

Multimedia Retrieval

Chapter 3: Web Retrieval

Dr. Roger Weber, roger.weber@ubs.com

[3.1 Motivation](#)

[3.2 Ranking in Web Retrieval](#)

[3.3 Link Analysis](#)

[3.4 Literature and Links](#)



3.1 Motivation

- Web Retrieval was first performed like ordinary text retrieval. But soon it was clear that web retrieval is entirely different. At the time Google started, the earlier search engines all used vector space retrieval or some form of probabilistic retrieval. Google was the first engine to use ordinary Boolean retrieval but enhanced with a clever ranking system that we will consider in the following. Although the mechanics of the Google search are well kept secrets, we know from the early prototypes of Brin and Page at the Stanford University how the search engine works.
- We first consider the differences between classical and web retrieval. Not only the size varies, but also the quality and how people are searching for information:

	Classical Retrieval	Web Retrieval
Collection	controlled set	uncontrolled, incomplete
Size	small to large (20 GB)	extremely large (>10PB)
Documents	homogenous	heterogeneous (HTML, PDF, ASCII)
Structure	homogenous	heterogeneous
Links	seldom (citations of other documents)	lots of links among documents
Quality	good to excellent	broad range of quality: poor grammar, wrong contents, incorrect, spamming, misspellings, click baits
Queries	precise and structures	short and imprecise, names!
Results	small number of hits (<100)	large numbers of hits (>1,000,000)

- These days, a web search engine has to deal with **40+ billion pages, 60+ trillion unique URIs,** and an **index size of 100+PB**. A typical query returns several millions of hits but users expect the top page (or the first link) to be the most relevant for them. But how can we find the most relevant documents for queries with one or two terms given that millions of pages contain them?
 - Example query="ford": what do you expect at the top of the list? The car manufacturer, the president, or a ford to cross a river?
 - Example query ="uni basel": what should be at the top? this course? the main page of the university?
 - Example query="it": is the movie the best answer? the book by Stephen King? an IT company? the definition of "it" as a pronoun?
- With all the examples above, it is clear that the short queries are not sufficient to define what is relevant. So Brin and Page considered what users actually want to see and designed their search algorithms to optimize towards this most common information need. With all the queries above, the average user is expecting the page he/she most likely wants to visit. Hence, if more people are interested in ford as the car manufacturer, than that page should be at top. The answers may change over time! As we see with "it", a recently released movie, the results can depend very much on current events and rankings can drastically change over time.
- In summary: when the context is not clear, and when the query is ambiguous, a web search should return the page at the top that most people consider relevant.
 - This may not be your interpretation of "what is best". But it is the average interpretation of all internet users.
 - This concept is not entirely new: broadcast stations have always played those songs that most people (in the area) like best. The term "pop song" indicates an entire music industry that is chasing the mainstream listeners.

3.2 Ranking in Web Retrieval

- In this chapter, we consider some of the known elements of the Google search (as published by Brin and Page during their Stanford time). Google was the first engine that focused entirely on the most common query and expected search results, and added a number of features to web search to attract investors and advertisers. After almost 20 years, it is still the market leader even though others have caught up with regard to size, precision, and performance.
- Google's search starts at the feature extraction level. It was the first engine to consider HTML tags and the links between documents. For each term, not only the number of occurrences is extracted but further attributes like: font size, font styles, meta tags, link tags and many more. We will describe their usages as we discuss the individual ranking components.
- Unlike in classical retrieval, the similarity function of Google does not relate on terms only. It takes different aspects into account, weighs them, and sums them up to final similarity score. In the following sections, we consider:
 - Proximity of terms
 - Visual attributes of terms and their usage in links
 - PageRank
 - Other extensions
- Initially, the search engine interpreted a query automatically as list of mandatory terms (all connected with AND) and only rated documents that contain all query terms. Meanwhile these hard constraints have somewhat disappeared: for instance, spelling mistakes are automatically corrected (or added to the query). Rare terms may not have to appear in the documents (partial match).
 - Google reports the number of hits for every search (sometimes just as a guess)
 - Engine still uses inverted lists, but a lot of optimization went into the code to avoid a full scan through all postings

3.2.1 Proximity of Terms

- Assume we are search with “White House” and we have the following documents:

“The white car stands in front of the house“

“The president entered the White House“

Which one would you expect to match the query better?

- Brin and Page, with the example of Bill Clinton, realized that most people implicitly assume proximity between query terms in the documents. Especially, with the most popular search type (celebrities, brands, names), the implicit proximity assumption is key. If you are looking for “Bill Clinton”, you do not want to find:

“....Bill Rosweld was winning....and John Clinton raised his hand...”

“...the dollar bill was on the floor ... Mrs. Clinton went home...”

- The average user is expecting that the query terms are next to each other (or at least very close) and that the order is as given in the query. Try it yourself:
 - “White House” → returns the official homepage for the White House
 - “House White” → returns another page at the top with the name “House White”
- To enable a similarity value based on proximity, Google uses two options:
 - n-grams: add “white house” as a new n-gram term and use it for searches. This ensures that hits have both words in proximity
 - extract position information from the document, calculate proximity for terms in the document, and push similarity values if proximities are high

- With the position information, we can evaluate a simple metric for proximity. The following is a rough sketch of what Google's early search engines did, but still applies in one or the other way in today's version. The basic idea is to store not only term frequencies in the inverted lists but the positions of occurrences in the documents (so-called hit-lists). For example: consider the query "White House".
 - We read the hit lists for each of the terms and a given document from the inverted file:

```
hitlist['white'] = [1, 13, 81, 109, 156, 195]
hitlist['house'] = [2, 82, 112, 157, 189, 226]
```

The hit lists are then combined pairwise to obtain all interesting combinations. This leads to the following pairs that bear some proximity information between "white" and "house"

```
pairs = [(1, 2), (81, 82), (109, 112), (156, 157), (189, 195)]
```

- Proximity is expressed as the distance between the elements of pairs and is quantized to a small set proximity values (in this example we use values between 1 and 10):

```
proximity = [1, 1, 3, 1, 6]
```

In this simplified approach, '1' denotes adjacent terms, '3' close-by, and '7' distant. Any quantization is thinkable at this point as long as it matches user expectations. Based on these proximity values, a histogram is built, i.e., counting how often a proximity values occurs.

```
pbins = [3, 0, 1, 0, 0, 1, 0, 0, 0, 0]
```

- Finally, the bins are weighted and summed up to a proximity score:

```
weights = [89, 55, 34, 21, 13, 8, 5, 3, 2, 1]
score_proximity =  $\sum_i \text{pbins}[i] * \text{weights}[i]$ 
```


3.2.2 Term Frequencies and HTML Attributes

- Classical retrieval was simply counting term frequencies regardless of where they occur in the text. For HTML documents, we may want to take the surrounding tags into account and add more weight to occurrences if the term is part of `<title>`, `<h1>`, `` or `<i>`.
- Brin and Page went a step further: they realized in their research that hyperlinks not only describe the document containing the anchor text but also provide additional keywords for the referenced web pages. Even more, anchor texts tend to be short and concise and so we obtain very relevant keywords that describe the referenced document most accurately. Nobody is writing a hyperlink with a lengthy anchor text about what is expected at the referenced location. On the other side, we frequently encounter anchor texts with low relevance like “click here”, “back to start” or “follow me”.
- The very first search engines were plagued by spammers: the authors wrote documents containing numerous key words, sometimes repeated 1'000 times, to push their web pages to the top of the search results. Google stopped these spamming approaches by firstly weighting terms in (external) anchor texts much more (what others say about you), and secondly ceiling the number of occurrences at a low number. In contrast to the classical vector space retrieval, Google was not ranking based on term frequencies and idf-weights but used a more elaborated scheme to assess the significance of a term to describe the content.
- Again, we only know what Brin and Page did as part of their research. Meanwhile, Google has extended its scheme to describe documents better and to prevent spammers, click baits, and other dubious pages to appear at the top of searches. Their original approach had 3 steps:
 - Describe the document with the key words and their tags
 - Add keywords of anchor texts to the referenced document
 - When creating the index, sum up the weighted occurrences to a single term score

