithms

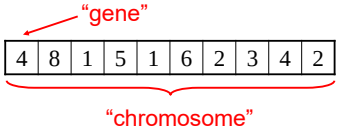
Genetic Algorithms



- 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

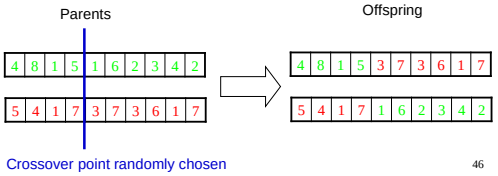
Definitions

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



Definitions

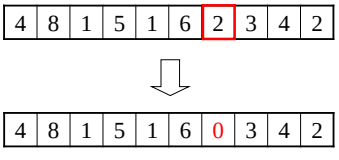
- Selection: Pick two random individuals for reproduction
- Crossover: Mix the two parent strings at the crossover point



Crossover point randomly chosen

Definitions

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



Randomness aids in avoiding small local extrema

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

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

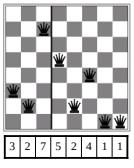
Lots of variants

- Variant 1: Culling - individuals below a certain threshold are removed
- Variant 2: Selection based on:

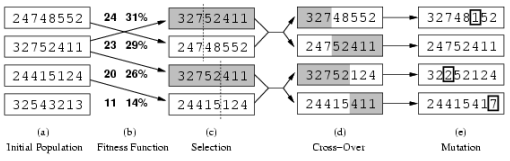
$$P(X \text{ selected}) = \frac{Eval(X)}{\sum_{Y \in Population} Eval(Y)}$$

Example: 8-queens

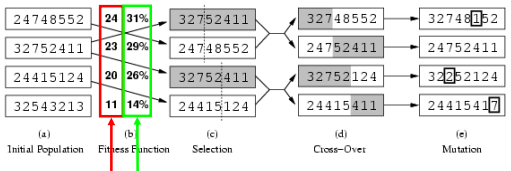
- Fitness Function: number of nonattacking pairs of queens (28 is the value for the solution)
- Represent 8-queens state as an 8 digit string in which each digit represents position of queen



Example: 8-queens



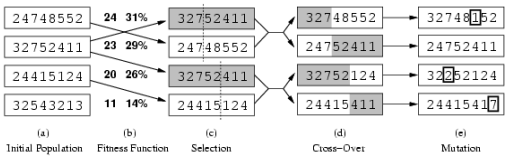
Example: 8-queens (Fitness Function)



Values of Fitness Function

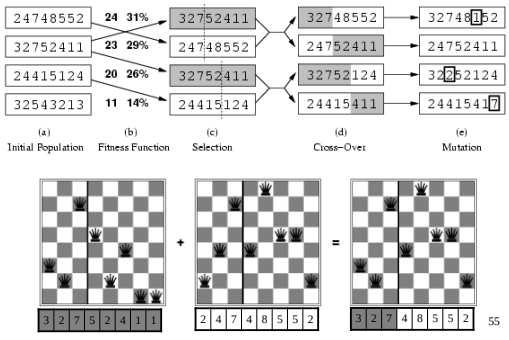
Probability of selection (proportional to fitness score)

Example: 8-queens (Selection)



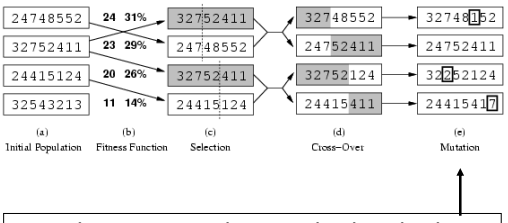
Notice 3 2 7 5 2 4 1 1 is selected twice while 3 2 5 4 3 2 1 3 is not selected at all

Example: 8-queens (Crossover)



55

Example: 8-queens (Mutation)



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

56

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

57

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

58

Schemas

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

59

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)

60

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)

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

Gradient Descent

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

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

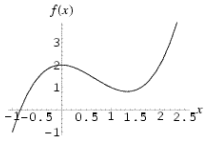
α is the learning rate, which is usually a small number like 0.05

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

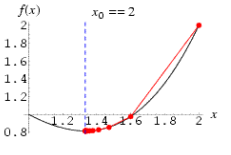
$$x \leftarrow x + \alpha \nabla f(x)$$

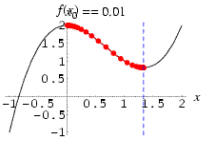
Gradient Descent Examples

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



These pictures were taken from Wolfram Mathworld





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)

Multivariate Gradient Descent

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

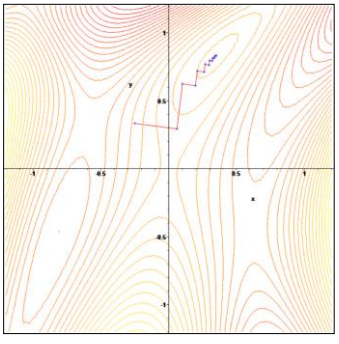
$$\nabla f(x_1, x_2, x_3) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right)$$

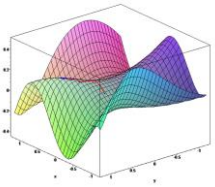
- The gradient descent rule becomes:

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

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

Multivariate Gradient Ascent





Pictures taken from Wikipedia entry on gradient descent

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

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?

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

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

73