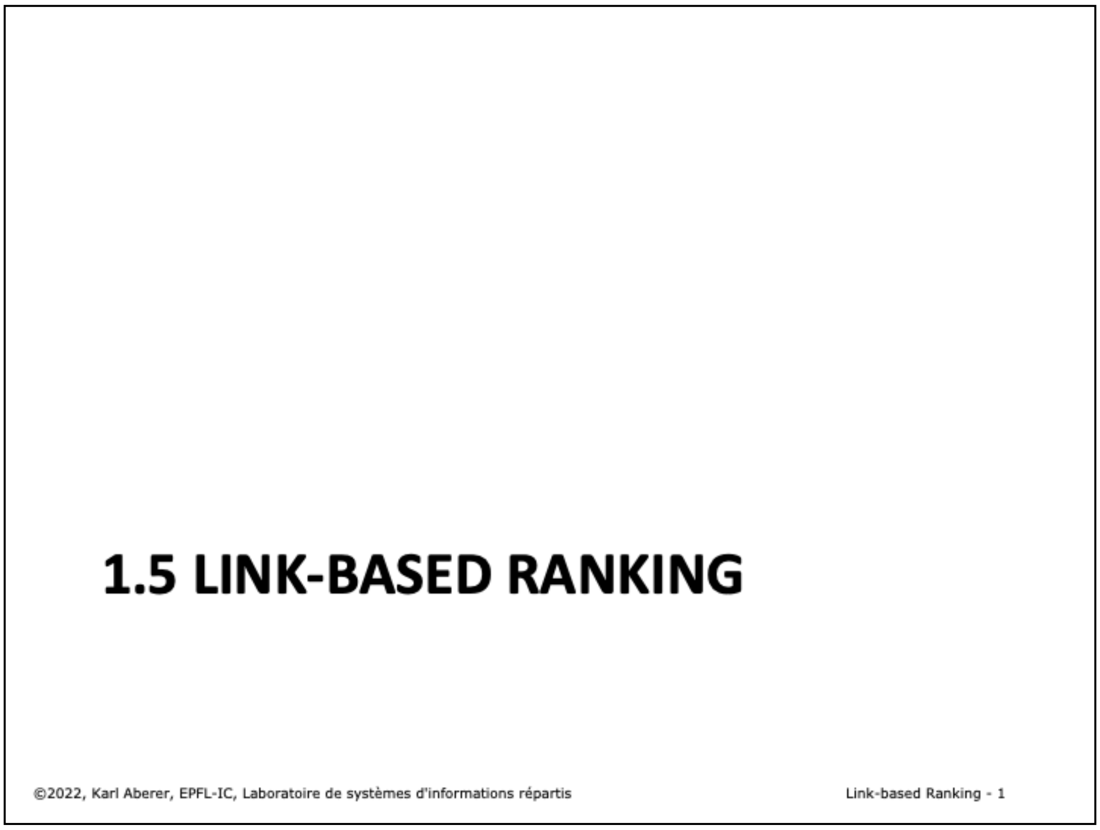
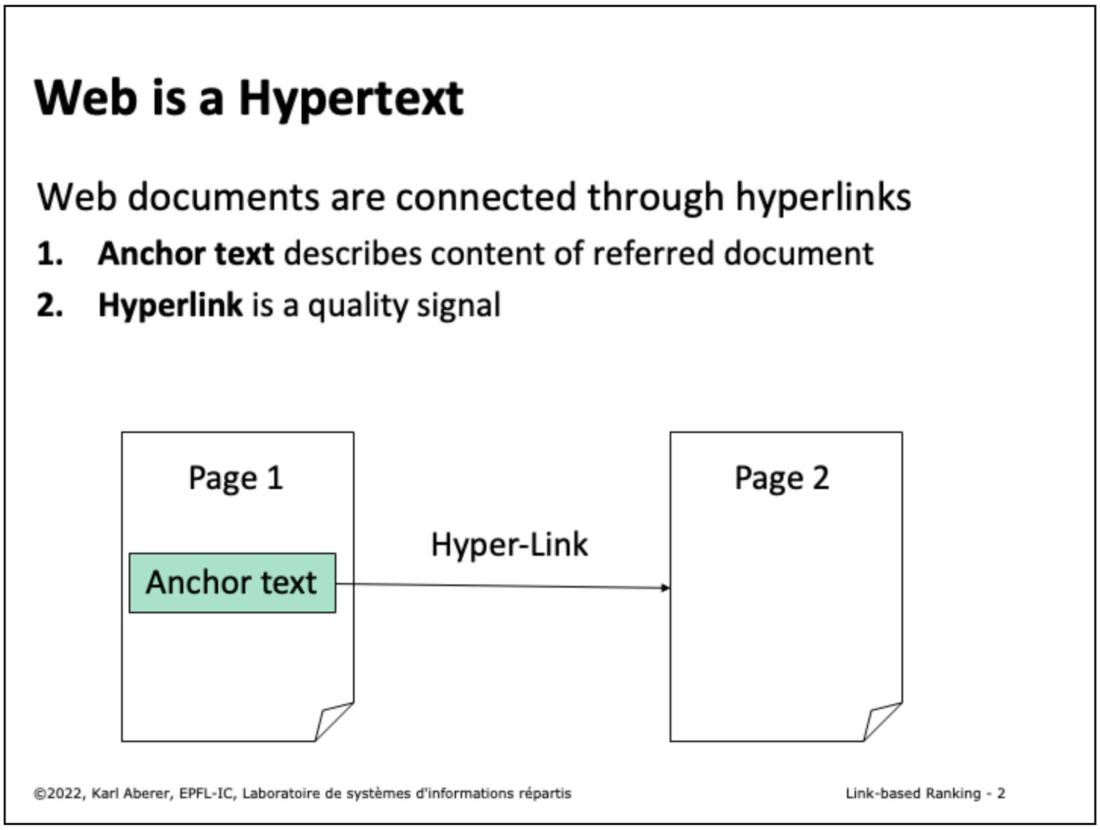