- Consider a web page and a term “university”. We extract all the term occurrences and their surrounding tags, and associate a term occurrences whenever an anchor text contains “university” and points to the page:

The diagram illustrates the process of extracting term occurrences from a web page. On the left, a list of universities is shown, with 'Erasmus Universität Basel' circled. An arrow points from this circled text to a snippet of the university's website. The website snippet shows the header 'UNIVERSITÄT BASEL' and a navigation menu. On the right, a list of HTML tags is shown, with the term 'university' circled. The tags are: <title> ... university ... </title>, <h1> ... university ... </h1>, ... university ... , <p> ... university ... </p>, <td> ... university ... </td>, <i> ... university ... </i>, <h1> ... university ... </h1>, ... university ... , and <h1> ... university ... </h1>.

- Terms are extracted as usual but we keep <tag>-information and count how often a term-tag pair occurs (if multiple tags are active, for each tag a separate pair is added). Whenever we encounter a hyper link to our web page, we add the terms of the anchor text:

```
terms = [...(university, <title>,1),...(university, <h1>,2),...(university, <b>,10),
...(university, <p>,55),...(university, <td>, 2),...(university, link, 23)]
```


- Upon creating the index, a final score for a term is computed using an upper limit (e.g. 100) for the occurrences and weights depending on the tag:

```
weights[tag → weight] = [<title> → 13, <h1> → 5, <p> → 1, link → 55]
score[university] =  $\sum_{\text{terms}[i,1]=\text{university}} \min(100, \text{terms}[i,3]) * \text{weights}[\text{terms}[i,2]]$ 
```

- Interestingly, we can now search for documents we have never seen (but only heard about). The scores of the query terms are added up for the ranking (together with all other scores).

3.2.3 PageRank

- Assume we search with the key words “uni basel”. What would you expect to be at the top of the ranking? The two pages below both qualify as they have the keywords in prominent places.
 - As a student of this course, you are obviously visiting the course page more often and hence it is more important than the home page of uni basel.
 - However, the average student (or the average web surfer) is more interested in the home page than in a single course.
 - Looking only at key words is not sufficient. Yes, we can take hyperlinks into account and consider the page with most matching keywords in hyperlinks as being more relevant. But that would not help with very popular brands like Apple and Google and with 1000s of websites discussing various aspects of the brands (and hence competing for the top spots in searches).



> INFORMATIK . DEPARTEMENT MATHEMATIK UND INFORMATIK

Studium/Education

Forschung/Research

Lehre/Teaching

Team

Kolloquien

FAQ

Vorlesung: Multimedia Retrieval

VV-Nr	15731-01
Dozierende	Roger Weber
Zeit und Ort	Mo 08:15 – 12:00; Seminarraum 05.001, Spiegelgasse 5
Start	20.09.2017
Voraussetzungen	Basics of programming
Inhalte	Introduction to information retrieval with a focus on text documents and images. Main topics comprise extraction of characteristic features from documents, index structures, retrieval models, search algorithms, benchmarking, and feedback mechanisms. Searching the web, images and XML collections demonstrate recent applications of information retrieval and their implementation.
Leistungsüberprüfung	Lehrveranst.-begleitend Bitte beachten: Oral exam Tuesday, 9 January 2018 (3-6 p.m.) and Tuesday, 16 January (3-6 p.m.) Room: 00.003, Spiegelgasse 1
Kreditpunkte	6
Skala	1-6 0,5
Module	Wahlbereich Master Informatik: Empfehlungen (Master Informatik 10) Modul Praxis aktueller Informatikmethoden (MSF - Informatik (Studienbeginn vor 01.08.2016)) Modul Applications of Distributed Systems (Master Computer Science 16) Modul Applications of Machine Intelligence (Master Computer Science 16) Modul Concepts of Distributed Systems (MSF - Computer Science)
Belegen	Services (Anmeldung mit Passwort)

(Lecture) Notes / Slides

0. Introduction (pdf)

1. Performance Evaluation (pdf)

2. Text Retrieval (pdf)

3. Web Retrieval (pdf)

4. Basic Image, Audio, and Video Retrieval (pdf)

5. High-Level Features with Machine Learning (pdf)

6. Similarity Search (pdf)

7. Relevance Feedback (pdf)

Exercises

1. Evaluation Task | Naive Improved | Solution

English

Organisationseinheiten


Dokumente

Informationen...

Standorte

Suche

Personen



Universität
Basel

Aktuell


Studium

Forschung

Innovation

Weiterbildung

Universität



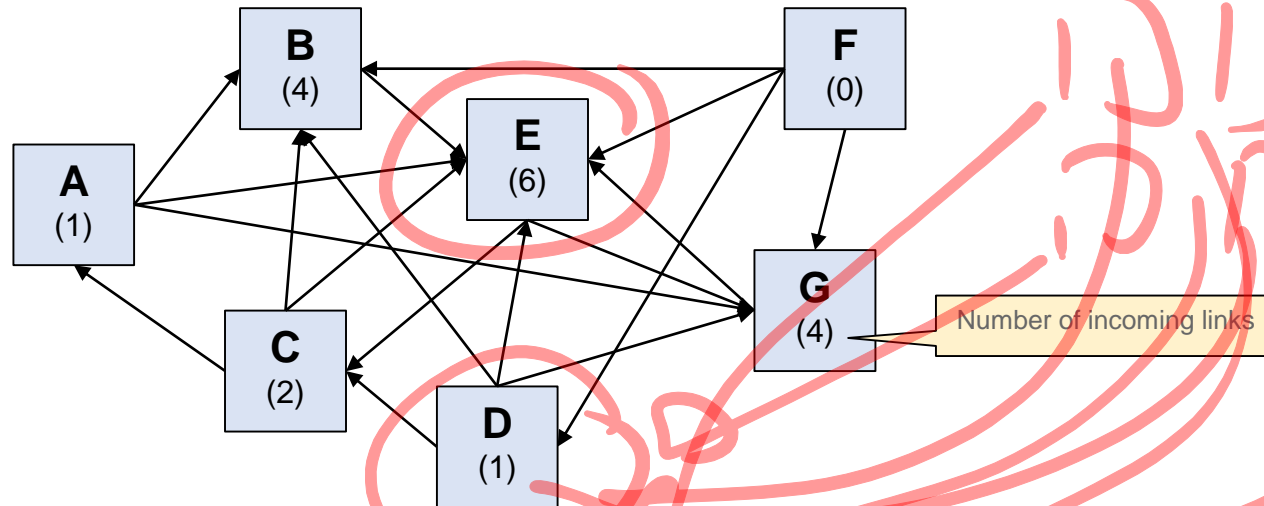
Lasker-Preis
Grosser Empfang für Michael Hall

●

●

||

- PageRank was invented by Larry Page (one of the Google founders) during his research time at Stanford. His preliminary idea was to consider a page more relevant (for the average user) if it has many incoming links. Consider the following example:



- PageRank assigns an absolute ranking to all pages in a graph like the one above. A naive approach considers only the incoming links. In the running example, E would be top ranked as it has the largest number (6) of incoming links. B and G follow with 4 links, then C with 2 links, D with 1 link, and finally F with no incoming links. This also would reflect our intuitive understanding of importance as E appears to be indeed the center of the graph.
- Consider B and G: both have 4 incoming links and hence tie on 2nd place. If we look closer, we see that G is referenced by E while B is referenced by less important pages. In other words, simply considering incoming links is not enough, we want to also weight the link based on the quality of the source.
- Note that incoming links as a ranking measure is not very robust. A web page author can easily create thousands of incoming links and thus optimize the ranking of the page (link farms).

