Great Ideas in Computing

University of Toronto CSC196
Winter/Spring 2019

Week 6: October 19-23 (2020)

Announcements

- I added one final question to Assignment 2. It concerns search engines. The question may need a little clarification.
- I have also clarified question 1 where I have defined (in a non standard way) the meaning of a *strict* binary search tree which is what I had in mind. Please answer the question for a strict binary search tree. If you answered the quiz question for a non strict binary search tree withn a proper explanation you will get full credit.
- Quick poll: how many students feel that Q1 was a fair quiz?
- A1 has now been graded by Marta. I will scan over the assignments and hope to release the grades later today. If you plan to make a regrading request, you have up to one week to submit your request. You must specify clearly why you feel that a question may not have been graded fairly. In general, students did well which is what I expected.

Agenda for the week

- We will continue to discuss search engines. We ended on what is slide 10 (in Week 5) on Friday and we will continue with where we left off.
- I was surprised that in our poll, most students felt that the people advocating the "AI view" of search "won the debate" whereas today I will try to argue that the people (e.g., Salton and others) advocating the "combinatorial, algebraic, statistical view" won the debate as to current search engines. But things change and ML is playing an increasing role in search.
- If time permits, we will discuss networks (also known as graphs). In particular, we will discuss *social networks*. Later in the term, we may also discuss the internet.

Key word search

At a very very general level, we can think of current search as the following process:

1. A user converts an "information need" into a query (i.e. a set of key words)
2. The search engine is then an algorithm for the mapping: query $\times$ {collection of documents} $\rightarrow$ <ranked list of "relevant documents">.
3. Upon receiving highly ranked documents, the user may choose to refine the query.
4. This process continues until the user is either satisfied or gives up. How often do you have to refine your queries?

As we discuss the ideas behind key word search in search engines, it should be noted that there are many specific ideas and engine specific details that go into making a search engine successful (in terms of the quality, speed, and coverage) and these ideas and details are kept reasonably confidential. Why?

Why the secrecy?

There are two main reasons for not disclosing specific search engine ideas and details:

- These ideas are trade secrets that give a company an edge
- Perhaps less obvious, knowing exactly how a compnay does its searches allows one to easily spam documents so as to raise their ranking (and hence lower the quality of the ranking). Similarly, advertisers can "spam" their advertisement to improve their chance of getting an ad slot.

So please be advised that what I am discussing is just the high level ideas and not the specifics say being utilized by Google, Yahoo or Microsoft.

It clearly took significant progress in technology (i.e., the dramtic improvement in the speed and memory of large numbers of distributed machines) to make key word search as successful as it is today. Equally important, many significant algorithmic ideas plus extensive and ongoing experience with user requests has been necessary for search engine success.

However, collecting information from user interactions is, of course, an important privacy issue.

The challenge of real time information retrieval

In addition to algorithmic ideas used to improve search quality (i.e., precision, recall), commercial search engines deal with huge collections of documents and must return responses in what appears to be "real time".

Estimates of the size of the web vary. One site (WorldWideWebSize) provdes daily reports on the size of the web: "The Indexed Web contains at least 5.42 billion pages (Sunday, 04 October, 2020)"

Precision in a set of documents $\mathcal{D}$ (for an information need) is defined as the fraction of documents that are relevant. In a ranked list we can say that precision means the fraction of documents in (say) the top 10 highest ranked documents that are relevant.

Recall in a set of documents $\mathcal{D}$ is defined as the fraction of all relevant documents contained in the entire collection $\mathcal{C}$ that are in the set $\mathcal{D}$. In a ranked list of retrieved documents (with possibly thousands of relevant documents), we can say that recall is the fraction of (say) the ten highest ranked documents in $\mathcal{C}$ that occur within the top 10 documents in $\mathcal{D}$.

Do we want diversity in the documents retrieved?

We may (or may not) want the highest ranked documents to reflect some desired diversity.

For example, what if I provide the query "What has Donald Trump accomplished as US president"? Do I want just what is reported as his positive accomplishments? Or do I want just the negative aspects of his presidency? Or do I want a diversity of opinions?

Similarly, if I ask whether the stock market has recently been rising? Do I want some overall assessment, or do I want reports on different sectors of the market?

