CSE 326: Data Structures

Good News, Bad News

James Fogarty

Spring 2009
The Good News

- You’ve learned a set of tools that allow you to think about, and talk about, a wide variety of computing problems
How Do You Decide Which Examples are Positive?
KNN Graph

• Compute pairwise distance between every pair (using domain knowledge to determine the distance metric)

• Preserve edges corresponding only to the k nearest neighbors

• Run a search from your positive and negative examples, classify each based on whichever is closer
KNN Graph

- KNN transformation greatly reduces $|E|$, from $|V^2|$ to $k|V|$, dense to sparse

- Your classification runs faster

- It’s also semi-supervised, respecting the distribution of your data
How Do You Decide Which Examples are Positive?
How About Now?
Unconnected Graph

• The KNN transformation does not necessarily preserve a path between your labels and all of your data

• You have to decide what this means and what to do about it

• What kinds of tools can you work with?
How Do You Choose a Representative Set?
The Bad News

• We do not know how to efficiently compute something as simple as “the most representative of these”

• And it gets worse
Your First Task

Your company has to inspect a set of roads driving along each of them.

Driving costs money (fuel), and there are a lot of roads.

Your boss wants you to figure out how to drive each road exactly once.

You get a bonus if, after inspecting the last road, the car is back where it started.
Try it with paper and pencil

Which of these can you draw without lifting your pencil, drawing each line only once?
Can you start and end at the same point?
Graph Problem for the Bridges of Königsberg

Want to cross all bridges but…
Can cross each bridge only once
Euler Circuits and Tours

• **Euler tour**: a path through a graph that *visits each edge exactly once*

• **Euler circuit**: an Euler tour that *starts and ends at the same vertex*

• Named after Leonhard Euler (1707-1783), who cracked this problem and founded graph theory in 1736

• Some observations for undirected graphs:
  › An Euler circuit exists *iff* the graph is connected and each vertex has *even* degree
  › An Euler tour exists *iff* the graph is connected and either all vertices have even degree or exactly two have odd degree
Finding Euler Circuits

Given a connected, undirected graph \( G = (V,E) \), find an Euler circuit in \( G \).

**Euler Circuit Existence Algorithm:**
Check to see that all vertices have even degree

Running time =

**Euler Circuit Algorithm:**
1. Do an edge walk from a start vertex until you are back at the start vertex. Mark each edge you visit, and do not revisit marked edges. You never get stuck because of the even degree property.
2. The walk is removed leaving several components each with the even degree property. Recursively find Euler circuits for these.
3. Splice all these circuits into an Euler circuit

Running time =
Euler Circuit Example

Euler(A) :
Euler Circuit Example

Euler(A) :
A B G E D G C A
Euler Circuit Example

Euler(A) : 
A B G E D G C A

Euler(B)
Euler Circuit Example

Euler(A) :
A B G E D G C A

Euler(B) :
B D F E C B
Euler Circuit Example

Euler(A):
A → B → G → E → D → G → C → A

Euler(B):
B → D → F → E → C → B

Splice:
A → B → D → F → E → C → B → G → E → D → G → C → A
Euler Tour

For a Euler Tour, exactly two vertices are odd, the rest even. Using a similar algorithm, you can find a path between the two odd vertices.
Your Second Task

Your boss is pleased, and has more work.

Your company has to send someone by car to a set of cities.

The primary cost is the exorbitant toll going into each city.

Your boss wants you to figure out how to drive to each city exactly once, returning in the end to the city of origin.
Hamiltonian Circuits

• Euler circuit: A cycle that goes through each edge exactly once

• Hamiltonian circuit: A cycle that goes through each vertex exactly once (except the first=last one)

• Does graph I have:
  › An Euler circuit?
  › A Hamiltonian circuit?

• Does graph II have:
  › An Euler circuit?
  › A Hamiltonian circuit?

• Does the Hamiltonian circuit problem seem easier or harder?
Finding Hamiltonian Circuits

• Problem: Find a Hamiltonian circuit in a connected, undirected graph G.

• One solution: Search through all paths to find one that visits each vertex exactly once
  › Can use your favorite graph search algorithm to enumerate various potential paths

• This is an exhaustive search (“brute force”) algorithm.

• Worst case → need to search all paths
  › How many paths??
Analysis of our Exhaustive Search Algorithm

• Worst case → search all paths
  › How many paths?

• Can depict these paths as a search tree
Analysis of our Exhaustive Search Algorithm

- Let the average branching factor of each node in this tree be $B$
- $|V|$ vertices, each with $\approx B$ branches
- Total number of paths $\approx B \cdot B \cdot B \ldots \cdot B = \ldots$
- Worst case $\rightarrow$ Exponential time!