- PageRank is based on a random surfer model: a user navigates through the web by clicking on links. At some point, the user switches randomly to another page (e.g., picked from the bookmarks). We make the following assumptions for the random surfer model:
 - When on a page, the user can perform two actions: 1) with a probability α he follows a link, and 2) with a probability $1 - \alpha$ he enters a random URL (from his or her bookmarks, or by search)
 - For 1) user navigates by picking a random link (all links are equally probable)
 - For 2) if a page has no outbound links (sink), the user picks a random URL
- We can interpret PageRank as a Markov chain in which the states are pages and the transitions are the links between pages. The PageRank of a page is then the probability that a random surfer is visiting that page. The higher the value, the more popular the page and we expect it to be more often at the top of the search results.
 - To compute the probability equations, we consider two aspects: 1) all incoming links, and 2) a random switch to the page. Let $q \rightarrow p$ denote that q contains a link to p , and let $L(q)$ be the number of outgoing links from q . The set of all pages \mathbb{P} contains $N = |\mathbb{P}|$ elements. We can then express the PageRank with the given probability α that the user follows a link as:

$$PR(p) = \frac{1 - \alpha}{N} + \alpha \cdot \sum_{q \rightarrow p} \frac{PR(q)}{L(q)} \quad \forall p \in \mathbb{P}$$

- Interpretation: to obtain a high PageRank, the number of incoming links is still important but each incoming link is weighted by the PageRank (aka importance) of the source page. If a page has several outgoing links, its PageRank is evenly distributed to all referenced pages. The method is more robust and adding artificial links does not help to push the PageRank. On the other hand, it favors older pages that are well connected while new pages, even very good ones, lack the number of links necessary to obtain high ranks.

- Evaluation: the PageRank equation defines an implicit equation system that can be solved iteratively. Let $\mathbf{r} \in \mathbb{R}^N$ be the vector holding all PageRanks of documents in \mathbb{P} . We represent the links between pages with a matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$:

$$M_{i,j} = \begin{cases} \frac{1}{L(p_j)} & \text{if } p_j \rightarrow p_i \\ \frac{1}{N} & \text{if } p_j \text{ has no outgoing links} \\ 0 & \text{otherwise} \end{cases}$$

With this, we can rewrite the PageRank equation as follows:

$$\mathbf{r} = \frac{1 - \alpha}{N} \cdot \mathbf{1} + \alpha \cdot \mathbf{M}\mathbf{r}$$

with $\mathbf{1}$ being a column vector of length N with only ones

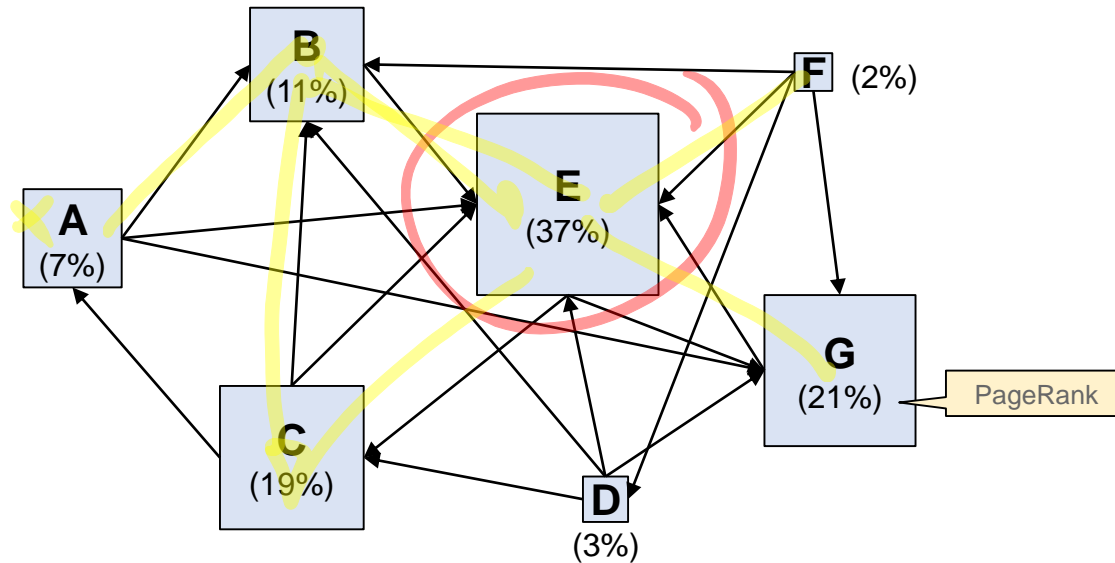
We now can describe the iterative process to solve the above equation system. Let $\mathbf{r}^{(t)}$ denote the PageRank values of pages after the t -th iteration:

1. Initialization: $\mathbf{r}^{(0)} = \frac{1}{N}$, $\alpha = 0.85$
2. Iteration:
 - $\mathbf{r}^{(t+1)} = \frac{1 - \alpha}{N} \cdot \mathbf{1} + \alpha \cdot \mathbf{M}\mathbf{r}^{(t)}$
 - stop if $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}| < \epsilon$

< 100 iterations

Because of the sparse link matrix, the iteration converges rather quickly and it can easily scale to larger document sets. In their original study, Brin and Page reported 52 iterations of a network with 322 millions of links, and 45 iterations for 161 millions of links. They concluded that the number of iterations is linear to $\log(n)$ with n being the number of edges. Due to the sparse matrix, compressed representations are used to minimize memory consumption.

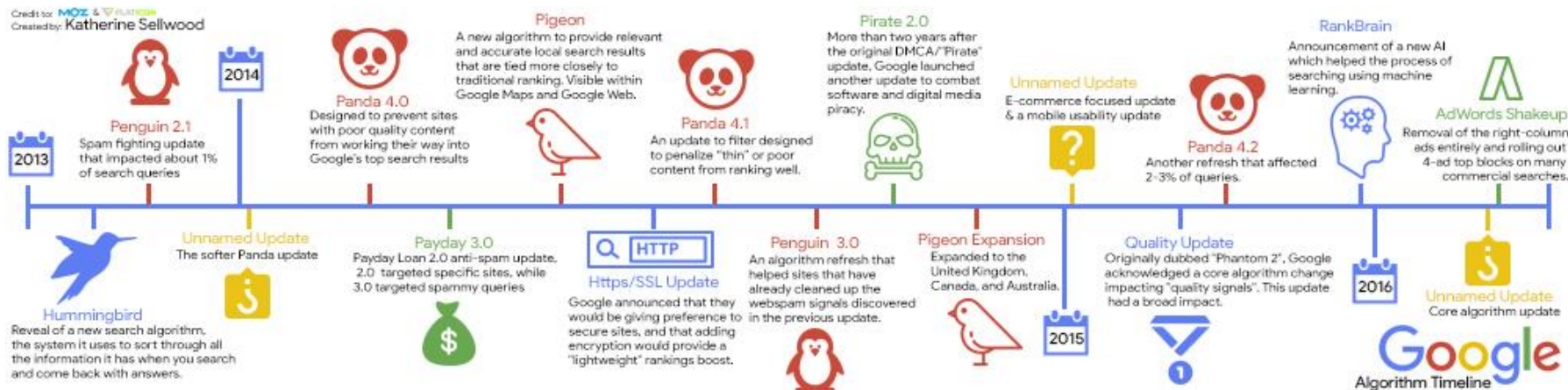
- **Example from before:** let us apply the PageRank formula to the graph below. The size of the nodes represent now the PageRank and the values the probabilities that a random surfer visits the page:



- **Discussion:** E is still the center of the network but G and C are now more important than B. Even though B has 4 incoming links, two of them come from the least important pages D and F. On the other side, E has only two outgoing links and hence both C and G receive about 43% (with $d = 0.85$) of the PageRank of E.
- **Usage for ranking:** the PageRank is an absolute measure for the importance of a page regardless of the query. It is computed once after a complete crawl and used for all queries. Even though PageRank is an important measure, it is only one of many criteria. If we would emphasize too much on PageRank, we would only see the same sites in our search results. **Terms and proximity are equally important** but PageRank helps to favor pages that are more likely visited by users (and hence requested in the search results to be at the top). However, negative publicity pushes pages to the top as well.

