Lecture 17

What we’ve done and what’s to come
TLDR:

CS21SI: AI for Social Good is a 2-unit student-taught class in which students learn about and apply cutting-edge artificial intelligence techniques to real-world social good spaces, such as healthcare, government, education, and the environment.

The course alternates between lectures on machine learning theory and discussions with invited speakers, who will challenge students to apply techniques in their social good domains. You can learn more about the class at http://web.stanford.edu/class/cs21si/

Apply by 11:59pm on Sunday, March 17 at bit.ly/AIForGoodApp.
Announcements

• Final exam:
  • 8:30am – 11:30am, Thursday 3/21.
  • Location:
    • Last name A-LIA: Cubberley Auditorium
    • Last name LIB-Z: Hewlett 200
  • You may bring two double-sided sheets of notes
  • Format: similar to the midterm

• See announcements from Monday 3/11 for more info, as well as study tips!
More announcements

- Course feedback open!
  - Fill it out on axess!
  - Your feedback is super-important!
  - It will help us make the course better!

Figure 1: Feedback
Today

• What just happened?
  • A whirlwind tour of CS161

• What’s next?
  • A few gems from future algorithms classes
It’s been a fun ride...

- Sorting and friends!
- Divide-and-conquer and recurrence relations
- $O()$ and worst-case analysis
- Data structures: BSTs and Hashing!
- Randomized algorithms
- Graphs!
- Divide-and-conquer and recurrence relations
- BFS, DFS, SCCs
- Dynamic Programming!
- Greedy algorithms!
- Scheduling and etc.
- Minimum Cuts
- LCS, Knapsack(s)
- Bellman-Ford, Floyd-Warshall
- MSTs: Prim and Kruskal
- Dijkstra’s algorithm
- Karger and Karger-Stein
- Minimum Cuts
What have we learned?

16 lectures in 12 slides.
General approach to algorithm design and analysis

**Does it work?**
**Is it fast?**
**Can I do better?**

To answer these questions we need both **rigor** and **intuition**:

- **Algorithm designer**
- **Plucky the Pedantic Penguin**
  - Detail-oriented
  - Precise
  - Rigorous
- **Lucky the Lackadaisical Lemur**
  - Big-picture
  - Intuitive
  - Hand-wavey
We needed more details

Does it work?
Is it fast?
Can I do better?

What does that mean??

Worst-case analysis

Here is an input!

big-Oh notation

\[ T(n) = O(f(n)) \]
\[ \iff \exists c, n_0 > 0 \text{ s.t. } \forall n \geq n_0,\]
\[ 0 \leq T(n) \leq c \cdot f(n) \]

Anakin the Adversarial Aardvark
Algorithm design paradigm: divide and conquer

- Like MergeSort!
- Or Karatsuba’s algorithm!
- Or SELECT!
- How do we analyze these?

By careful analysis!
Useful shortcut, the master method is.

Plucky the Pedantic Penguin
Jedi master Yoda
While we’re on the topic of sorting

Why not use randomness?

• We analyzed QuickSort!
• Still worst-case input, but we use randomness after the input is chosen.
• Always correct, usually fast.
  • This is a Las Vegas algorithm
All this sorting is making me wonder... Can we do better?

- Depends on who you ask:
  - **RadixSort** takes time $O(n)$ if the objects are, for example, small integers!
  - Can’t do better in a comparison-based model.
beyond sorted arrays/linked lists: Binary Search Trees!

• Useful data structure!
• Especially the self-balancing ones!

Red-Black tree!

Maintain balance by stipulating that black nodes are balanced, and that there aren’t too many red nodes.

It’s just good sense!
Another way to store things
Hash tables!

All of the hash functions
\( h : U \rightarrow \{1, \ldots, n\} \)

Choose \( h \) randomly from a universal hash family.

It’s better if the hash family is small!
Then it takes less space to store \( h \).
OMG GRAPHS

• BFS, DFS, and applications!
• SCCs, Topological sorting, ...
A fundamental graph problem: shortest paths

- Eg, transit planning, packet routing, ...
- Dijkstra!
- Bellman-Ford!
- Floyd-Warshall!
Bellman-Ford and Floyd-Warshall were examples of...

- Not programming in an action movie.

- **Step 1:** Identify **optimal substructure**.

- **Step 2:** Find a **recursive formulation** for the value of the optimal solution.

- **Steps 3-5:** Use **dynamic programming**: fill in a table to find the answer!

Instead, an algorithmic paradigm!

We saw many other examples, including Longest Common Subsequence and Knapsack Problems.
Sometimes we can take even better advantage of optimal substructure...with Greedy algorithms

• Make a series of choices, and commit!

• Intuitively we want to show that our greedy choices never rule out success.

• Rigorously, we usually analyzed these by induction.

• Examples!
  • Activity Selection
  • Job Scheduling
  • Huffman Coding
  • Minimum Spanning Trees

Prim's algorithm: greedily grow a tree
Kruskal's algorithm: greedily grow a forest
Minimum cuts

- Karger’s algorithm!
- Karger-Stein algorithm!

