The Traveling Salesman Problem

Search Algorithms
The Traveling Salesman Problem
Reading: CSO 12.1 -- 12.5

Search

- An important class of algorithms in computer science are search algorithms
  - exact match in string searches
  - using patterns (e.g. regular expressions)
  - inexact match (e.g. k-mismatch, BLAST)

- This week: how to search when the target is even less well-specified
  - "I can't describe it, but I'll know it when I see it"
  - example: SPAM
  - regular expressions (e.g. that match "mor(ge)$") and examples of mail containing offers to refinance your house are too specific
  - we want a filter that recognizes generic SPAM, something that will also match mail offering diplomas, low-cost drugs, lottery winnings, ...

Machine Learning

- One way to find SPAM: machine learning
  - teach the computer what SPAM is

- Start with a filter that knows nothing about SPAM

- As new SPAM arrives, tell the search algorithm "this is an example of what I want you to throw away"

- We want the algorithm to be able to generalize
  - if we're successful it will start throwing away mail has little in common with previous examples of SPAM

Examples of Machine Learning

- Some areas where machines have been trained to perform useful tasks:
  - sorting mail -- computers at the US Postal Service accurately route 85% of handwritten letters
  - understanding speech -- phone trees at customer support centers, “Julie” at Amtrak, many other automated systems that respond to verbal input
  - patterns in scientific data -- computers have successfully learned to distinguish many complex objects (stars vs. galaxies, volcanoes vs. impact craters, ...)

solarsystem.nasa.gov
Why Training?

- It is often easier to give examples than it is to define exact rules
  - these should all be recognized as the letter “A”

  ![Rule: two slanted lines connected by horizontal line](image)
  ![Exception: vertical line](image)
  ![Extra bits hanging off the side or top](image)
  ![Curvy lines...](image)

Research on Machine Learning

- Machine learning is an active area of research in computer science
- Combination of algorithms, artificial intelligence, statistics, and much more
- See:
  - “The Discipline of Machine Learning”
  - Tom M. Mitchell
  - a PDF copy is on the class web site (click on “resources”)
- Fairly technical, but a nice overview of the field and how it relates to computer science

Genetic Algorithms

- The rest of this lecture looks at one type of algorithm used for machine learning
- A genetic algorithm is based on ideas from population biology
- The basic idea:
  - a problem solver has a collection of many different potential solutions
  - the “fitness” of a solution is a measure of how well it solves the problem
  - use “natural selection” to keep good solutions
  - replace bad solutions with new ones derived from the survivors

Genetic Algorithms (cont’d)

- Example: learning to recognize the letter “A”
- Start with a collection of samples -- make sure they are diverse
- Write a program that has parameters (number of line segments, angles, ...)
- Make copies of the program, give each a different set of values for the parameters
- Evaluate all the programs
Genetic Algorithms (cont’d)

- Keep the best programs (i.e. the ones that correctly identify the most samples)
- Rebuild the population by making copies of these successful programs
- Tweak the new versions so they are slightly different than their “parents”
- Stop when a program recognizes all the samples (or if no further progress seems possible)

There are lots of difficult issues
- what is the input (pixels, line segments, ...)?
- how many parameters?
- include counter-examples (F, Z, other angular letters) in the test set
- need to test the system after it is trained
- make sure the test set is a representative sample (don’t train on sans serif, evaluate on handwriting...)

Example Application: Traveling Salesman

- The traveling salesman problem is a classic problem in computer science
  - suppose you have a list of \( n \) cities
  - the goal is to define a tour that visits all \( n \) cities and returns to the starting place
  - you can visit each city only one time
- What is the lowest cost tour?
  - at right: a tour of 13,500 US cities

Cost can be defined as driving time, distance, air fare, ...
- assume the cost of going from X to Y is the same as going from Y to X
- Although it sounds simple this is a very hard problem to solve
  - it’s easy to write a program that works for a small number of cities
  - but the expected time is \( O(2^n) \)
  - each time a city is added to the list the time to find a tour doubles
  - for a tour of 20 cities the program might have to check 1 billion combinations
- Some figures from TSPlib++ (an “industrial strength”) solver:
  - 5.5 hours for 10,000 cities
  - estimate over 6,000 years for 25,000 cities
Real-Life Applications