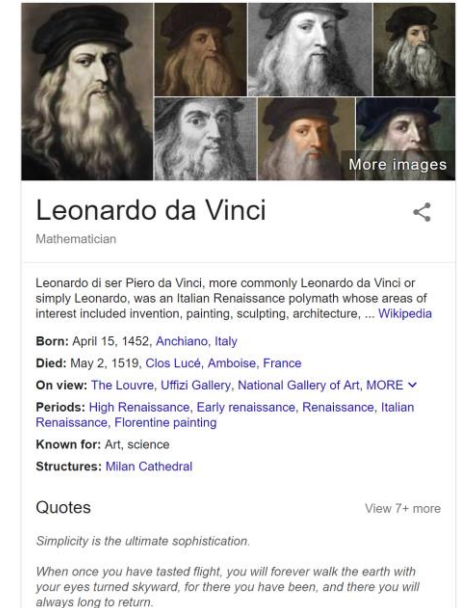
3.2.4 Further Improvements

- Google is using more than 200 criteria to compute the top results. We consider here a few of the published criteria but much is hidden within the Google algorithms as a trade secret. Other search engines like Bing use similar approaches, but Google is the best documented search engine.
- Hummingbird (2013): a series of changes themed on conversational queries and synonyms
 - First major update since 2001 focusing on data retrieval, artificial intelligence, and how data is accessed and presented to users. Especially, the integration into an immersive mobile experience was key concept. Users would no longer provide keywords but ask entire questions. Rather than searching for keywords, Hummingbird takes the context of the sentence into account. It uses synonyms to find more relevant keywords.
 - Rewards content pages over click baits, link farms, and pages with lots of advertisements. The reward helps to find relevant content related to the user's intent.
 - Considers co-citations of web pages to boost popular pages in niche topics (where PageRank is limited). Also consider keywords around anchor text to describe the referenced page.
 - Keeps users longer on Google pages by presenting integrated result tables.



- Pigeon (2014): prefers local (to the user) search results taking location and distance to business into account. Not available everywhere but continuous roll-out to more countries.
- Penguin (2012): series of updates that penalize sites for not following rules (so-called black-hat search engine optimizations) using manipulative techniques to achieve high rankings.
 - Keyword spams, i.e., excessive use of some key words
 - Sites using link farms or other techniques to push PageRank
 - Doorway pages: built to attract search engine traffic but do not provide any content
 - Page Layout Algorithm: targets web sites with too many ads or too little content in the upper part of the page (above the fold)
 - Since 2012, 7 updates of Penguin were released. Affected a small percentage (<3%) of queries. Only recently added to the core search engine
- Panda (2011): updates to lower the rank of low-quality or thin sites, especially content farms. High quality pages should return higher in the rankings.
 - Thin content: web pages with very little relevant or substantial text
 - Duplicate content: content copied from other places. Initial versions had issues to correctly distinguish sources and scrapers replicating content only.
 - Lack of trustworthiness: sources that are not definitive or verified. To avoid impact, web sites should work to become an authority on the topic. Pages full of spelling and grammatical errors.
 - High ad to content ratio: pages with mostly ad and affiliate programs as content
 - Websites blocked by users
 - Panda affects the ranking of an entire site / section rather than an individual page. A penalty remains until the next update of Panda (not part of the core search). If a site has removed the dubious parts, the penalty is removed. Affected almost 12% of queries.

- **Caffeine (2010):** improved the entire index engine and turned the batch process into a more continuous update process providing up to 50 percent fresher content
 - Historically, Google crawled the web during 30 days, created the index, and then used this index for another 30 days. Soon, the engine was extended by the freshbot which captured news content and of important pages more frequently providing fresh index data to searches.
 - With Caffeine, the entire engine was overhauled. Instead of using several layers with web sites updated at different frequencies, the Caffeine update brought a continuous update process with it. The pages are not more frequently crawled than before, but the updates would become visible more quickly.
 - Internally, the engine was switched from the MapReduce algorithm to BigTable, Google's distributed scale-out database. Caffeine operates on a 100 PB database and adds new information at a rate of a petabyte per day.
- **Knowledge Graph (2012):** a knowledgebase used to enhance semantic search results. Information is gathered from a wide range of sources
 - Collected from sources like the CIA World Factbook, Wikipedia, and similar sites.
 - Freebase (community managed content) was handed over into Wikidata
 - The newer Knowledge Vault uses artificial intelligence to derive data automatically from web content
 - As of 2016, the knowledge graphs holds 70 billion facts. There is an open Google API to programmatically access the database

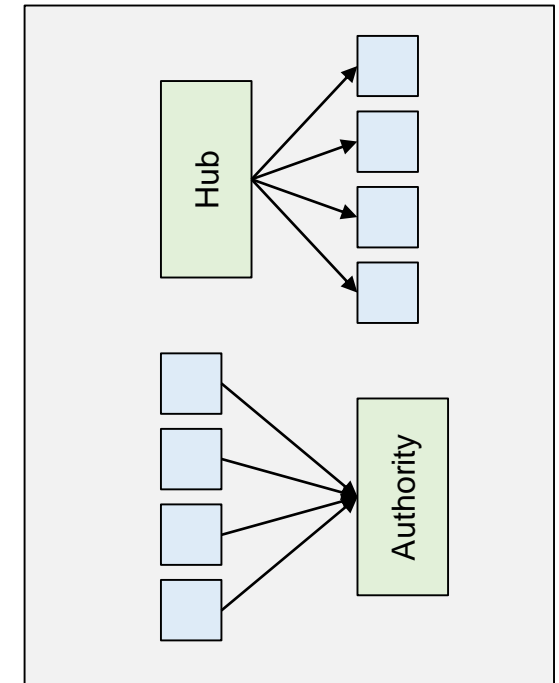


3.3 Link Analysis

- So far, we have looked at PageRank for link analysis. But there are many other ways to consider links during retrieval. A common observation is that there are two prototypes of web pages
 - **Authorities** are web pages that discuss a topic and are recognized by the community as the authoritative source for information. A good example is Wikipedia, IMBD, MusicBrainz, etc.
 - **Hubs** are web pages that group authorities in a directory like style without actually discussing the topics. Your bookmarks are your personal hub, but also web sites like Yahoo, Yellow Pages, etc.