Even for a more classical and now perhaps a more mundane example, when I ask for recent information about "jaguars", do I mean the car, the animal, or the NFL football team? I probably only want one of these. Even if I make my request clear, a search engine has to avoid ambiguous meanings.

Should a search engine use my previous history of requests to better identify the most relevant documents personalized for me?

The basic bag of words model

Suppose $\mathcal{C} = \{D_i\}$ is a collection of web documents (URLs).

- We can treat each document as a *bag of words*. Let's just say 200 words per document as some very rough average.
- Each query can also considered as a very small bag of words (or terms, which can be defined as common two or three words occuring consecutively). Most queries are two or three words. One estimate is an average of 2.2 words per query.
- The most naive approach. Find all the documents that contain all the words (and terms) in the query. As a naive first approach call these the "relevant documents".
- The most naive way to find all these (potentially) relevant documents would look at each document and check if all the query words occur.
- Even if all the documents were stored locally (which is not possible), what would be a rough estimate for the time to find all the relevant documents?

A quick calculation

You can do a quick calculation: compute $|\mathcal{C}| \cdot \frac{\text{number words}}{\text{document}} \cdot \frac{\text{number words}}{\text{query}}$ and then divide by $\frac{\text{number comparisons}}{\text{second}}$ to estimate the time for naively looking for documents that contain all the query terms.

Let's say that we have approximately 5 billion URLs, 200 words per document, 2 words per query which naively would result in $2 \cdot (10)^{12}$ comparisons. And let's say $(10)^7$ comparisons per second. Then a query would take $2 \cdot (10)^5 = 200,000$ seconds.

OK I might have some miscalculations but clearly this is *not* "real time" and not even feasible.

Making search feasible

One simple idea but very useful idea is the following. When a search engine *crawls* the web to find documents, it indexes documents so that for each *term* (i.e., word and frequent 2 word and 3 word phrases) it maintains a sorted list of documents that contain that term. We usually ignore common articles such as "the", "an, etc.

A term may also represent a number of strongly related terms. For example, a match for "cook" might be satisfied by "cooking".

While it doesn't happen that often, what do we get when no document matches the query terms?

Making search feasible

One simple idea but very useful idea is the following. When a search engine *crawls* the web to find documents, it indexes documents so that for each *term* (i.e., word and frequent 2 word and 3 word phrases) it maintains a sorted list of documents that contain that term. We usually ignore common articles such as "the", "an, etc.

A term may also represent a number of strongly related terms. For example, a match for "cook" might be satisfied by "cooking".

While it doesn't happen that often, what do we get when no document matches the query terms?
You can get spelling suggestions, or maybe get a partial match, and sometimes be told that no documents match your query or there are no good matches but still get some suggested matches.

What can happen often is that there be too many documents matching the query terms. So as we already suggested we really need a ranked list of documents that are the "most relevant" for the individual user.

The vector space model and ranking documents

Instead of simply matching for query terms, we want to account for the fact that the occurence of certain terms are more important for relevance.

Gerald Salton's idea was that a document (and a query) are represented by a vector of weighted counts of words/terms. Here are some ways to weight the occurences of terms in a document.

1. Count the number of occurences of a query term in a document, and better yet normalize this count by the relative frequency of terms in "the corpus of documents". This normalized count is called *tf-idf* standing for *term frequency-inverse document frequency*. Terms that occur infrequenly throughout the corpus but appear frequently in a document should be weighted more. Wikipedia quotes a 2015 study that states "83 % of text based recommender systems in digital libraries use tf-idf".
2. Terms that appear in the title of the document or the title of a section heading should be given higher weights.
3. Terms that appear in the *anchor text* are important.

The vector space model continued

The above ideas for weighting terms are independent of the user queries. In contrast, we could also give higher weights to terms that relate to an individuals interests (say as learned by previous searches).

There can be many other ways to weight terms say by using machine learning techniques.

Now once we adopt this vector space representation, we can measure the similarity of a document and a query by say the cosine of these vectors.

An additional idea (in additional to the term similarity of the document and the query) is to expoint the "popularity" of a document. Popularity of a document in Google was done using *page rank* which is basically a random walk on the graph defined by the hyperlinks. This leads to a stationary distribution (i.e., an equilibirum) on the vertices (i.e., the relevant docuemnts).