- The idea that anyone would really plan a road trip to 13,000 cities is a bit silly
- But this problem is identical to several important “real world” problems:
  - transportation: school bus routes, service calls, delivering meals, ...
  - manufacturing: an industrial robot that drills holes in printed circuit boards used in computers, video and stereo, almost anything electronic
  - communication: planning new telecommunication networks
  - biology: genetic markers on chromosomes / reassembly

History

- The traveling salesman problem has been studied by mathematicians for over 100 years
  - originally posed as a puzzle in the 1800s
  - started attracting serious attention in the 1930s
  - one of the most widely studied problems in applied math and operations research

Optimization

- The TSP is an example of an optimization problem
- The goal is to find the order of cities that has the lowest cost
- A plot like the one shown here is helpful to visualize the process
- assume there is a method for ordering the solutions along the x axis

Brute Force

- The “obvious” way to solve the problem is to check every tour
- Start with tour 0
  - evaluate the cost
  - modify the tour and repeat
  - keep track of the tour number that gives the lowest cost
- The problem: way too many tours to check
  - for $n$ cities: up to $n!$ orderings
  - but there is a way to “reduce” the problem to $2^n$ steps
Hill Climbing

- A slightly better approach is known as **hill climbing**
- Pick a starting spot
  - may be random, or it may be based on some extra information (e.g. “the tour must go through X, Y, and Z at the start”)
  - check neighboring points
  - move toward the neighbor with the better cost and repeat
- “Hill climbing” refers to a search for maximum, but the idea still applies when searching for minimum

Local Minima

- The problem with hill climbing is that it might find a **local minimum**
- Ideally we want an algorithm that will find the **global minimum**
- We also want to avoid an exhaustive search of all possible tours

Genetic Algorithms and Optimization

- We can use genetic algorithms in optimization problems
- For TSP:
  - create a “population” of possible tours
  - the “fitness” of a tour is the cost of the tour
  - use “natural selection” to keep best tours
  - replace bad tours with new ones derived from the survivors

Genetic Algorithm (cont’d)

- The key idea in a GA is that “individuals” represent problem solutions
- Generation of new solutions happens by:
  - **mutation**: make a copy of an existing solution and make a small change
  - **cross-over**: select two existing solutions, combine elements at random to produce a new solution
- In both cases the result is a **complete solution**
Genetic Changes

- The idea of using point mutations and cross-overs was inspired by genetics.

First phase of meiosis (production of germ cells)

Change to one base causes change in amino acid


GA for TSP: Solutions

- The first step in designing a genetic algorithm to solve the TSP is to figure out how to represent a solution.

- The problem is often described in terms of a graph:
  - Graphs are similar to trees — they are collections of nodes.
  - A graph does not have a root.
  - There can be any number of connections between nodes.

A graph with 7 nodes

Two examples of complete tours based on the graph

Solutions (cont’d)

- For many applications there is a “road” between every city.
  - Example: the robot arm drilling holes in a PC board can move freely from any point to any other point.
  - For the remainder of these slides, we’ll assume that’s the case.
  - The graph for this problem is “fully connected.”

- A simple way to represent the tour is to use a string:
  - If there are $n$ cities, there are $n$ letters in the string.
  - Tours of more than 26 cities would use arrays of integers, but strings are useful for small demos (easy to understand, easy to display).
  - For the small graph shown below, strings would have the letters “A” through “G.”

Any string that is a permutation of these letters is a valid solution.

Q: how many possible tours are there in this graph?
GA for TSP: Mutations

- The role of mutations in a genetic algorithm is to make small changes to one of the current solutions.
- This allows the algorithm to close in on the final solution.
- The example at right shows a population of 5 blue solutions with 5 new solutions (green) that are only slightly different from the current ones.
- Not every mutation leads to a better solution.

![Diagram showing the role of mutations in a genetic algorithm.](image)

Mutations (cont’d)

- One technique for defining mutations on paths:
  - pick two links at random
  - swap their endpoints
  - reverse the direction of the loop between the endpoints
- Example: swap ends of C \(\rightarrow\) D and H \(\rightarrow\) I.