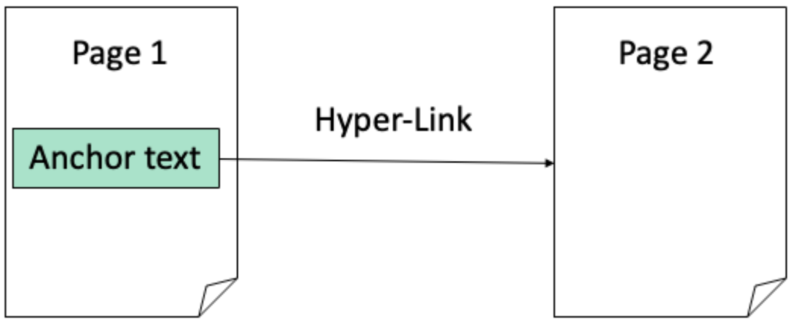
1.5 LINK-BASED RANKING



Web is a Hypertext

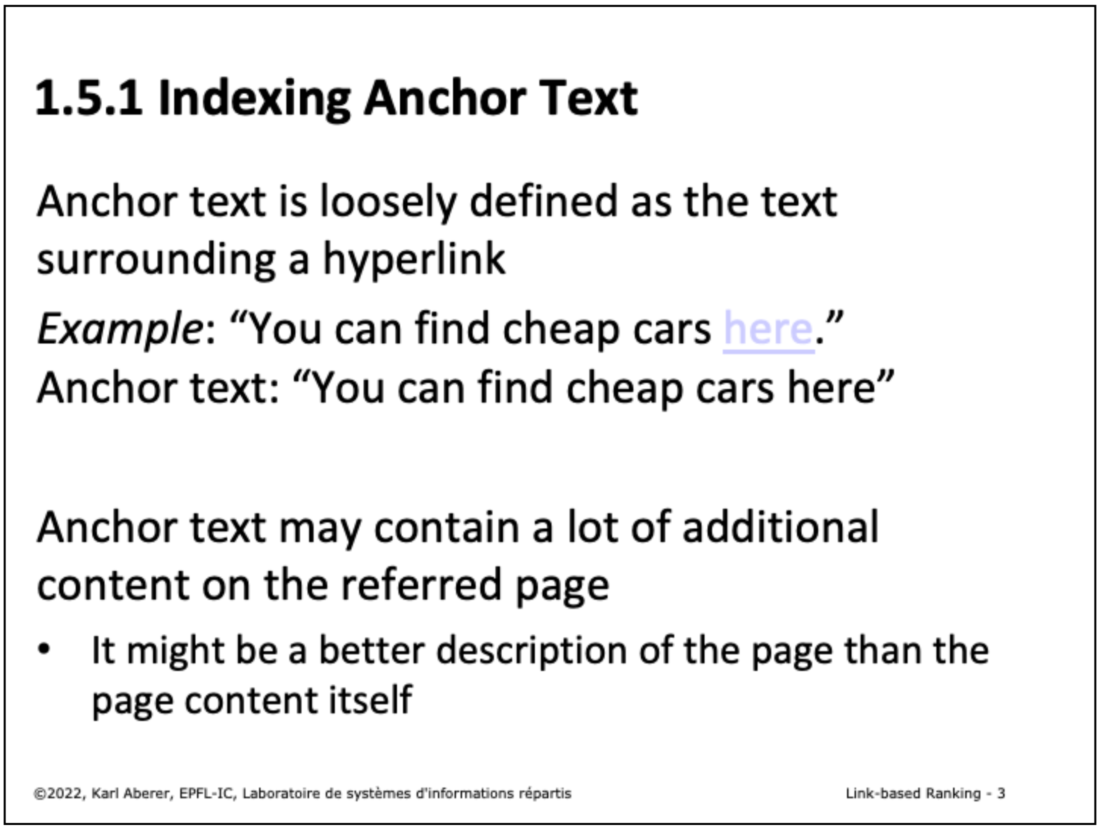
Web documents are connected through hyperlinks

1. **Anchor text** describes content of referred document
2. **Hyperlink** is a quality signal



In addition to the textual content, Web documents contain also hyperlinks. A hyperlink can be exploited for information retrieval in two ways:

1. The link is embedded into some text that typically contains relevant information on the content of the document the link is pointing to. Thus, this text can complement the content of the referred document.
2. The link can also be considered as an endorsement of the referenced document by the author of the referring document. Thus, the link can be used as a signal for quality and importance of the referred document.



1.5.1 Indexing Anchor Text

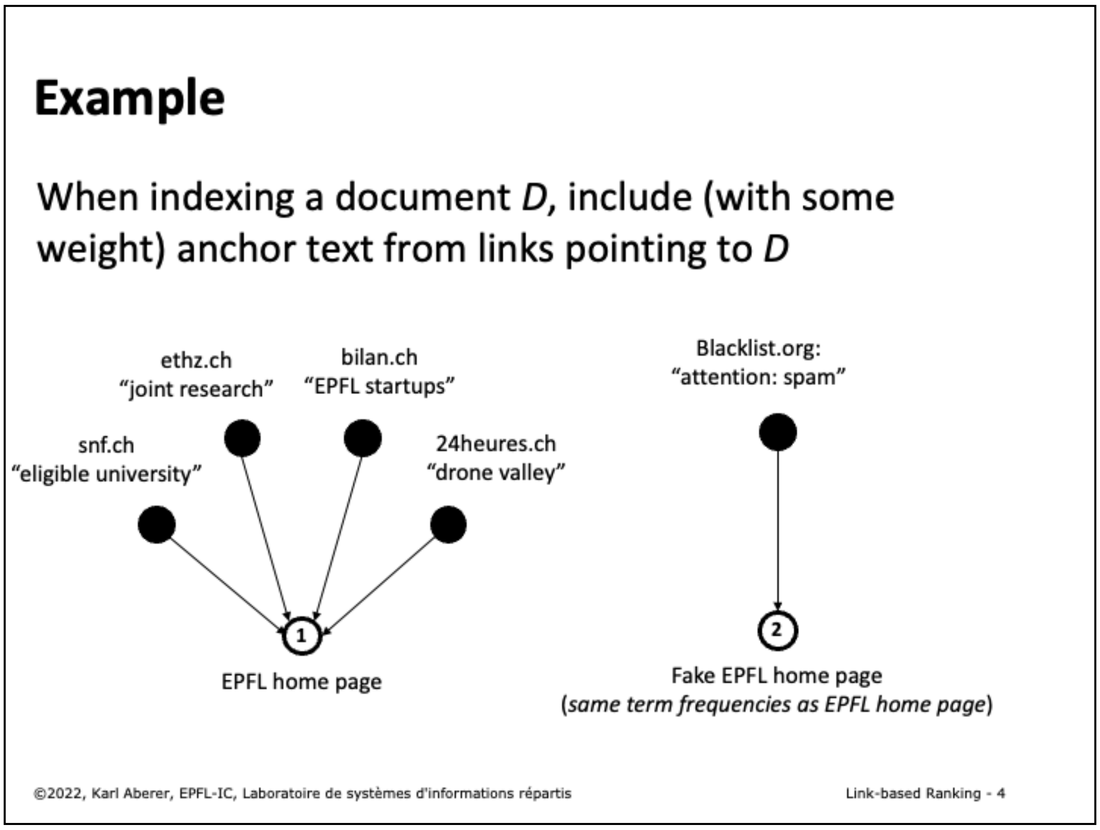
Anchor text is loosely defined as the text surrounding a hyperlink

Example: "You can find cheap cars here."
Anchor text: "You can find cheap cars here"

Anchor text may contain a lot of additional content on the referred page

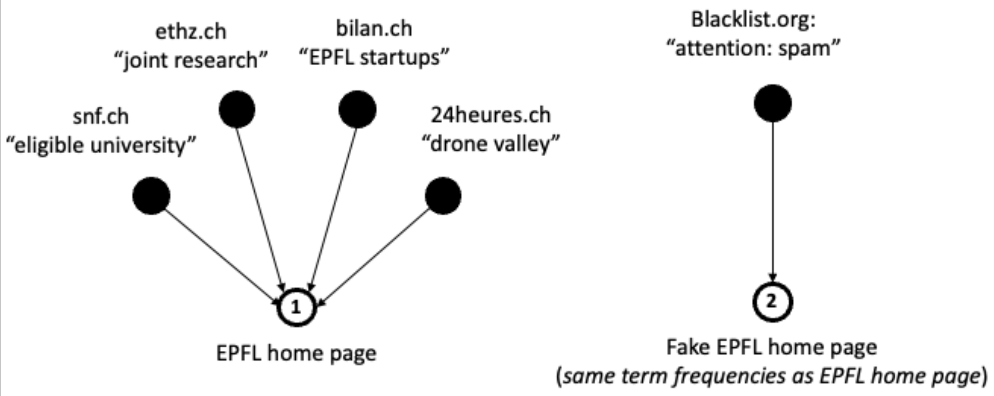
- It might be a better description of the page than the page content itself

Anchor text corresponds to the text that is surrounding the link, and not only the text contained as part of the link tag (in the example, the text in the link tag would simply be "here".) The anchor text can contain valuable information on the referred page and thus be helpful in retrieval.



Example

When indexing a document *D*, include (with some weight) anchor text from links pointing to *D*



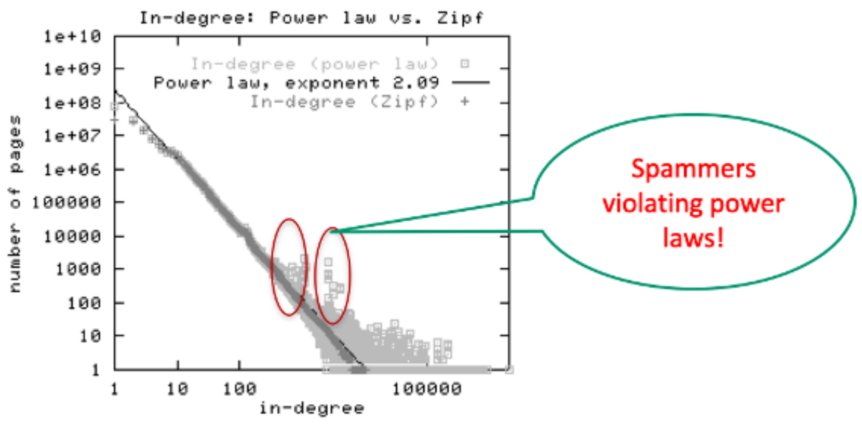
This example illustrates the use of anchor text in retrieval. Often, a home page is very visual and contains often little relevant text content. If we consider a home page, such as the EPFL home page, we probably find many pages pointing to the EPFL home page that very well characterize EPFL, such as pages mentioning topics related to research and technology transfer.