*Search tree of paths from B*
Exponential Time

PC, since Big Bang

PC, 1 day
Polynomial vs. Exponential Time

- Most of our algorithms so far have been $O(\log N)$, $O(N)$, $O(N \log N)$ or $O(N^2)$ running time for inputs of size $N$
  - These are all *polynomial time* algorithms
  - Their running time is $O(N^k)$ for some $k > 0$

- Exponential time $B^N$ is asymptotically worse than *any* polynomial function $N^k$ for *any* $k$
Problem Spaces

Simplified view: if a problem is not polynomial-time solvable (not in P), then it is an exponential-time problem.

Shorthand:

- P solutions are fast
- EXPTIME are slow
  - Sometimes viewed as “intractable”
Try as you might, every solution you come up with for the Hamiltonian Circuit problem runs in exponential time.

You have to report back to your boss.
When is a problem easy?

• We’ve seen “easy” graph problems:
  › Shortest-path
  › Maximum Flow
  › Minimum Spanning Tree

• Not easy for us to come up with, but easy for the computer after we figure out the algorithm.
When is a problem hard?

• Almost everything we’ve seen in class has had a near linear time algorithm.

• But of course, computers can’t solve every problem quickly.

• In fact, there are perfectly reasonable sounding problems that no computer could ever solve in any reasonable amount of time (as far as we know!).
When is a problem hopeless?

- If the solution to a problem requires generating a result that is exponential in size, then the algorithm must run in exponential time (even if just to print the solution!).
- Some problems are “undecideable”: no algorithm can be given for solving them.
  - The Halting Problem: is it possible to specify any algorithm, which, given an arbitrary program and input to that program, will always correctly determine whether or not that program will enter an infinite loop?
    - No! [Turing, 1936]

- We’ll focus on problems that have a glimmer of hope…
A Glimmer of Hope

Suppose you have a problem that is at least decideable.

If the output can be checked for correctness in polynomial-time, then \textit{maybe} a polynomial-time solution exists!
The Complexity Class NP

- **Definition**: NP is the set of all problems for which a given candidate solution can be tested in polynomial time.

- Are the following in NP:
  - Hamiltonian circuit problem?
  - Euler circuit problem?
  - All polynomial time algorithms?
Why NP?

• NP stands for *Nondeterministic Polynomial*

  › Why “nondeterministic”? Corresponds to algorithms that can guess a solution (if it exists) → the solution is then verified to be correct in polynomial time

  › Nondeterministic algorithms don’t exist – purely theoretical idea invented to understand how hard a problem could be
Problem Spaces (revisited)

We can now augment our problem space to include NP as a superset of P.

Whenever someone finds a polynomial time solution to a problem currently believed to be in NP - P, it moves to P.
Your Chance to Win a Turing Award (and make $$$)

• It is generally believed that $P \neq NP$, i.e. there are problems in NP that are not in P
  › But no one has been able to show even one such problem!
  › This is the fundamental open problem in theoretical computer science.
  › Nearly everyone has given up trying to prove it. Instead, theoreticians prove theorems about what follows once we assume $P \neq NP$!
Perhaps instead $P = NP$, but that would seem to be even harder to prove…

$P \neq NP$

$P = NP$
Satisfiability

In 1971, Stephen Cook studied the Satisfiability Problem:

› Given a Boolean formula (collections of ANDs, ORs, NOTs) over a set of variables,
› is there a TRUE/FALSE assignment to the variables such that the Boolean formula evaluates to TRUE?
Satisfiability

…and he proved something remarkable:

Every problem in NP can be polynomially transformed to the Satisfiability Problem.

Thus, Satisfiability is at least as hard as every other problem in NP, i.e., it is the hardest NP problem.

We call it an “NP-complete” problem.
NP-completeness

• In fact, Satisfiability can be polynomially reduced to some other NP problems (and vice versa).
• These other problems are equivalent to Satisfiability, and so all other problems in NP can be transformed to them, as well.
• NP-complete problems thus form an equivalence set in NP (all equivalently hard, i.e., the hardest).
• Solving one would give a solution for all of them!
  › If any NP-complete problem is in P, then all of NP is in P
Coping with NP-Completeness

1. Settle for algorithms that are fast on average:
   Worst case still takes exponential time, but doesn’t occur very often.
   But some NP-Complete problems are also average-time NP-Complete!

2. Settle for fast algorithms that give near-optimal solutions:
   In traveling salesman, may not give the cheapest tour, but maybe good enough.
   But finding even approximate solutions to some NP-Complete problems are NP-Complete!
Coping with NP-Completeness

3. Just get the exponent as low as possible! Much work on exponential algorithms for satisfiability: in practice can often solve circuits with 1,000+ inputs. But even $2^{n/100}$ will eventually hit the exponential curve!

4. Restrict the problem: Longest Path is easy on trees, for example. Many hard problems are easy for restricted inputs.