Karger’s algorithm is a Monte-Carlo algorithm: it is always fast but might be wrong.
And now we’re here
What have we learned?

• A few algorithm design paradigms:
  • Divide and conquer, Dynamic Programming, Greedy

• A few analysis tools:
  • Worst-case analysis, asymptotic analysis, recurrence relations, probability tricks, proofs by induction

• A few common objects:
  • Graphs, arrays, trees, hash functions

• A LOT of examples!
What have we learned?
We’ve filled out a toolbox

• Tons of examples give us intuition about what algorithmic techniques might work when.
• The technical skills make sure our intuition works out.
But there’s lots more out there

• What’s next???
A taste of what’s to come

- CS154 – Introduction to Complexity
- CS166 – Data Structures
- CS167 – Readings in Algorithms
- CS168 – The Modern Algorithmic Toolbox
- MS&E 212 – Combinatorial Optimization
- CS250 – Error Correcting Codes
- CS254 – Computational Complexity
- CS255 – Introduction to Cryptography
- CS261 – Optimization and Algorithmic Paradigms
- CS264 – Beyond Worst-Case Analysis
- CS265 – Randomized Algorithms
- CS269Q – Quantum Computing
- CS269O – Introduction to Optimization Theory
- EE364A/B – Convex Optimization I and II

...and many many more upper-level topics courses!

**findSomeTheoryCourses():**
- go to theory.stanford.edu
- Click on “People”
- Look at what we’re teaching!

This Spring!!!
Today

A few gems

• Linear programming
• Random projections
• Low-degree polynomials

This will be pretty fluffy, without much detail – take more CS theory classes for more detail!

NOTHING AFTER THIS POINT WILL BE ON THE FINAL EXAM
Linear Programming

• This is a fancy name for optimizing a linear function subject to linear constraints.

• For example:

\[
\text{Maximize } \quad x + y \\
\text{subject to } \\
x \geq 0, \quad y \geq 0, \quad 4x + y \leq 2, \quad x + 2y \leq 1
\]

• It turns out the be an extremely general problem.
Example

Transportation problem:

Goal: minimize transportation cost subject to meeting everyone’s needs.

\[
\min \sum_{edges} \text{cost on that edge} \quad \text{s.t. Supply/demand constraints are met.}
\]

Produce: 6 pounds of fish
Need: 5 pounds of fish

Produce: 3lb
Need: 5lb

Produce: 10lb
Need: 2lb

Produce: 6 pounds of fish
Need: 5 pounds of fish

Cost of transport: $3/pound of fish.

Cost of transport: $5/pound of fish.
Linear Programming

Has a really nice geometric intuition

Maximize

\[ x + y \]

subject to

\[ x \geq 0 \]
\[ y \geq 0 \]
\[ 4x + y \leq 2 \]
\[ x + 2y \leq 1 \]
Linear Programming

Has a really nice geometric intuition

Maximize

\[ x + y \]

subject to

\[ x \geq 0 \]
\[ y \geq 0 \]
\[ 4x + y \leq 2 \]
\[ x + 2y \leq 1 \]
Linear Programming

Has a really nice geometric intuition

Maximize

\[
x + y
\]

subject to

\[
\begin{align*}
x + y &\leq 2 \\
x &\geq 0 \\
y &\geq 0 \\
4x + y &\leq 2 \\
x + 2y &\leq 1
\end{align*}
\]

The function is maximized here!
In general

- The constraints define a **polytope**
- The function defines a **direction**
- We just want to find the vertex that is **furthest in that direction**.

The function is maximized here!
Duality

How do we know we have an optimal solution?

I claim that the optimum is $5/7$.

Proof: say $x$ and $y$ satisfy the constraints.

- $x + y = \frac{1}{7} (4x + y) + \frac{3}{7} (x + 2y)$
- $\leq \frac{1}{7} \cdot 2 + \frac{3}{7} \cdot 1$
- $= \frac{5}{7}$

You can check this point has value $5/7$. So it must be optimal!
cute, but

How did you come up with $1/7, 3/7$?

I claim that the optimum is $5/7$.

Proof: say $x$ and $y$ satisfy the constraints.

- $x + y \leq (4x + y) + (x + 2y)$
- $\leq 2 + 1$
- $= 3$

I want to choose things to put here

So that I minimize this

Subject to these things

Maximize

\[ x + y \]

subject to

\[ x \geq 0 \]
\[ y \geq 0 \]
\[ 4x + y \leq 2 \]
\[ x + 2y \leq 1 \]
That’s a linear program!

• How did I find those special values $1/7, 3/7$?
• I solved some linear program.
• It’s called the dual program.

Minimize the upper bound you get, subject to the proof working.

The optimal values are the same!

Maximize stuff subject to stuff

Primal

Minimize other stuff subject to other stuff

Dual
dual to transportation: price-setting

Goal: maximize total prices so that there are no arbitrage opportunities.

\[
\max \sum_{v \in V} \text{price at } v \quad \text{s.t.}
\]

Cannot gain by buying from one penguin, transporting, and selling to another.