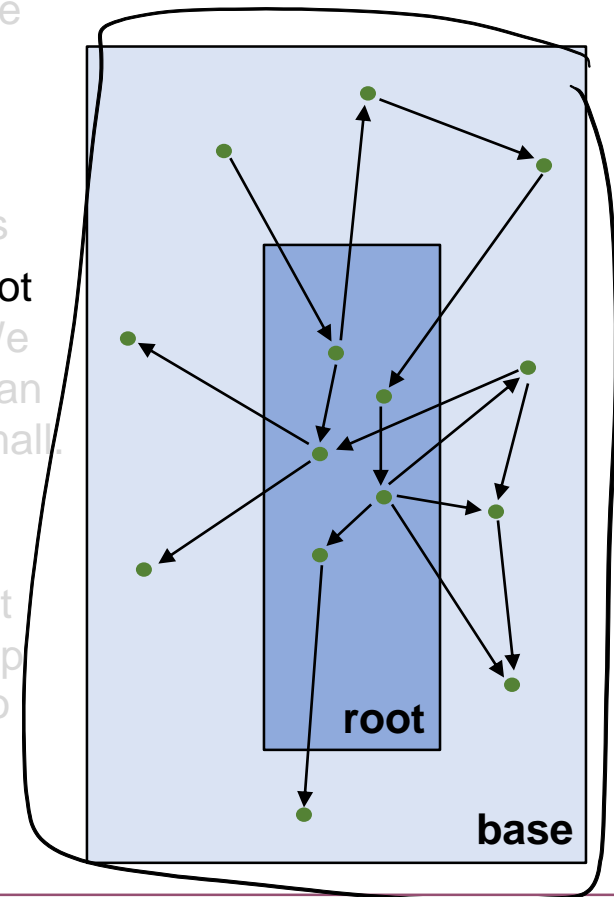
Note that PageRank was only considering how likely a user would visit the page but not whether it contains any authoritative content. PageRank is also a random walk across the entire web. The methods we consider in this section only look at the current topic of the query, hence the terms **Topic Search** is often used with these methods.

- How can we recognize a good hub and a good authority?
 - A hub is a web page with many links to authorities. We observe the typical hub structure depicted on the right side
 - An authority is web page with many incoming links from hubs. We observe a typical (sink) structure as depicted on the right side
 - To be a good hub, it must link to good authorities on the topic. To be a good authority, it must be linked by many good hubs on the topic.
 - Note: “on the topic” means that we are not just interested in the number of incoming / outgoing links, but they have to be related with the current topic. This is the biggest difference to PageRank where all links regardless of any topic are considered.



3.3.1 Hyperlink-Induced Topic Search (HITS)

- Jon Kleinberg developed the HITS algorithm in 1997. He observed the concepts of hubs and authorities in the emerging web where directories were the pre-dominant way to find information on the Internet (search engines existed but lacked the sufficient quality). To better guide searches, he introduced to metrics for a web page p :
 - $h(p)$ denotes the hub value of the page p , i.e., its ability to serve as a hub for the user
 - $a(p)$ denotes the authority value of page p , i.e., its ability to provide content to the user
- As we are only interested in a single topic, not the entire web structure is used. Instead, we create a base set with the following two steps:
 - For a query / topic Q determine the top results with the help of a search engine. This set is called the root set. It already contains a fair number of hubs and authorities, but not yet all relevant ones
 - Extend the root set with a) web pages that link to a page in the root set, and b) pages that are referenced by a page in the root set. We can remove links within the same domain (navigation links) and can limit the number of incoming / outgoing links to keep the graph small. This set is called the base set
- In practice, we need to execute several searches and downloads to compute the base set. 2b) requires downloading the pages of the root set, extracting link information, and adding the referenced pages. Step 2a) requires a search with a `link:-` clause to obtain pages that link to a member of the root set. A previous crawl of the web to obtain the link structure greatly reduces the costs for constructing the base set.



- We use the notation $p \rightarrow q$ to denote that p contains a link to q . We now can formulate the HITS algorithm as an iterative process. Assume that the base set \mathbb{P} contains N pages:

1. Initialization: $h^{(0)}(p) = a^{(0)}(p) = \sqrt{1/N} \quad \forall p \in \mathbb{P}$

2. Iteration:

- Update: $a^{(t+1)}(p) = \sum_{q \rightarrow p} h^{(t)}(q)$

- $h^{(t+1)}(p) = \sum_{p \rightarrow q} a^{(t)}(q)$

- Normalize $a(p)$ and $h(p)$ such that:

$$\sum_p a^{(t+1)}(p)^2 = \sum_p h^{(t+1)}(p)^2 = 1$$

- Stop if $\sum_p |a^{(t+1)}(p) - a^{(t)}(p)| + \sum_p |h^{(t+1)}(p) - h^{(t)}(p)| < \epsilon$

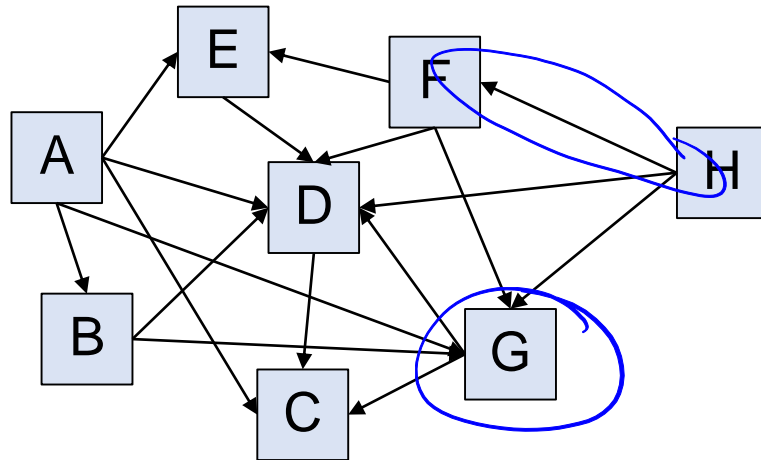
- Once computed, we can return the top hubs (highest $h(p)$ values) and the top authorities (highest $a(p)$ values) to the searcher.
- We can rewrite the equations in matrix notation. Let \mathbf{h} be the vector of hub values for all pages, and let \mathbf{a} be the vector of authority values. We can construct the adjacency matrix \mathbf{A} from the graph:

$$A_{i,j} = \begin{cases} 1 & \text{if } p_i \rightarrow p_j \\ 0 & \text{otherwise} \end{cases}$$

The rows of \mathbf{A} contain all outgoing links while the columns contain all incoming links. With this the computational scheme for the iteration becomes

$$\begin{aligned} \mathbf{h}^{(t+1)} &= \mathbf{A} \mathbf{a}^{(t)} \\ \mathbf{a}^{(t+1)} &= \mathbf{A}^T \mathbf{h}^{(t)} \end{aligned}$$

- Example: consider the following graph



Adjacency Matrix

	A	B	C	D	E	F	G	H
A		1	1	1	1		1	
B				1			1	
C								
D			1					
E				1				
F				1	1		1	
G			1	1				
H				1		1	1	

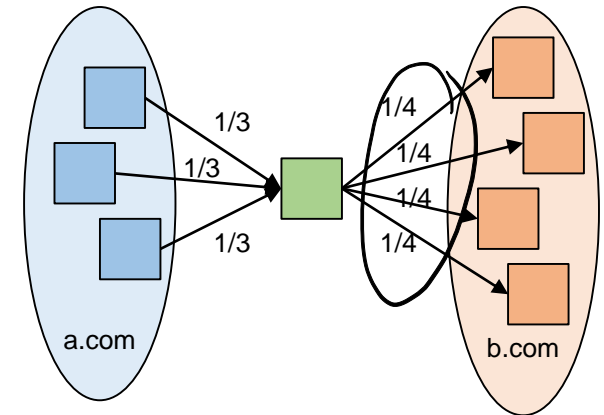
<i>h</i>	<i>a</i>
60%	0%
36%	18%
0%	29%
9%	69%
20%	31%
46%	12%
29%	54%
40%	0%

- We observe that A is the best hub. It links to the best authorities D, G, and E. E is a slightly better authority than C despite having only 2 (compared to 3) incoming links. But it is referenced by the good hubs A and F, while C is referenced by the good hub A, the average hub G and the bad hub D.
- Note that its not always clear whether a page is a hub or an authority. B for instance is a bad authority and an average hub. C is not in the top authorities and a bad hub. E is a top authority but also has some hub value.
- Finally, remember that we are only looking at the base set of nodes, that is, only at pages that are somewhat related to the topic. Hence, in contrast to PageRank, the hub and authority values of pages change with different queries / topics.

