

CS 163 & CS 265: Graph Algorithms

Week 0: Introduction

Lecture 1: Course overview; web search engines and pagerank algorithm

David Eppstein

University of California, Irvine

Fall Quarter, 2022



This work is licensed under a Creative Commons Attribution 4.0 International License

Course overview

Who is running the course?

Instructor

David Eppstein
eppstein@uci.edu

Teaching assistants

Daniel Frishberg, Nitya Raju, and Evrim Ozel

Online resources

Course web site:

<https://www.ics.uci.edu/~eppstein/163/>

Online course discussion forum: Ed Discussion, via Canvas

Turn in homework and return graded homeworks: Gradescope

Confidential questions about your performance: Email us!

Course material

Lecture notes will be linked on course web site before each lecture

Weekly homeworks

Do them on your own

Midterm and final exam

In person!

But what about a textbook?

There are many graph theory textbooks but not so many on graph algorithms, and the ones I know about are at too low a level

They don't cover enough of the topics I want to cover

Instead, the course web page has links to online readings, mostly from Wikipedia

Their quality is variable but they're all we have

Expectations

What do I expect you to know already?

- ▶ Undergraduate-level algorithm design and analysis (as covered in CS 161): recursion, divide and conquer, dynamic programming, O -notation
- ▶ Some previous exposure to basic graph algorithms including depth-first search, topological sorting, and Dijkstra's algorithm
(we will go over them again, but as review)

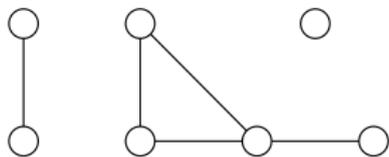
Web search

Graphs

A graph is just a set of objects related to each other in pairs

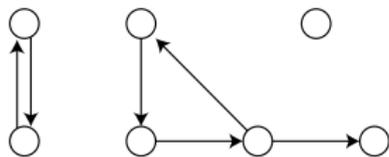
One object is called a **vertex**; more than one are **vertices**
(Do not ever call one of them a “vertice”; that is not a word.)

Pairs are called **edges** and their two vertices are **endpoints**



Undirected graph:

The endpoints are unordered



Directed graph:

Each edge goes from one
endpoint to another

We will see many real-world examples and applications

The web graph

Vertices = all web pages in the world-wide web

Directed edges = links from one web page to another

The image shows two browser windows side-by-side. The left window displays a 'Tentative Schedule of Topics' for a course on Graph Algorithms, listing various graph theory topics over several weeks. The right window shows the Wikipedia page for 'PageRank', which includes a diagram of a network graph with nodes A through F and their respective PageRank values.

Tentative Schedule of Topics

Week 1.
Extracting information from graph structure: [Web search engines and the PageRank algorithm](#), [Computer representation of graphs](#) and the [decorator pattern](#).
[Web crawler](#) case study. Review of [depth-first search](#) and [breadth-first search](#).

Week 2.
[Tarjan's algorithm](#) for [strongly connected components](#).
[Maze and river network simulation](#) via [invasion percolation](#) case study. [Minimum spanning trees](#) and their properties.
[Prim-Dijkstra-Jarnik algorithm](#), [Boruvka's algorithm](#), and [Kruskal's algorithm](#).

Week 3.
Holiday Monday, January 18: Martin Luther King, Jr. Day
Spreadsheet case study. [DAGs](#) and [topological ordering](#).
[Critical path scheduling](#) case study. Shortest and longest paths in directed acyclic graphs.

Week 4.
[Road map path planning](#) case study. [Shortest paths](#) and [relaxation algorithms](#).
[Dijkstra's algorithm](#), [Bellman-Ford algorithm](#), [Johnson's algorithm](#).
[A* algorithm](#), [landmark-based distance estimation](#).
[Preference voting case study](#) and the [widest path problem](#).

Week 5.
Transportation scheduling case study. [Euler tours](#) and the [traveling salesperson problem](#).
[Approximation algorithms](#) and the [approximation ratio](#). [MST-doubling heuristic](#), [Christofides' heuristic](#).
[Baseball elimination case study](#). [Maximum flow problem](#), [minimum cut problem](#), [max-flow min-cut theorem](#).
[Augmenting path \(Ford Fulkerson\) algorithm](#).

Week 6.
[Exponential-time dynamic programming](#) for the [traveling salesperson problem](#).
[Baseball elimination case study](#). [Maximum flow problem](#), [minimum cut problem](#), [max-flow min-cut theorem](#).
[Augmenting path \(Ford Fulkerson\) algorithm](#).

Week 7.