Assume that a malicious Internet user would create a fake EPFL home page. Then chances that such a page is referred by reputed organizations, such as SNF, is very low. On the other hand, pages listing spam pages might point to such a page and reveal its true character. These pages would probably also mention terminology related to spam pages or blacklists, and such text can give indications about the true character of the spam page.

In addition, links to the EPFL home page indicate a higher importance of the page, as compared to other less referenced pages, such as pages containing the EPFL regulations.

Indexing Anchor Text

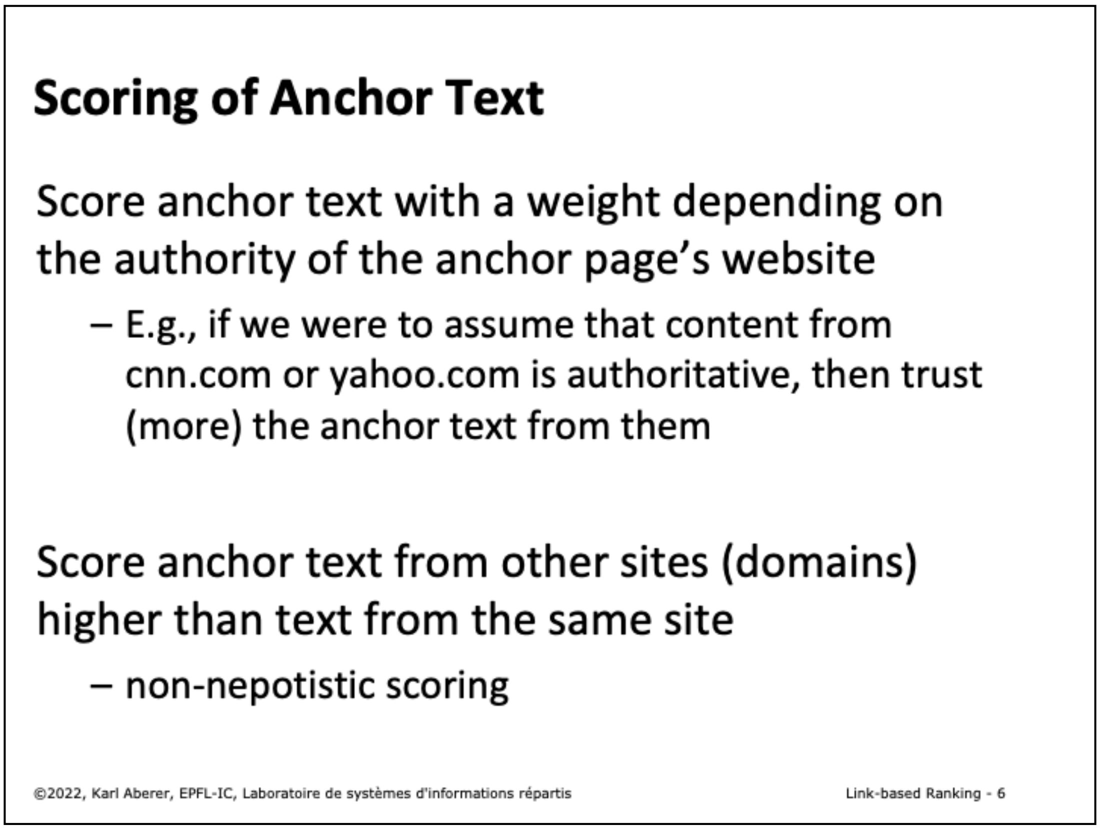
Can sometimes lead to unexpected effects, e.g., easily spammable



One of the risks of including anchor text is that it makes pages spammable. Malicious users could create spam pages that point to web pages and try to relate it to contents that serve their interests (e.g., higher the quality of preferred pages by adding links, lower the quality of the undesired page by attaching negative anchor text). That this is. Real phenomenon can be inferred from statistics on the in-degree distribution of Web pages that has been produced.

The figure shows a standard log-log representation of the in-degree vs. the frequency of pages. Normally this relationship should follow a power-law, which shows in a log-log representation as a linear relationship. In real Web data, we see that this power law is violated, and that certain levels of in-degrees are over-represented. This can be attributed to link spamming, which does create moderate numbers of additional links on Web pages.

This is of course only one example of spamming techniques, and Web search engines are in a continuous "battle" against this and other forms of spam.



Scoring of Anchor Text

Score anchor text with a weight depending on the authority of the anchor page's website

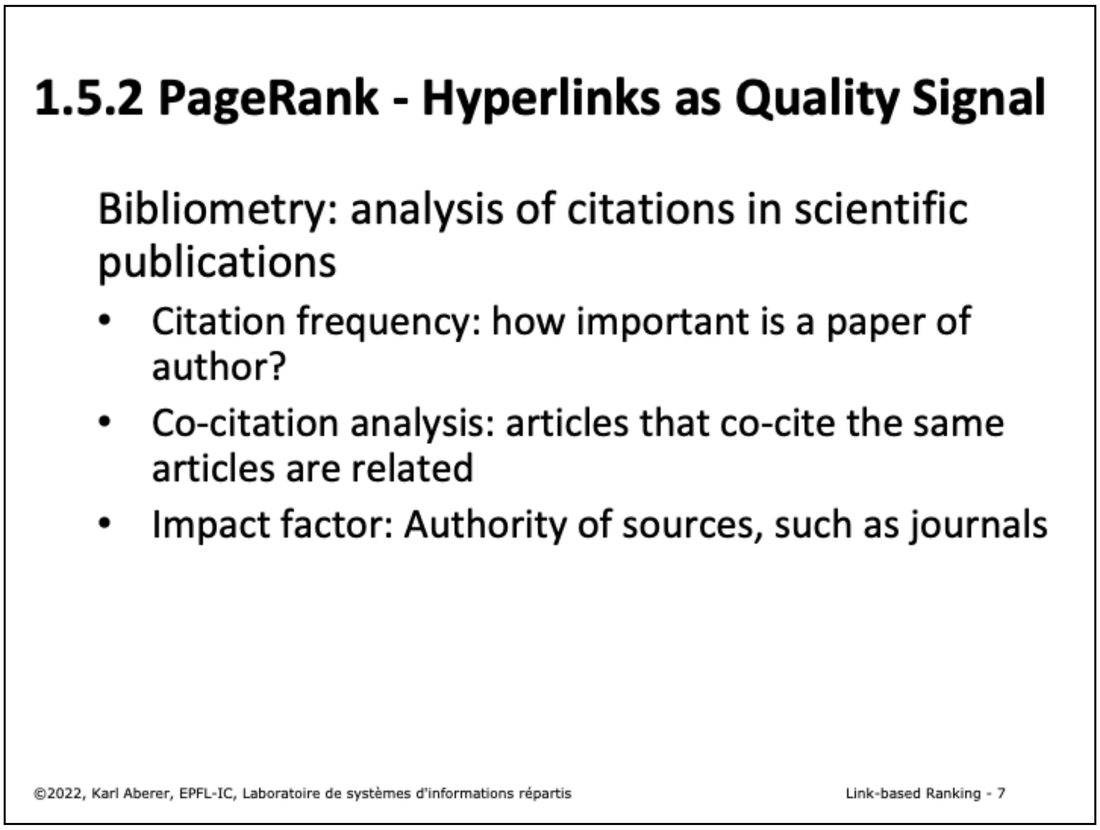
- E.g., if we were to assume that content from cnn.com or yahoo.com is authoritative, then trust (more) the anchor text from them

Score anchor text from other sites (domains) higher than text from the same site

- non-nepotistic scoring

In order to fight link spamming, when considering anchor text, the text from pages with poor reputation can be given lower weight. We will later introduce methods of how to rank pages based in the hyperlink graph, which is one method to evaluate the reputation of a page.

In order to avoid self-promotion, another method to fight link spamming is to give lower weights to links within the same site (nepotism = promoting your own family members).