3.3.2 Extensions of HITS (Henzinger, 1998)

- The HITS algorithm suffers from three fundamental problems:
 1. If all pages in a domain reference the same external page, that page becomes too strong an authority. Similarly, if a page links to many different pages in the same domain, that page becomes too strong a hub.
 2. Automatically established links, e.g., advertisements or links to the provider/host/designer of a web site, provide the linked sites a too high authority value (even though they are off topic)
 3. Queries such a "jaguar car" tend favor the more frequent term (here "car") over the less frequent term. The idea of the query, however, was to distinguish from the animal.
- The first improvement addresses domains. Instead of every page having a single vote on the authority (hub) of an external page, the entire domain gets a single vote.
 - Assume that k pages q_i in a domain link a page p , then we weigh the hub values in the authority formula for page p with $aw(q_i, p) = \frac{1}{k}$.
 - Similarly, assume that a page p links to l pages q_i in the same domain, then we weigh the authority values in the hub formula for page p with $hw(p, q_i) = \frac{1}{l}$.
 - With these weights, we adjust the iteration of HITS as follows:

$$a^{(t+1)}(p) = \sum_{q \rightarrow p} aw(q, p) \cdot h^{(t)}(q) \quad h^{(t+1)}(p) = \sum_{p \rightarrow q} hw(p, q) \cdot a^{(t)}(q)$$



- The second improvement focuses on the topic and applies penalties for pages that are not following the topic. This helps to sort out advertisements. To push less frequent terms, and $tf * idf$ scheme is chosen similar to the methods in vector space retrieval.
 - We construct a reference document C from all documents in the root set (e.g., taking from each document the terms with highest $tf * idf$ values)
 - Compute a similarity value $s(p)$ for page p using the $tf * idf$ vectors of p and the reference document C , i.e., $s(p) = \frac{c^T p}{\|c\| \cdot \|p\|}$
 - For a given threshold t , eliminate all pages p with $s(p) < t$ from the base set. To get a good threshold, we can use the median of all $s(p)$ values, i.e., eliminate 50% from the base set.
 - Use the similarity values $s(p)$ to adjust how much authority and hub value is passed to a page. We adjust the iteration of the HITS algorithm as follows:

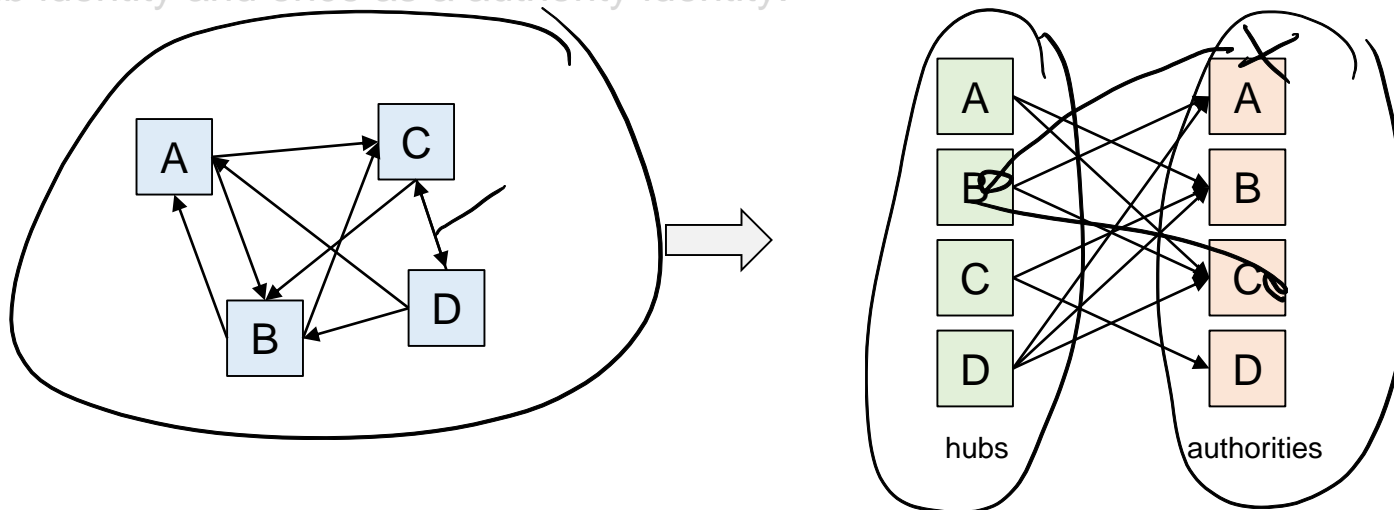
$$a^{(t+1)}(p) = \sum_{q \rightarrow p} aw(q, p) \cdot s(q) \cdot h^{(t)}(q)$$

$$h^{(t+1)}(p) = \sum_{p \rightarrow q} hw(p, q) \cdot s(q) \cdot a^{(t)}(q)$$

- This extension has resulted in a 45% improvement over the original HITS algorithm.

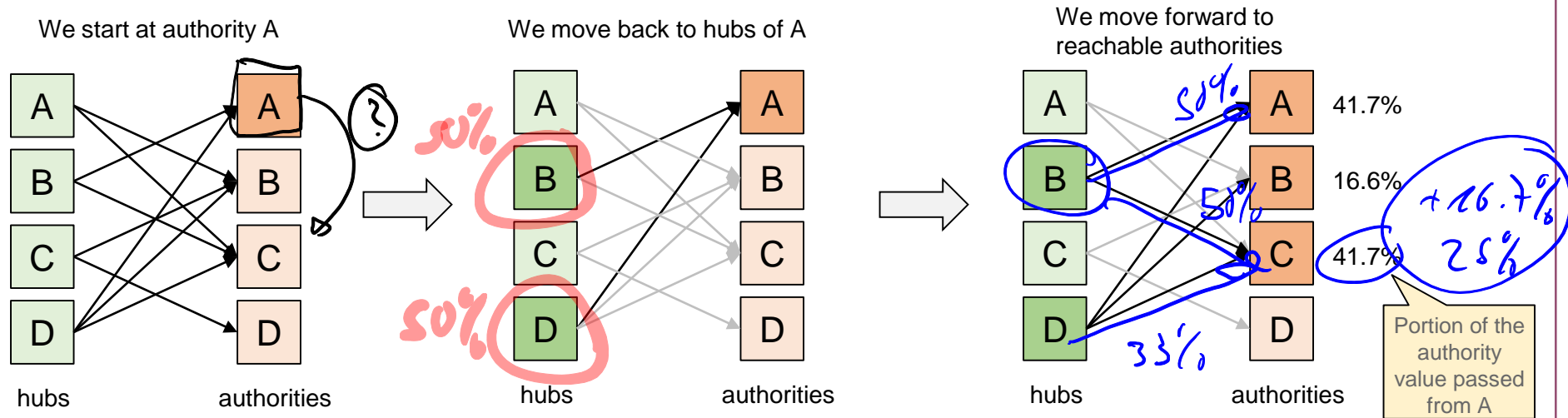
3.3.3 SALSA Algorithm

- The Stochastic Approach for Link Structure Analysis – SALSA is a further extension of the HITS algorithm. Similar to PageRank, it considers the transitions from one page to the other and models it with a Markov chain. However, it only considers two steps in the network, and not an infinite walk across the entire web. Similar to HITS, it only considers a base set of pages obtained with the same approach as with HITS and given a query / topic.
- SALSA considers the graph of the base set as a bipartite graph with pages having a double identity, once as a hub identity and once as an authority identity.