A pound of fish is worth $4 to me.

Cost of transport: $3/pound of fish.

Cost of transport: $5/pound of fish.
LPs and Duality are really powerful

- This *general phenomenon* shows up all over the place
  - Transportation and price-setting is a special case.

- Duality helps us reason about an optimization problem
  - The dual provides a *certificate* that we’ve solved the primal.

- We can solve LPs quickly!
  - For example, by intelligently bouncing around the vertices of the feasible region.
  - This is an *extremely powerful algorithmic primitive*. 
Today
A few gems

• Linear programming

• Random projections

• Low-degree polynomials
One of my favorite tricks
Take a random projection and hope for the best.

High-dimensional set of points
For example, each data point is a vector
(age, height, shoe size, ...)

Their shadow is a projection onto the ground.
Why would we do this?

- High dimensional data takes a long time to process.
- Low dimensional data can be processed quickly.
- “THEOREM”: Random projections approximately preserve properties of data that you care about.
Example: nearest neighbors

• I want to find which point is closest to this one.

That takes a really long time in high dimensions.

**Johnson-Lindenstrauss Lemma:**

*Euclidean distance is approximately preserved by random projections.*

Find the closest point down here, you’re probably pretty correct.
Another example: Compressed Sensing

• Start with a sparse vector
  • Mostly zero or close to zero

(5, 0, 0, 0, 0, 0.01, 0.01, 5.8, 32, 14, 0, 0, 0, 12, 0, 0, 5, 0, .03)

• For example:

This image is sparse

This image is sparse after I take a wavelet transform.
Compressed sensing continued

• Take a random projection of that sparse vector:

Random short fat matrix

Goal: Given the \textbf{short} vector, recover the \textbf{long sparse vector}.  

\[ \text{Long sparse vector} = \text{Short vector} \]
Why would I want to do that?

• Image compression and signal processing
• Especially when you **never have space to store the whole sparse vector to begin with**.

Randomly sampling (in the time domain) a signal that is sparse in the Fourier domain.

Random measurements in an fMRI means you spend less time inside an fMRI.

A “single pixel camera” is a thing.
All examples of this:

Random short fat matrix

Goal: Given the short vector, recover the long sparse vector.
But why should this be possible?

- There are tons of long vectors that map to the short vector!

Goal: Given the short vector, recover the long sparse vector.
Back to the geometry

Theorem:
random projections preserve the geometry of sparse vectors too.
If we don’t care about algorithms, that’s more than enough.

All of the sparse vectors

Multiply by

Random short fat matrix

This means that, in theory, we can invert that arrow.

How do we do this efficiently??

There may be tons of vectors that map to this point, but only one of them is sparse!
An efficient algorithm?

What we’d like to do is:

Minimize number of nonzero entries in $x$

$s.t.$ $Ax = y$

Problem: I don’t know how to do that efficiently!

Instead:

Minimize $\sum_i |x_i|$

$s.t.$ $Ax = y$

- It turns out that because the geometry of sparse vectors is preserved, this optimization problem gives the same answer.
- We can use linear programming to solve this quickly!
Today
A few gems

• Linear programming

• Random projections

• Low-degree polynomials
Another of my favorite tricks

Polynomial interpolation

• Say we have a few evaluation points of a low-degree polynomial.

• We can recover the polynomial.
  • 2 pts determine a line, 3 pts determine a parabola, etc.

• We can recover the whole polynomial really fast.
  • It’s a divide-and-conquer algorithm

• Even works if some of the points are wrong.
One application:
Communication and Storage

- Alice wants to send a message to Bob

“Hi, Bob!”

\[ f(x) = H + I \cdot x + B \cdot x^2 + O \cdot x^3 + B \cdot x^4 \]

Bob can do super-fast polynomial interpolation and figure out what Alice meant to say!
This is actually used in practice

- It’s called “Reed-Solomon Encoding”
Another application:
Designing “random” projections that are better than random

The matrix that treats the big long vector as Alice’s message polynomial and evaluates it REAL FAST at random points.

- This is still ”random enough” to make the LP solution work.
- It is much more efficient to manipulate and store!
Today

A few gems

• Linear programming
• Random projections
• Low-degree polynomials

To learn more:

CS168, CS261, ...
CS168, CS264, CS265, ...
CS168, CS250, ...
What have we learned?

CS161

Tons more cool algorithms stuff!
To see more...

- Take more classes!
- Come hang out with the theory group!
  - Theory lunch, Thursdays at noon
  - Theory seminar, usually Fridays at 3pm
  - Join the theory-seminar mailing list for updates

theory.stanford.edu
Stanford theory group: We are very friendly.
A few final messages...
1. Thanks to the TAs!!!
tell them you appreciate them!

Susanna  Jiaxi  Noa  Nick  Reyna  Duligur
Aaron  Ingerid  Haojun  Megha  Richard  Dana
Jayden  Shashwat  Maxime  Brian  Anna
THANKS to you!!!!!!