1.5.2 PageRank - Hyperlinks as Quality Signal

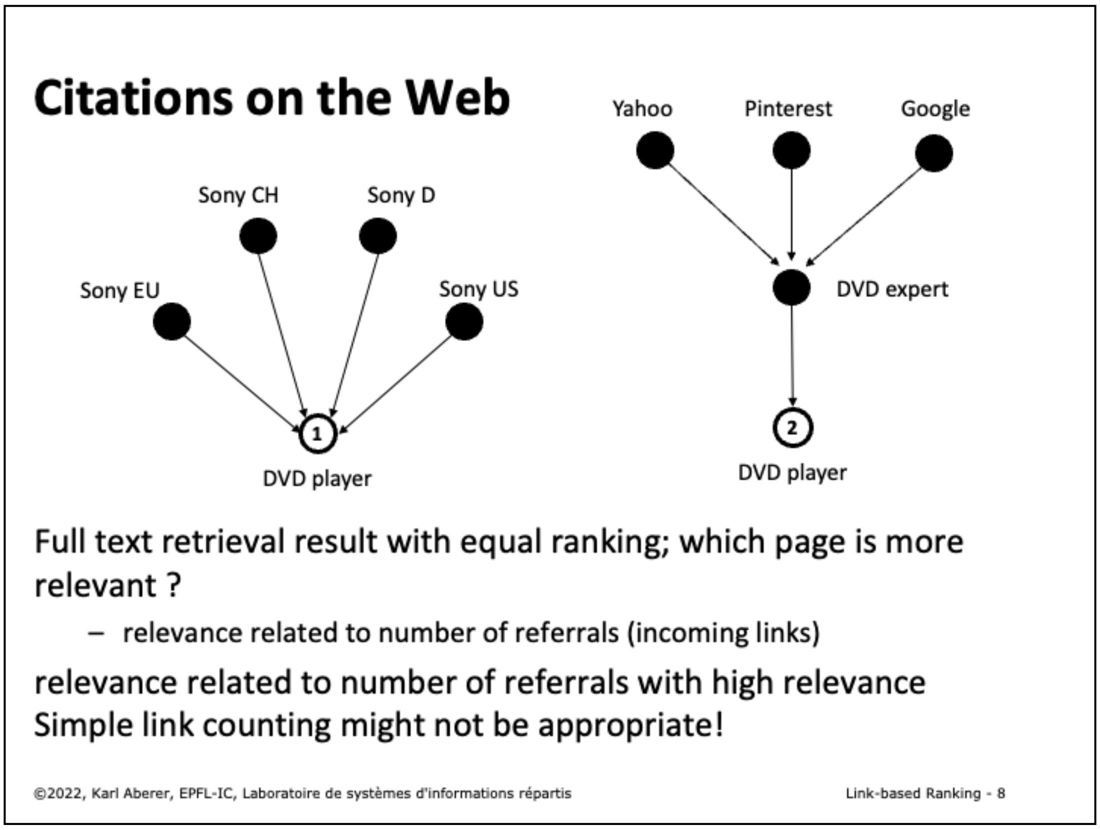
Bibliometry: analysis of citations in scientific publications

- Citation frequency: how important is a paper of author?
- Co-citation analysis: articles that co-cite the same articles are related
- Impact factor: Authority of sources, such as journals

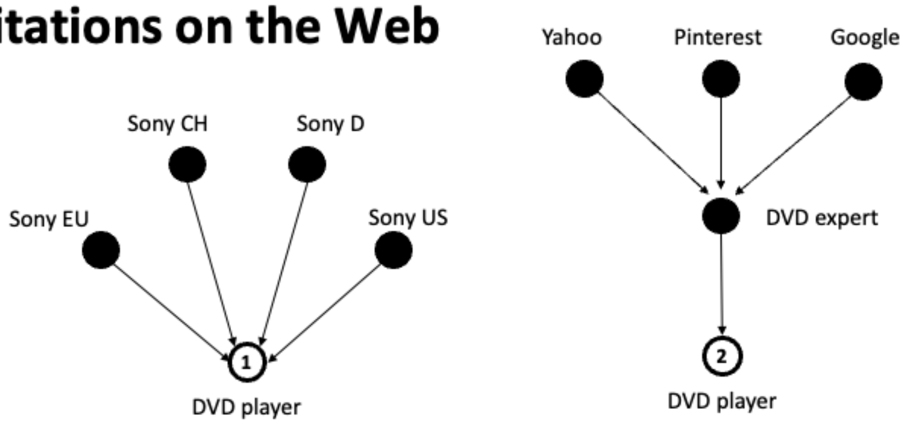
The use of links in order to evaluate the quality of information sources has a long tradition, specifically in science. The discipline of bibliometry is fully devoted to the problem of evaluating the quality of research through citation analysis. Different ideas can be exploited to that end:

- The frequency of citations to a paper, indicating how popular or visible it is
- Co-citation analysis in order to identify researchers working in related disciplines
- Analysis of the authority of sources of scientific publications, e.g., journals, publishers, conferences. This measure can then in turn be used to weight the relevance of publications.

All these ideas can also be exploited for any other document collections that have references, in particular, for Web document collections with hyperlinks.



Citations on the Web



Full text retrieval result with equal ranking; which page is more relevant ?

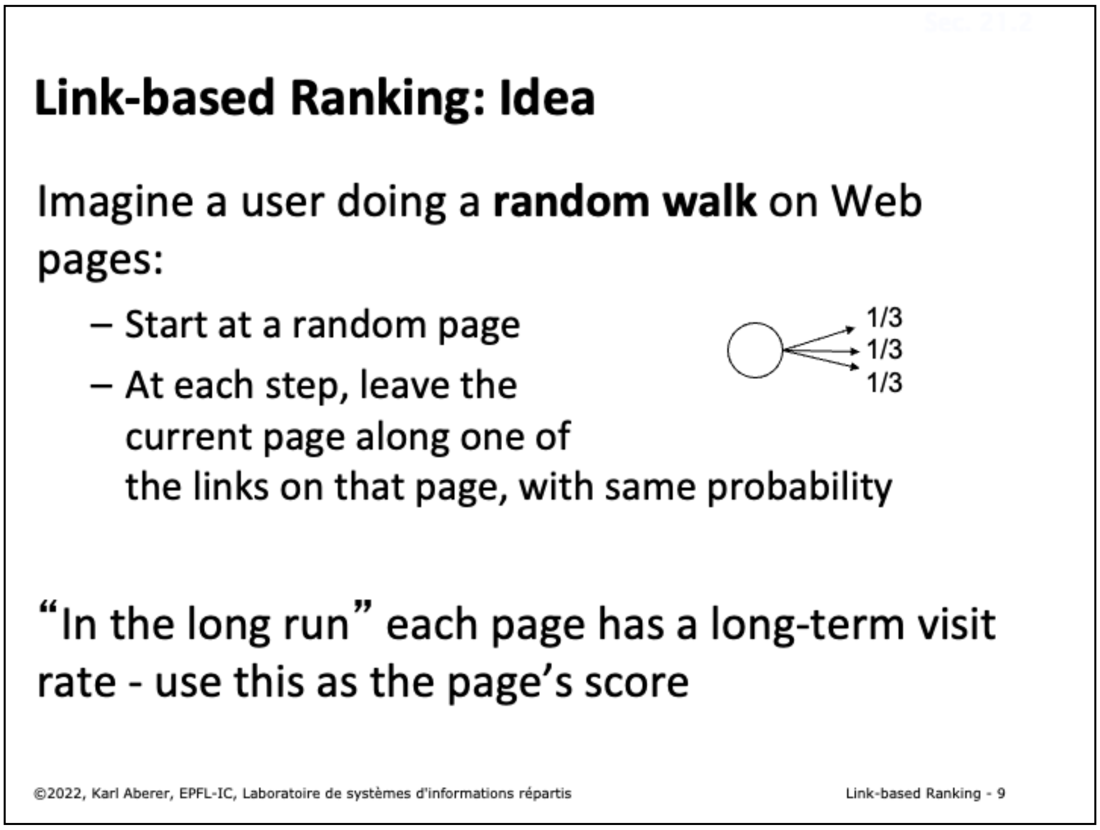
– relevance related to number of referrals (incoming links)

relevance related to number of referrals with high relevance
Simple link counting might not be appropriate!

When retrieving documents from the Web, the link structure bears important information on the relevance of documents. A document that is referred more often by other documents through hyperlinks, is likely to be of higher interest and therefore relevance. Therefore, a possibility to rank documents is considering the number of incoming links. Considering the number of incoming links allows to distinguish documents that otherwise would be ranked similarly when relying on text-based relevance ranking.