Some further comments on the history of search engines

Page rank was touted as an essential idea in the early days of Google search but not clear how much of a role it now plays.

At about the same time as page rank, Jon Kleinberg introduced another graph based popularity method called *hubs and auhtorities* which was used in IBMs search engine (which they never commercialized).

With regard to td-idf (now accepted as an important idea), I saw the following comment in a web post (Language Log)
https://languagelog.ldc.upenn.edu/nll/?p=27770)
"one of Marvin Minsky's students once told me that Minsky warned him 'If you're counting higher than one, you're doing it wrong'. Still, Salton's students (like Mike Lesk and Donna Harman) kept the flame alive."

Marvin Minsky is recognized as one of the pioneers of artificial intelligence.

Why is search so profitable?

Companies such as IBM and (initially) Microsoft did not try to commercialize search not recognizing, the profitability of search. Indeed, should one charge for information or should the business model be based on advertising? Or it possible that search would not be profitable?

We now know that search has turned out to be extremely profitable for companies based on advertising. The main way that Google and other comapnies sell advertising for search has spawned major research in algorithm design and and auction theory.

We can view the process of assigning queries to advertisers (say wanting to display an *ad* as an *online biparitite graph matching problem*).

When a query arrives it needs to be assigned to one (or more, depending on how many addvertising slots will be displayed) ads.

The "adwords" assignment problem

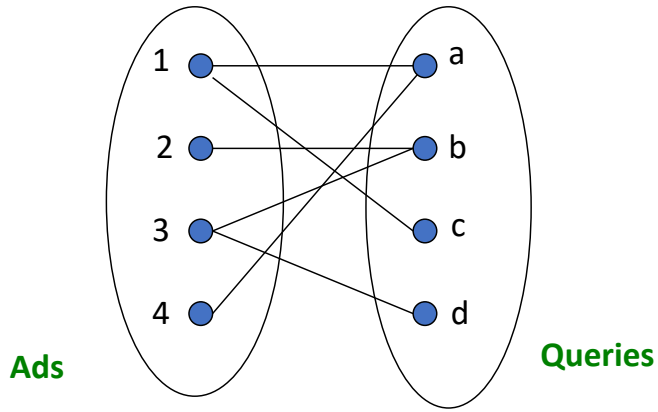

Figure: Figure taken from USC lecture notes by Rafael Ferreira da Silva

Each advertsiser may have a budget (say for a given day) and indicates for given queries (or keywords) what it is willing to pay for that query but never exceeeding its budget for all the queries assigned to that advertiser.

The search engine adjusts this advertiser *bid* for a query based on how well it thinks the ad matches the query and then decides whether or not to assign an advertising slot to an advertiser and the price paid by the advertiser (depending on the slot) for each click by search users for the ad.

The semantic web

We will end our discussion of search engines about where we began when I said, like other great ideas, sometimes these great ideas become so entrenched that it is hard to make further progress.

Is this the case with key word search? What kinds of "information needs" are beyond today's search engines? See 2008 "Ontologies and the Semantic Web" article by Ian Horrocks and also his 2005 Lecture by the same title.

The vague goal of the semantic web is "to allow the vast range of web-accessible information and services to be more effectively exploited by both humans and automated tools."

A more specific goal is to *integrate* information that occurs in the web but not in one decument.

Some specific examples of information that might not exist in any one document

One example Horrrocks gives is to retrieve a "list of all the heads of state of EU countries". Of course, once such an example is given, it is likley (as in this example) that one can successfully find the required information in a single query.

"The classic example of a semantic web application is an automated travel agent that, given various constraints and preferences, would offer the user suitable travel or vacation suggestions". This example still seems beyond something we can easily do with current search engines.

I decided to create the following query "list of all computer scientists whose last name is Cook". In my first search, most of the retrieved documents are not useful but the first of the retrieved documents is for Stephen Cook and the second document is a very incomplete list of computer scientists. Repeating this same query, I get this incomplete list of computer scientists, followed by famous people with last name Cook (eg James Cook) and then another incomplete list of computer scientists.