![Diagram illustrating mutations on paths.](image)

GA for TSP: Cross-Overs

- Cross-overs define larger changes.
- The idea is that combining the best parts of two very different solutions has a chance of “hopping out” of the region near a local minimum.
- Not every cross will lead to an improvement.

![Diagram showing the role of cross-overs in a genetic algorithm.](image)

Cross-overs (cont’d)

- Defining cross-overs for strings is a little more difficult:
  - pick a cross-over point \(x\)
  - in one string select left of \(x\), copy to the new string
  - in the other string select from the right of \(x\)
  - stop when selecting a letter already copied
  - finish up by selecting the remaining chars at random
- Example: \(HIECBAFGJ \times BAIHEGDFJ\) with D as the crossover point.

![Diagram illustrating cross-overs for strings.](image)
GA for TSP: Main Loop

- Create a string $S$ with one letter per city ("ABCD....")
- Create an initial population using $n$ random permutations of $S$
- Repeat:
  - natural selection -- remove individual $i$ with $p(\text{fitness}(i))$
  - rebuild the population to size $n$:
    (a) copy random individuals, apply point mutation
    (b) apply cross-over to random pairs
- Stop when the best solution does not improve, or after a maximum number of steps

Parameters and Variations

- There are lots of small adjustments one can make to the basic algorithm
- Selection:
  - what percentage of the population should be kept?
  - larger proportions lead to more stability but may be too slow to evolve
  - always remove the least fit? or use $p(\text{fitness})$?
  - keeping random poor solutions adds to variability in the population
- Mutation:
  - how many to apply at each round?
  - mutate only new solutions?
- Cross-over:
  - how often should cross-overs happen?
  - should there always be fewer cross-overs than point mutations?

Honor Diversity

- The first time a GA is tested the developer often finds selection is "too effective"
- The algorithm zeroes in on a few local minima
- The trick is to ensure enough variability in the population so cross-overs eventually find the valley with the global minimum
- One idea: delete solutions that are too similar to other solutions

Road Trip

- To test this algorithm I wrote a small Ruby program
- Data set: cities of the Pac-10 universities
- Distances between each city supplied by maps.google.com
- I decided to use driving time as the measure of cost
Best Path

- A run that found the best path:
  ```
  % tspga cities.txt -g 50 --ps .25 --pp .25 --n 50
  time = 3828
  Shortest route:  2 days 15 hours 48 minutes
  Eugene -> Corvallis -> Seattle -> Pullman -> Tempe -> Tucson
  -> Pasadena -> Los Angeles -> Stanford -> Berkeley -> Eugene
  ```

- One that came close:
  ```
  % tspga cities.txt -g 50 --ps .25 --pp .25 --n 50
  time = 3960
  Shortest route:  2 days 18 hours 0 minutes
  Eugene -> Los Angeles -> Tempe -> Tucson -> Pasadena ->
  Stanford -> Berkeley -> Corvallis -> Pullman -> Seattle -> Eugene
  ```

- Random starting tours had values as high as 6148 (4 days 6 hours 28 mins)

Summary

- A new way to implement a search: train an algorithm to recognize examples of
  the pattern you want it to find
  - SPAM, handwriting samples, images of craters, ...

- Genetic algorithms, inspired by concepts from population biology, are widely
  used in machine learning
  - make a population of solutions
  - new solutions are derived from existing solutions (using point mutations and
crossovers)
  - after several generations good solutions begin to emerge

- Programs that learn are very complex
  - to understand how genetic algorithms work we looked at a much simpler
  application: optimization
  - "learning" in this case is just "you’re getting warmer..."
  - no generalization

Summary (cont’d)

- The traveling salesman problem is an example of an optimization problem

- A genetic algorithm is a way of converging on the best solution
  - select several potential solutions at random
  - gradually improve the solutions
  - every now and then introduce a substantial change
  - depending on the problem domain and evolution parameters the process will
  eventually converge on a good (if not best) solution

- In Friday’s lecture:
  - a Ruby implementation of TSP
  - lab projects will use this program