However, when doing this, also the importance of the link sources can be different. Therefore, not only counting then number of incoming links, but also weighing the links by the relevance of documents that contain these links can help to better assess the quality of a document. The same reasoning of course again applies then for evaluating the relevance of documents pointing to the source of the link and so forth.

Different to scientific publishing, in the Web references are not reliable and therefore simple link counting might not be appropriate. Since 1998 when search engines started to consider links for ranking the phenomenon of link spamming started. Link farms are groups of websites that are heavily linked to one another to boost their ranking.


Sec. 21.2

Link-based Ranking: Idea

Imagine a user doing a **random walk** on Web pages:

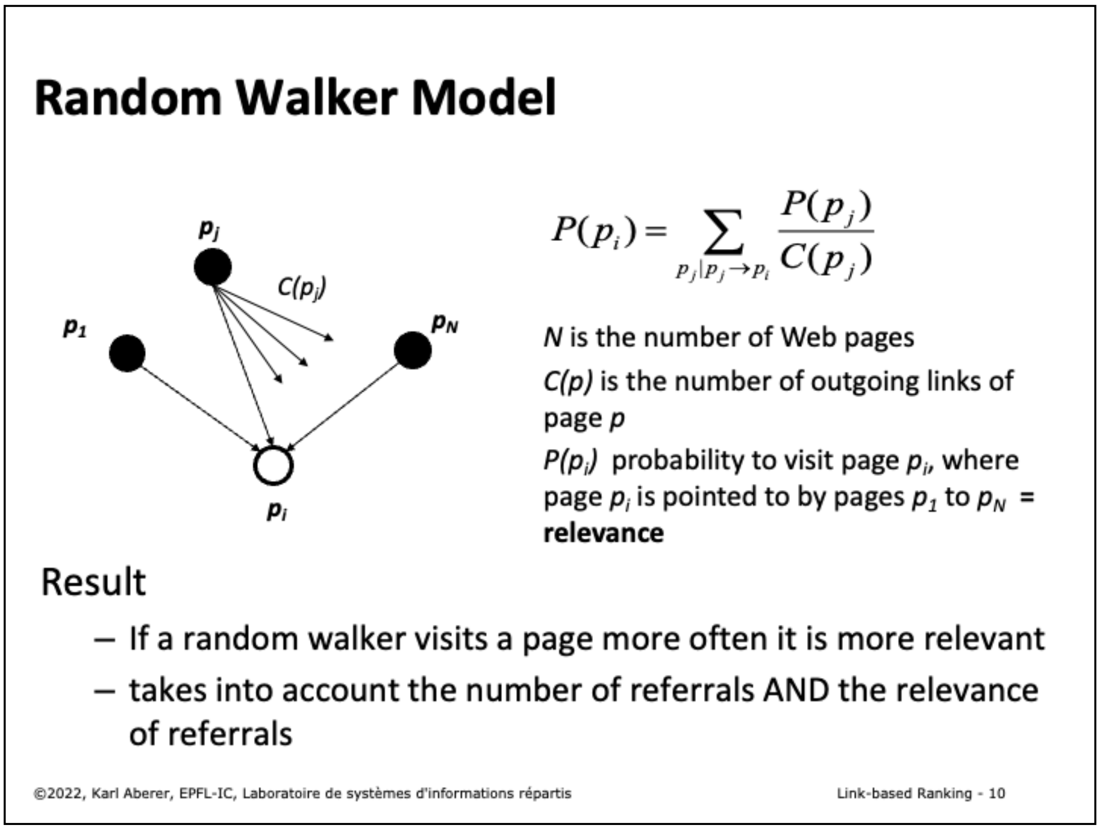
- Start at a random page
- At each step, leave the current page along one of the links on that page, with same probability

"In the long run" each page has a long-term visit rate - use this as the page's score

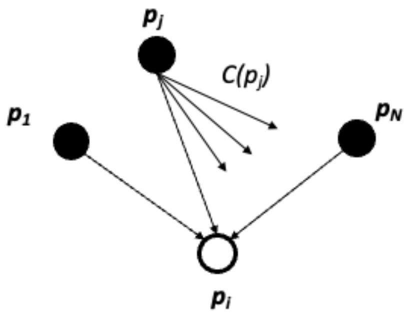
We introduce now an approach for link-based scoring that considers not only the absolute count of links, but also the quality of the link source. The basic idea is to consider a random walker that visits Web pages following the hyperlinks. At each page the random walker would select randomly among the hyperlinks of the page with uniform probability and move to the next page. When the random walker runs for a long time, it will visit every page with a given probability, which we can consider as a score for ranking the page. This score can be used to control the impact of the outgoing links of a page on the ranking of other pages.

One of the consequences of this model would be that pages that have few in-links, would be relatively infrequently visited. Since link farms and spam pages usually have not many links pointing to them, the expectation is that this approach could reduce their impact on ranking.

On the other hand, popular pages with many incoming links will have a higher impact on ranking, as they have a higher score.



Random Walker Model



$$P(p_i) = \sum_{p_j | p_j \to p_i} \frac{P(p_j)}{C(p_j)}$$

N is the number of Web pages

C(p) is the number of outgoing links of page *p*

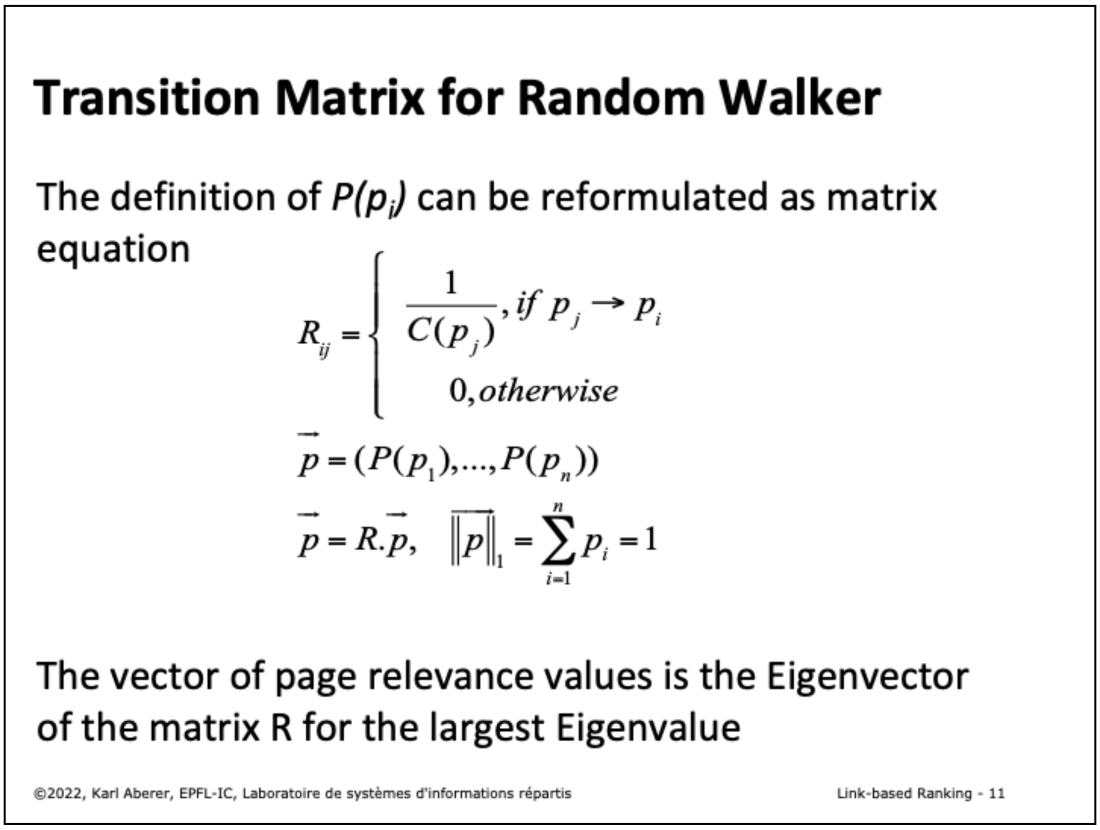
P(p_i) probability to visit page p_i, where page p_i is pointed to by pages p_1 to p_N = **relevance**

Result

- If a random walker visits a page more often it is more relevant
- takes into account the number of referrals AND the relevance of referrals

We provide a formal description of the random walker model. The model is a Markov chain, a discrete-time stochastic process in which at each time-step a random choice is made.