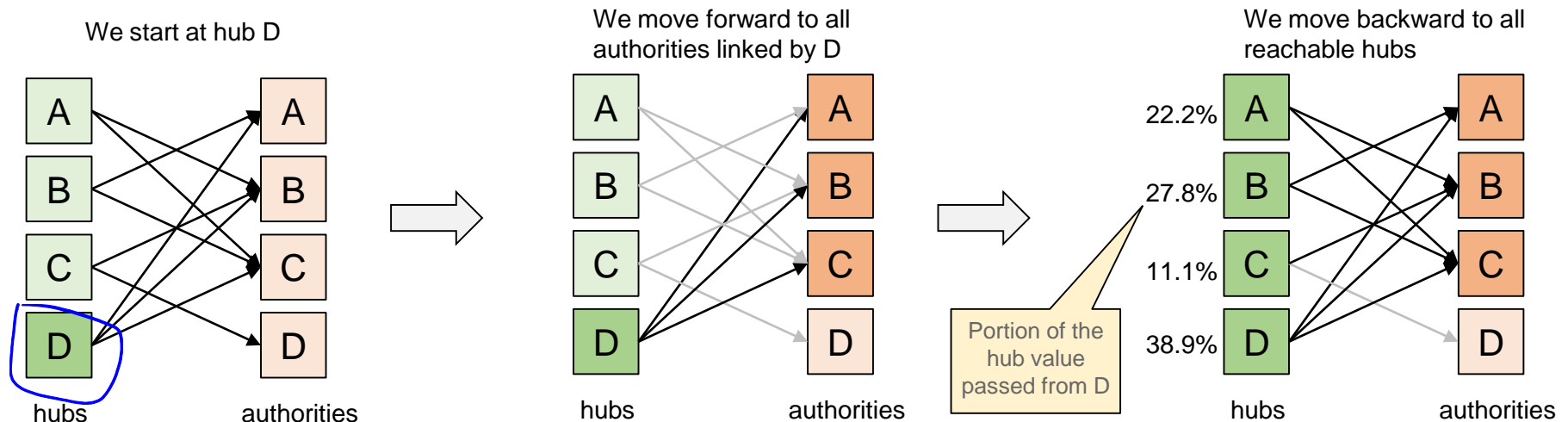


- To compute authority values, the algorithm is performing a random walk with two steps. Starting from a page p , it goes backward to all hubs that link to p and then walks forward to all pages reachable from these hubs. To determine how much authority value is passed from page p to a such reachable page q , we consider a random walk with two steps starting at p and use the probability of arriving at q as the fraction of authority passed from p to q .
- In contrast to HITS, authority values only depend on the authority value of other reachable authorities but not on the hub values.

- Example: consider the example before. We want to compute how much of the authority value of A is passed to C. We now consider the steps of the random walk starting at A:
 - We first walk backwards to all hubs that link to A: there are two hubs B and D each with a 50% chance for the random walk to select.
 - Walking forward: 1) From B, there are two links to A and C, again each path with a 50% chance to be taken. We note a first path from A to C with a 25% chance to be taken. 2) From D, there are three links to A, B and C, each path with a 33.3% chance to be taken. We note a second path from A to C with a 16.7% chance to be taken
 - Summing up, the two paths yield a 41.7% chance to get from A to C. This is the portion of the authority value of A passed to the authority value of C.



- Similarly, we can compute hub values. But this time, the random walk is first forward to all authorities linked by the starting hub, and then backwards to all hubs linking these authorities. The probability of reaching hub q from a hub p determines how much hub value is passed from p to q .
- Example: consider the same example as before. We want to compute how much of the hub value of D is passed to B. We now consider the steps of the random walk starting at D:
 - We first walk forwards to all authorities linked by D: there are three authorities A, B and C each with a 33% chance for the random walk to select.
 - Walking backwards: 1) The two hubs B and D link to A, each hub selected with 50% probability. We note a first path from D to B with a 16.7% chance to be taken. 2) The three hubs A, C and D link to B, each hub selected with 33% probability. There is no path to B. 3) The three hubs A, B and D link to C, each hub selected with 33% probability. We note a second path from D to B with a 11.1% chance to be taken.
 - Summing up, the two paths yield a 27.8% chance to get from D to B. This is the portion of the hub value of D passed to the hub value of B.



- More formally, we can compute hub and authority values as follows. Let \mathbf{A} be the authority-matrix and \mathbf{H} be the hub-matrix. Further, let $L_{in}(p)$ denote the number of incoming links to page p , and $L_{out}(p)$ denote the number of outgoing links of page p . We can determine the matrix elements with the two steps as described before as follows:

$$A_{j,i} = \sum_{q: q \rightarrow p_i \wedge q \rightarrow p_j} \frac{1}{L_{in}(p_i)} \cdot \frac{1}{L_{out}(q)}$$

$$H_{j,i} = \sum_{q: p_i \rightarrow q \wedge p_j \rightarrow q} \frac{1}{L_{out}(p_i)} \cdot \frac{1}{L_{in}(q)}$$

- We now can compute the hub and authority value using an iterative approach. Again, \mathbb{P} denotes the set of all pages in the base set with $N = |\mathbb{P}|$ being the number of pages.

1. Initialization: $h_i^{(0)} = a_i^{(0)} = 1/N \quad \forall i: 1 \leq i \leq N$
2. Iteration:
 - $\mathbf{a}^{(t+1)} = \mathbf{A}\mathbf{a}^{(t)}$
 - $\mathbf{h}^{(t+1)} = \mathbf{H}\mathbf{h}^{(t)}$
 - stop if $\|\mathbf{a}^{(t+1)} - \mathbf{a}^{(t)}\| + \|\mathbf{h}^{(t+1)} - \mathbf{h}^{(t)}\| < \epsilon$

- A variant of the SALSA algorithm is used at Twitter to recommend “whom to follow”. The twitter example is a very good fit as the graph is usually uni-directional (you follow someone but that person is not necessarily following you).