Search Wikipedia
Go Search
contribute
Help

PageRank [edit]

Algorithm. From Wikipedia, the free encyclopedia

PageRank (PR) is an algorithm used by Google Search to rank web pages in their search engine results. PageRank was named after Larry Page,^[1] one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google:

PageRank works by counting the number and quality of links to a page to determine

Mathematical PageRanks for a simple network are

Diagram description: A network graph with nodes A, B, C, D, E, F. Node B is the largest (PageRank 38.4), followed by C (34.3), E (8.1), D (3.9), and A (3.3). Node F is the smallest (1.6). Arrows indicate directed edges: A to B, B to C, C to B, C to E, D to B, D to E, E to B, E to C, E to F, F to E, F to C, and several smaller nodes (1.6) pointing to E.

www.ics.uci.edu/~eppstein/163/

en.wikipedia.org/wiki/PageRank

Web search engines

You type words or phrases

Database server looks up matching web pages

Shows them to you **in some order**

Examples: Google, Bing, DuckDuckGo, ...

A big change in the mid-1990s

Previous search engines:

Order search results by **text** (ignoring the graph structure)

Pages with many matching words shown first

Google (1998):

Order by **number of incoming links** (graph structure)

Intuition: if many other pages link to it, it's probably good

Refined version: Give more weight to links that come from other pages with many incoming links

This worked much better!

How Google worked circa 1998

(How they work today: known only to Google insiders)

When you make a query on Google's servers:

- ▶ Use text data structures (beyond the scope of this class) to find a set of matching web pages (limited to 500 results)
- ▶ Show these pages to you in sorted order by **pagerank**

What is pagerank?

- ▶ A number associated with each web page
- ▶ Bigger number = earlier in search results
- ▶ Computed by Google using only the link structure (does not depend on text of page)

Pagerank, definition 1 (mathy version)

$$\text{pagerank}(x) = 0.05 \frac{1}{\# \text{ vertices}} + 0.95 \sum_{\text{edge } y \rightarrow x} \frac{\text{pagerank}(y)}{\# \text{ edges out of } y}$$

Gives large system of linear equations that can be solved to compute pageranks

When x has many incoming links from pages with high pageranks, its own pagerank will be high

But why this equation? What does it mean?

Pagerank, definition 2 (intuitive version)

“Lazy web surfer”: model of a person looking at web pages

- ▶ Start at a random web page
- ▶ Repeatedly go to a new page:
 - ▶ Probability 0.95: choose a random outgoing link
 - ▶ Probability 0.05: get bored and start at a new random page

At each step i , each web page has some probability $P_i[x]$ of being the one the lazy web surfer is looking at

$$\text{Initially, } P_0[x] = \frac{1}{\# \text{ vertices}}$$

After enough steps, P_i will get very close to a “stable probability distribution” $P_i[x] \rightarrow P_\infty[x]$

Pagerank is just this limiting probability: $\text{pagerank}(x) = P_\infty[x]$

How to compute pagerank? Option 1

We have a large set of linear equations

$$\text{pagerank}(x) = 0.05 \frac{1}{\# \text{ vertices}} + 0.95 \sum_{\text{edge } y \rightarrow x} \frac{\text{pagerank}(y)}{\# \text{ edges out of } y}$$

Use Gaussian elimination to solve them

But that takes time $O(n^3)$, far too slow when $n = \text{billions}$

How to compute pagerank? Option 2

Simulate the lazy web surfer

Estimate pagerank as the number of times
the simulated surfer visits each page

But to get an accurate estimate we need to run the simulation
long enough to get many visits to each page
(theoretically at least proportional to $n \log n$ steps)

Still too slow

How to compute pagerank? Option 3

Compute probability $P_i[x]$ for i th step of lazy surfer, $i = 0 \dots 10$

Estimate pagerank $\approx P_{10}$ accurate enough for application

The equations are almost the same!

$$P_0[x] = \frac{1}{\# \text{ vertices}}$$

and, for $i > 0$,

$$P_i[x] = 0.05 \frac{1}{\# \text{ vertices}} + 0.95 \sum_{\text{edge } y \rightarrow x} \frac{P_{i-1}[y]}{\# \text{ edges out of } y}$$

Linear time if we can loop over all edges $y \rightarrow x$ quickly

How do we do this? Next time...

The morals of the story

Throwing away other information about the vertices of a graph and using only their link structure can still provide meaningful information about the graph

By doing this, Google produced a much better search engine than previous competitors and dominated the search engine market

Getting it to work required an efficient algorithm for pagerank (for problem sizes so big they don't fit into a single computer)