We assign to each page a visiting probability *P(p_i).* Then the probability that a page p_i is visited depends on the probabilities of the pages with a hyperlink to this page to be visited. For the source pages of the hyperlink the visiting probability is evenly split among all outgoing links. This formulation of the process results in a recursive equation, of which the solution is the steady-state of the process.



Transition Matrix for Random Walker

The definition of *P(p_i)* can be reformulated as matrix equation

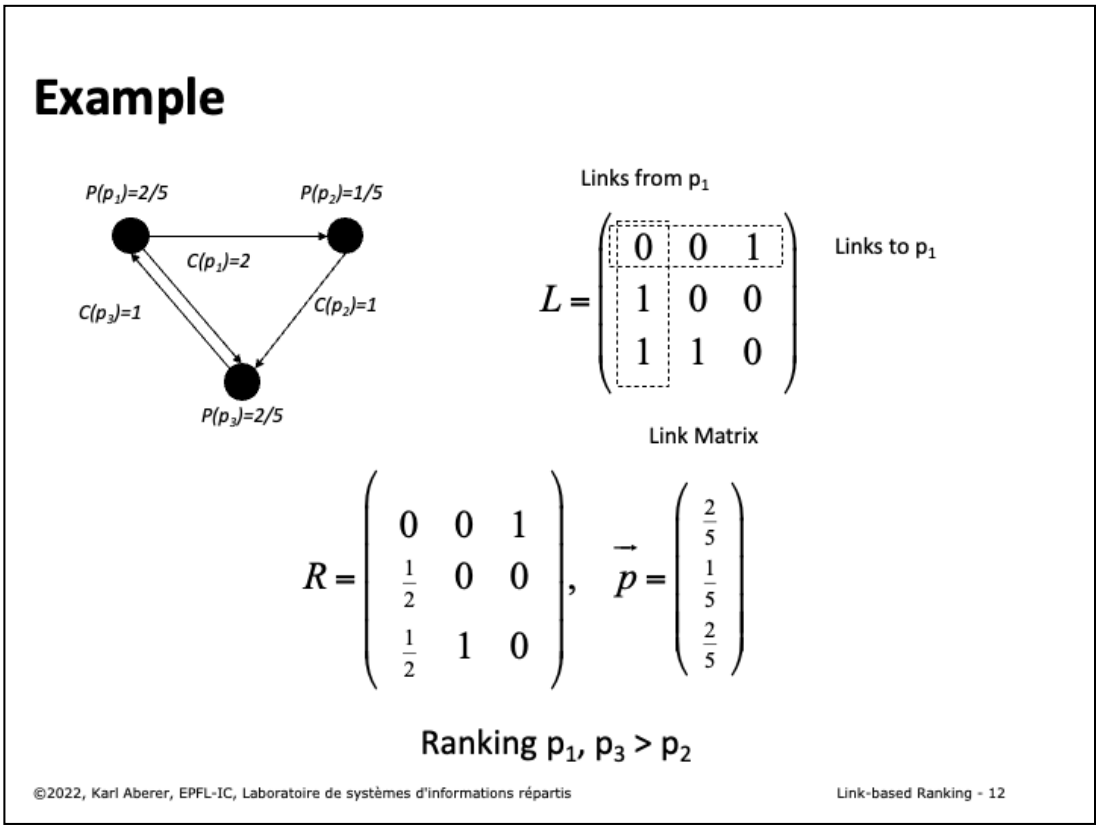
$$R_{ij} = \begin{cases} \frac{1}{C(p_j)}, if\ p_j \rightarrow p_i \\ 0, otherwise \end{cases}$$

$$\vec{p} = (P(p_1), \ldots, P(p_n))$$

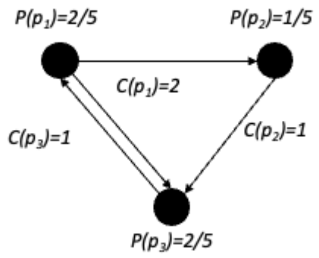
$$\vec{p} = R.\vec{p}, \quad \overrightarrow{\|p\|_1} = \sum_{i=1}^{n} p_i = 1$$

The vector of page relevance values is the Eigenvector of the matrix R for the largest Eigenvalue

In order to determine the solution to the recursive equation on the probabilities of a random walker to visit a page, we define a transition probability matrix R, which captures the probability of transitioning from one page to another. We also require that the probabilities of visiting a page add up to 1. With this formulation of the problem, the long-term visiting probabilities become the Eigenvector of matrix R. More precisely, they are the Eigenvector with the largest Eigenvalue.



Example



Links from p_1

$$L = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

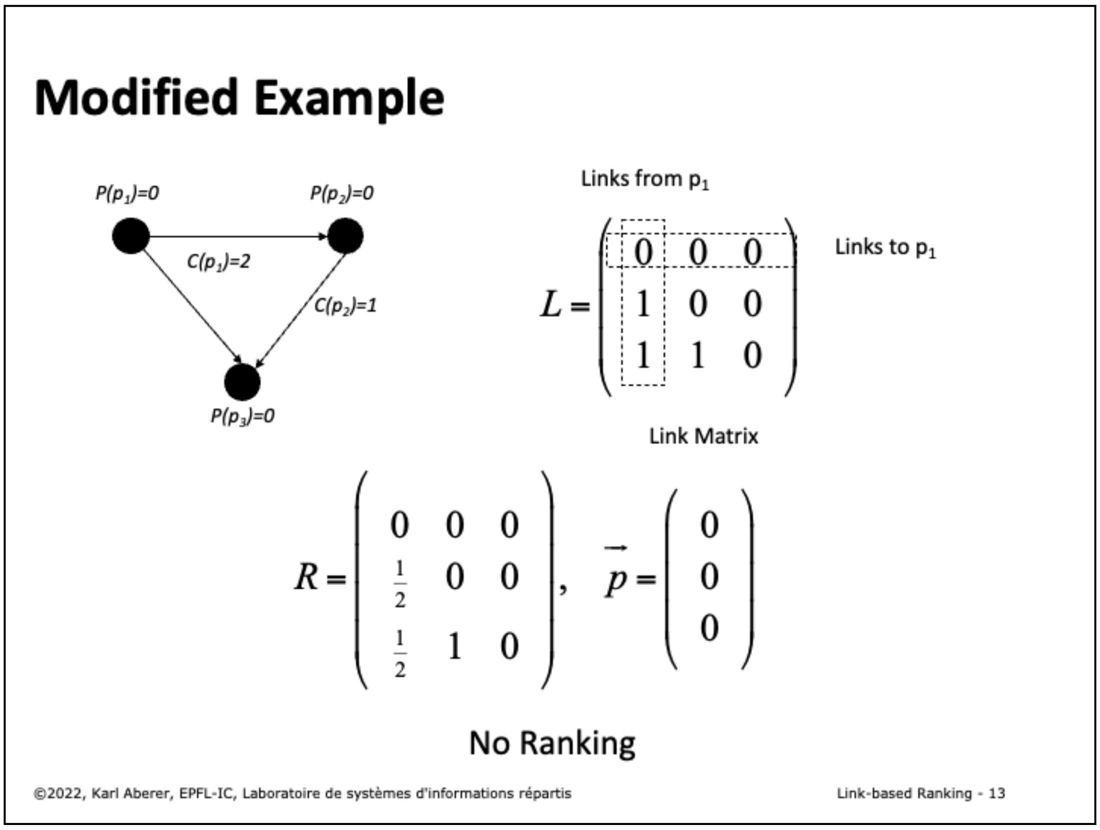
Links to p_1

Link Matrix

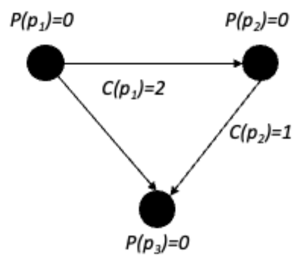
$$R = \begin{pmatrix} 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix}, \quad \vec{p} = \begin{pmatrix} \frac{2}{5} \\ \frac{1}{5} \\ \frac{2}{5} \end{pmatrix}$$

Ranking $p_1, p_3 > p_2$

This example illustrates the computation of the probabilities for visiting a specific Web page. The values $C(p_i)$ correspond to the transition probabilities. They can be derived from the link matrix. The link matrix is defined as $L_{ij}=1$ if there is a link from p_j to p_i. The link matrix is normalized by the outdegree, by dividing the values in the columns by the sum of the values found in the column, resulting in matrix R. The probability of a random walker visiting a node is then obtained from the Eigenvector of this matrix.