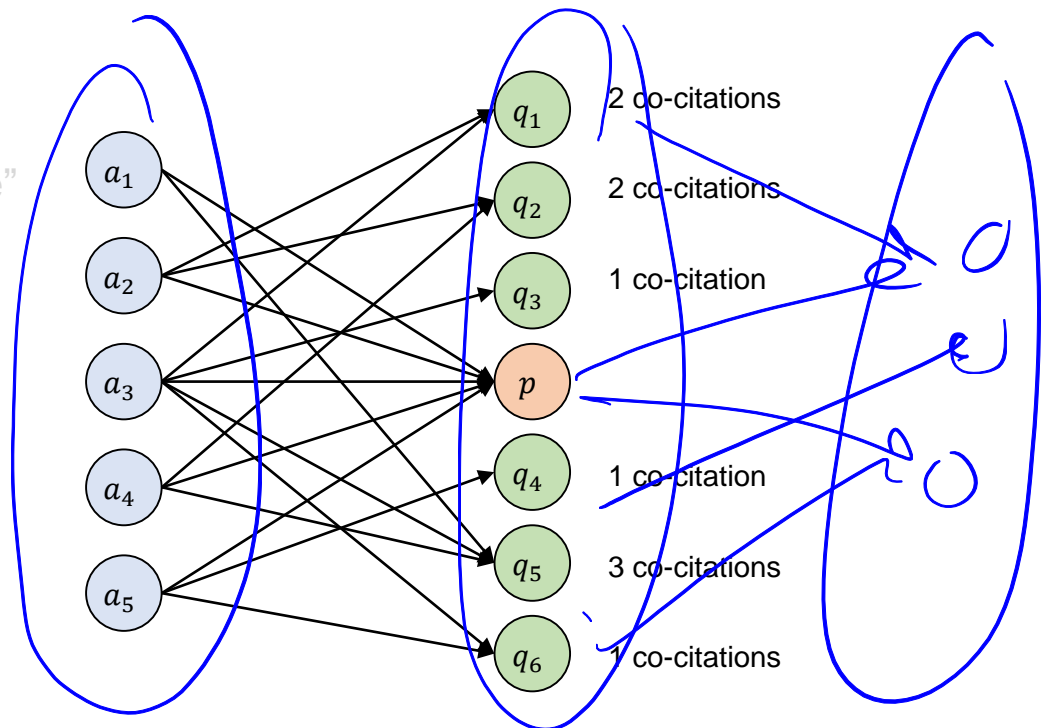
3.3.4 Co-citations and Similar Pages

- The basic idea of **Alexas „What's Related“** was to identify similar or related documents for a given document. As an example, if you are on the homepage of a car manufacturer (e.g., Ford Motor), a related page would be the one of another car manufacturer (e.g., Honda, GM, VW, Nissan). How can we compute such related pages from the web graph?
 - Surf History/Bookmarks Analysis: users often browse and keep pages on similar topics together. If you want to buy a car, you check several similar pages in succession
 - Co-citations: if two pages are often linked together by pages, we can assume some relationship
 - Deduce relationships from link structure and an implicit similarity score definition (similar pages are linked by similar pages)
- Alexa Toolbars observe surfers, record the history, and performs a static analysis on the surf behavior of users to compute various things such as the Alexa Ranking and suggestions for similar sites. The analysis follows typical data mining approaches for affinity analysis. It is not taking the link structure of the web into account.
- In the following, we look at the other two methods in more details:

- Co-Citations consider only the web graph and count how often two pages are linked together. The first publication by Dean and Henzinger in 1999 suggested the following simple algorithm. We start with a page p and are looking for related pages q_i :

- Determine at most k parent pages a_j of starting page p
- Extract for each parent page a_j at most l links to pages q_i that are in the proximity of the link to p
- Count how often a page q_i is obtained by step 2
- If we found less than 15 pages q_i with at least 2 co-citations with p then reduce URL of p and start again.
- Related pages q_i to p are the ones with most co-citations

- Note that not all links are extracted from parent pages a_i but only the ones that appear close to the starting page p . “Close” means the l links which are nearest to the link to p in the HTML file.
- The figure on the right hand shows a simple example with a starting page p , its parent pages a_1, \dots, a_5 and their linked pages q_1, \dots, q_6 . In this example, we find q_5 as the most co-cited page to p .



- Another co-citation method by Dean and Henzinger was published in 1999: the **companion algorithm**. It is a more complex variant of co-citation using similar techniques as with HITS. We start again with a page p and look for related pages q_i :

1. Build a neighborhood graph around page p as follows:
 - add starting page p to graph
 - add at most k parent pages a_i that link to p , and for each page a_i , add at most l links to child pages (around the link to p)
 - add at most m child pages c_j linked by p , and for each page c_j , add at most n parent pages that have a link to c_j
 - add edges between nodes based on the links in the web
2. Merge duplicates and near-duplicates
 - Two documents are near-duplicates if they contain more than 10 links and 95% of their links are the same (occur in both documents)
3. Assign weights to edges in graph based on domain linked
 - Assume that k pages q_i in a domain link a page p , then we weight the edges with $aw(q_i, p) = \frac{1}{k}$
 - Assume that a page p links to k pages q_i in a domain, then we weight edges with $hw(p, q_i) = \frac{1}{l}$
4. Compute hub and authority values for all nodes in the graph with the following iteration

$$a^{(t+1)}(p) = \sum_{q \rightarrow p} aw(q, p) \cdot h^{(t)}(q) \qquad h^{(t+1)}(p) = \sum_{p \rightarrow q} hw(p, q) \cdot a^{(t)}(q)$$
5. The pages q_i with the highest authority values are the related pages to p .

- **SimRank** is a method that defines a similarity score $\sigma(p, q)$ between two pages p and q in an implicit way: p and q are similar if they are linked by similar pages. Also, a page p is maximal similar with itself, i.e., $\sigma(p, p) = 1$. Given a “neighborhood” graph around p (use any method to construct such a neighborhood), we select those pages q_i with highest scores $\sigma(p, q_i)$.
 - Let $L_{in}(p)$ denote the number of incoming links to page p . Similarity is defined as follows:

$$\sigma(p, q) = \begin{cases} 1 & \text{if } p = q \\ \frac{C}{L_{in}(p) \cdot L_{in}(q)} \cdot \sum_{a \rightarrow p} \sum_{b \rightarrow q} \sigma(a, b) & \text{otherwise} \end{cases}$$

with C a decay factor, for instance $C = 0.8$.

- We can compute the similarity values with a simple iterative approach for all pages $p \in \mathbb{P}$:

1. Initialization: $\sigma^{(0)}(p, q) = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases} \quad \forall p, q \in \mathbb{P}$
2. Iteration:
 - Update: $\sigma^{(t+1)}(p, q) = \begin{cases} 1 & \text{if } p = q \\ \frac{C}{L_{in}(p) \cdot L_{in}(q)} \cdot \sum_{a \rightarrow p} \sum_{b \rightarrow q} \sigma^{(t)}(a, b) & \text{otherwise} \end{cases}$
 - Stop if $\sum_{p, q} |\sigma^{(t+1)}(p, q) - \sigma^{(t)}(p, q)| < \epsilon$
3. Return pages q_i with highest similarity $\sigma(p, q_i)$

3.4 Literature and Links

- Google: <http://www.google.com>
- Google Research: <https://research.google.com/pubs/papers.html>
- Brin, S.; Page, L. (1998). ["The anatomy of a large-scale hypertextual Web search engine"](#) (PDF). Computer Networks and ISDN Systems. **30**: 107–117.
- Page, Lawrence; Brin, Sergey; Motwani, Rajeev; Winograd, Terry (1999). ["The PageRank citation ranking: Bringing order to the Web"](#)., published as a technical report on January 29, 1998 [PDF](#)
- Luiz Andre Barroso, Jeffrey Dean, Urs Hölzle, [Web Search for a Planet: The Google Cluster Architecture](#), April 2003 (IEEE Computer Society).
- Ghemawat, S.; Gobioff, H.; Leung, S. T. (2003). "The Google file system". [Proceedings of the nineteenth ACM Symposium on Operating Systems Principles - SOSP '03](#) (PDF). p. 29. [CiteSeerX 10.1.1.125.789](#)
- Lempel, R.; Moran S. (April 2001). ["SALSA: The Stochastic Approach for Link-Structure Analysis"](#) (PDF). ACM Transactions on Information Systems. **19** (2): 131–160.
- G. Jeh and J. Widom. SimRank: A Measure of Structural-Context Similarity. In [KDD'02](#): Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 538-543. [ACM Press](#), 2002. [PDF](#)
- J. Dean, M. Henzinger, Finding related pages in the World Wide Web, Computer networks 31 (11), 1467-1479. [PDF](#)
- J. Kleinberg. ["Hubs, Authorities, and Communities"](#). ACM Computing Surveys 31(4), 1999.