Modified Example



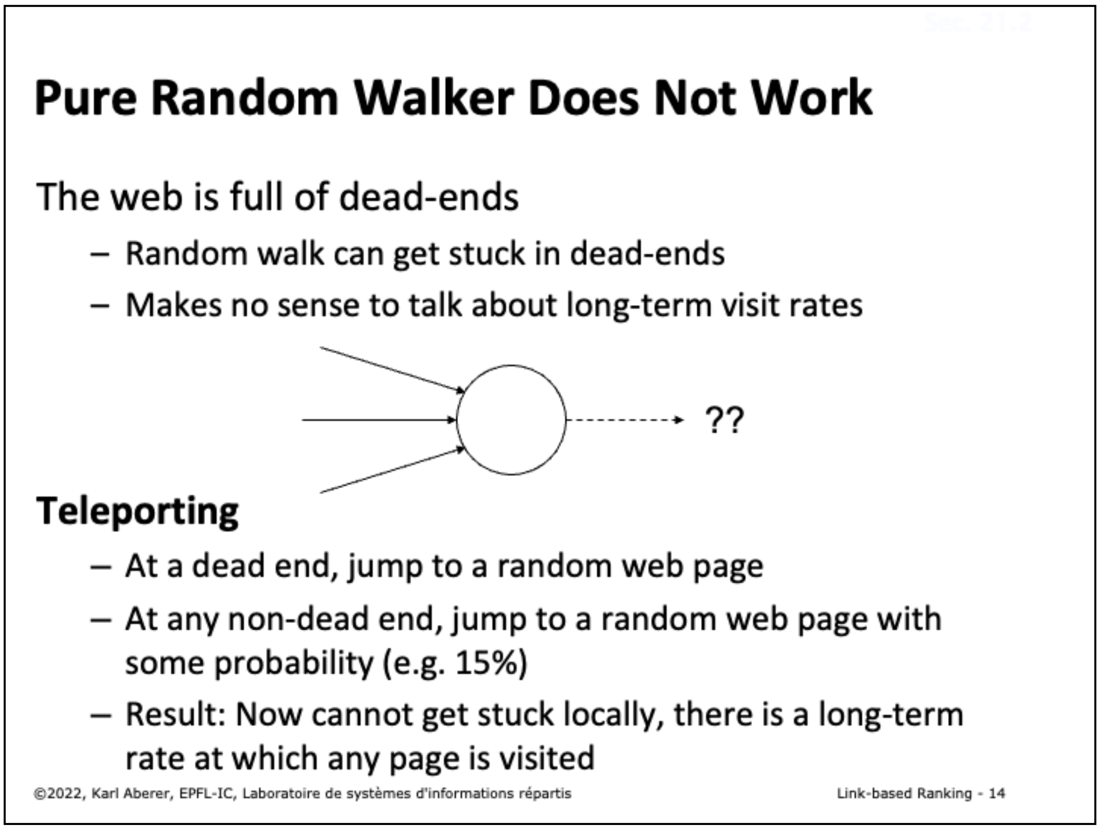
$$\begin{array}{cc} & \text{Links from } p_1 \\ L = & \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} \text{ Links to } p_1 \end{array}$$

Link Matrix

$$R = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix}, \quad \vec{p} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

No Ranking

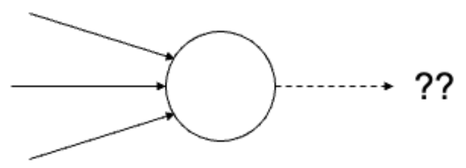
This example illustrates a problem with the random walker as we have formulated, the existence of dead ends. We see that there exists a node p_3 that is a "sink of rank". Any random walk ends up in this sink, and therefore the other nodes do not receive any ranking weight. Consequently, also the rank of sink does not. Therefore, the only solution to the equation p=Rp is the zero vector.



Pure Random Walker Does Not Work

The web is full of dead-ends

- Random walk can get stuck in dead-ends
- Makes no sense to talk about long-term visit rates



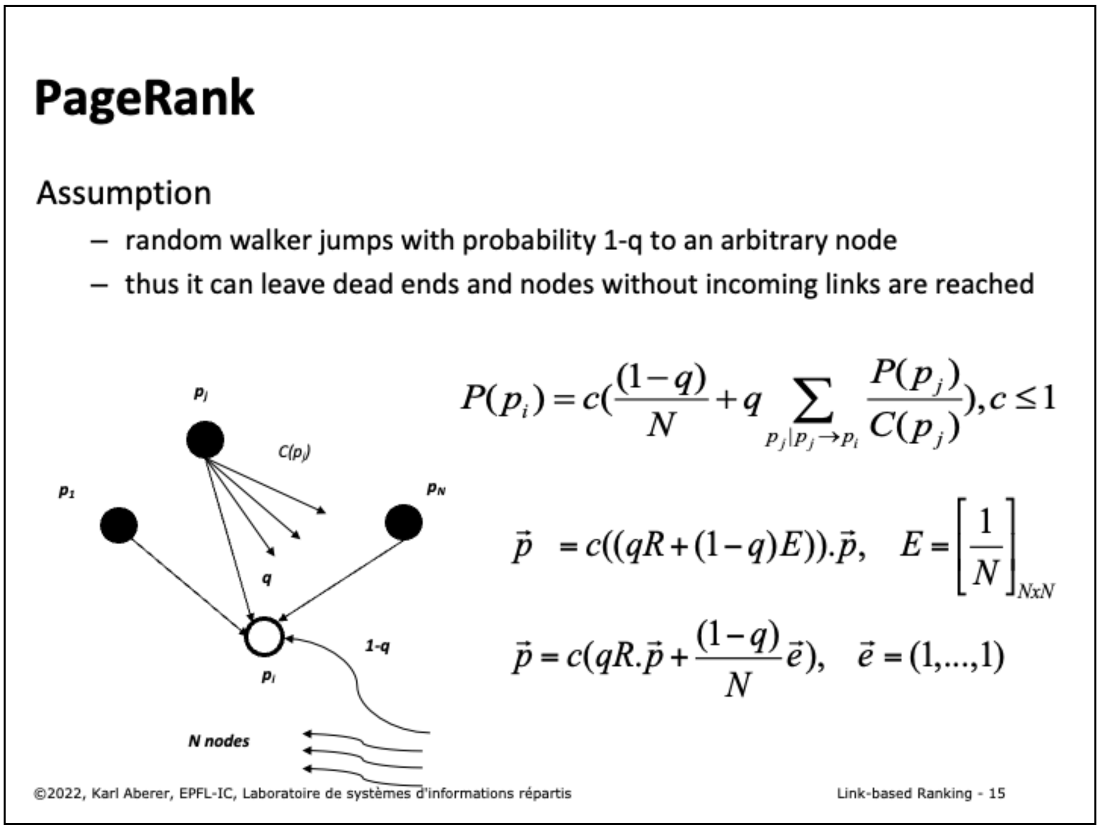
Teleporting

- At a dead end, jump to a random web page
- At any non-dead end, jump to a random web page with some probability (e.g. 15%)
- Result: Now cannot get stuck locally, there is a long-term rate at which any page is visited

Link-based Ranking - 14

A practical problem with the random walker is the fact that there exist Web pages that have no outgoing links. Thus, the random walker would get stuck. To address this problem, the concept of teleporting is introduced, where the random walker jumps to a randomly selected Web page with a given probability. If the random walker arrives at a dead end, it will then always jump to a randomly selected page.

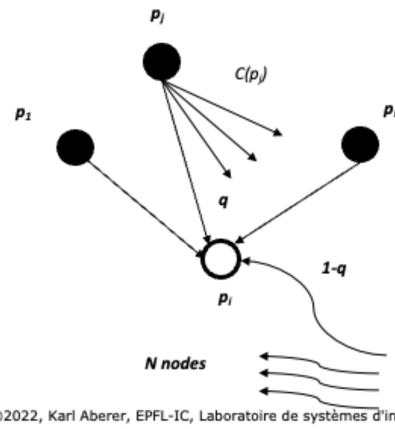
Another problem are pages that have no incoming links: they would never be reached by the random walker, and the weight that they could provide to other pages would not be considered. This problem is also addressed by teleporting.



PageRank

Assumption

- random walker jumps with probability 1-q to an arbitrary node
- thus it can leave dead ends and nodes without incoming links are reached

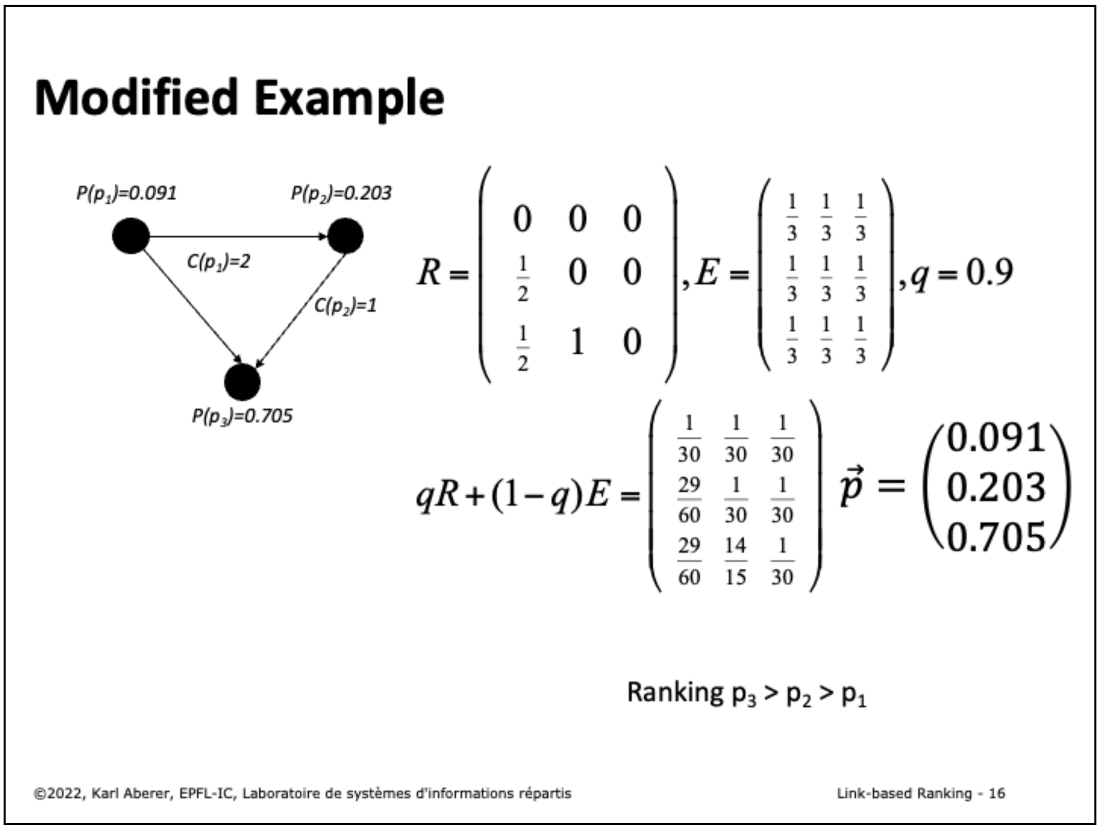


$$P(p_i) = c\left(\frac{(1-q)}{N} + q \sum_{p_j | p_j \to p_i} \frac{P(p_j)}{C(p_j)}\right), c \leq 1$$

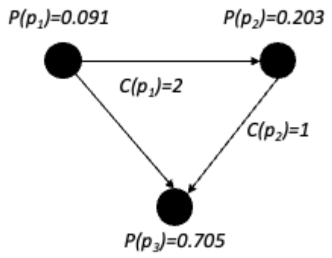
$$\vec{p} = c((qR + (1-q)E)).\vec{p}, \quad E = \left[\frac{1}{N}\right]_{NxN}$$

$$\vec{p} = c(qR.\vec{p} + \frac{(1-q)}{N}\vec{e}), \quad \vec{e} = (1,\ldots,1)$$

We give now the formal specification of the random walker with teleporting. At each step, the random walker makes a jump with a probability 1-q and any of the N pages is reached with the same probability. Therefore, an additional term is (1-q)/N is added to the probability for reaching a given page. Reformulating the equation for the probabilities in matrix form, results in adding a NxN Matrix E with all entries being 1/N. This is equivalent to saying that with probability 1/N transitions among any pairs of nodes (including transition from a node to itself) are performed. Since the vector p has norm 1, i.e., the sum of the components is exactly 1, E.p=e. Based on this property, an alternative formulation for the equation can be given. The method described is called PageRank and is used by Google for Web ranking. By modifying the values of the matrix E also a priori knowledge about the relative importance of pages can be added to the ranking algorithm.



Modified Example



$$R = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 1 & 0 \end{pmatrix}, E = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}, q = 0.9$$

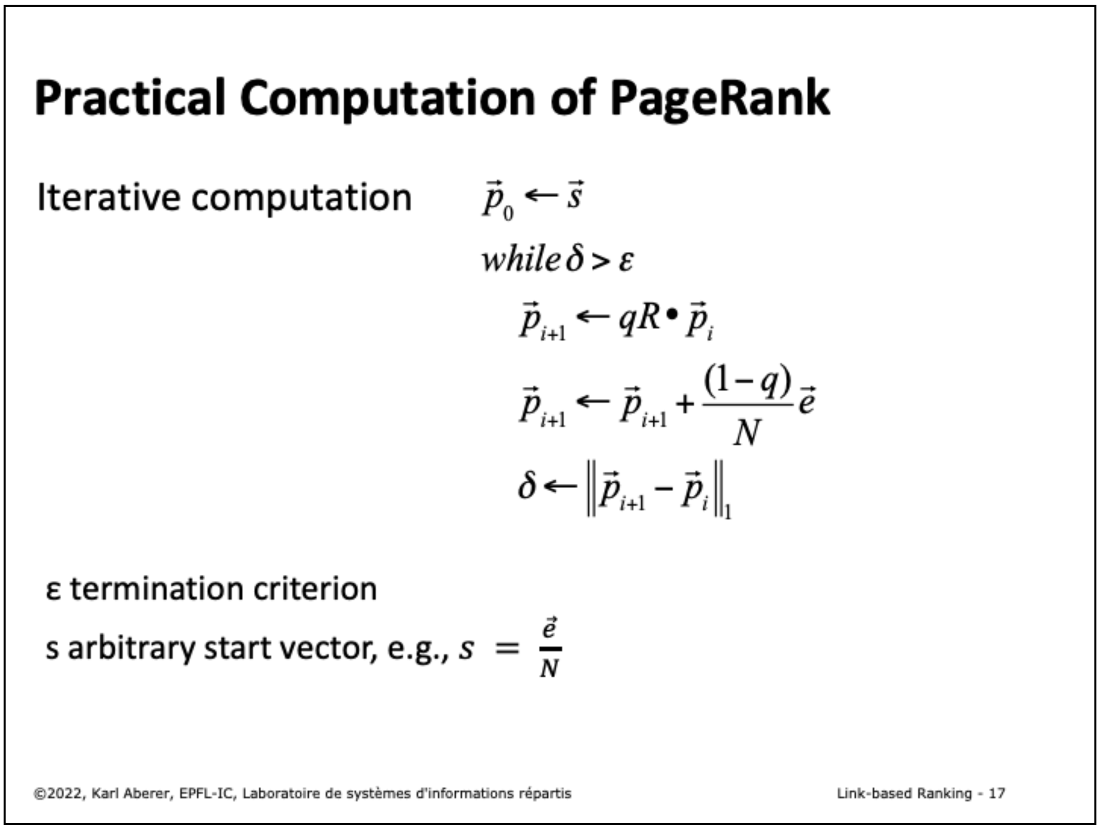
$$qR + (1-q)E = \begin{pmatrix} \frac{1}{30} & \frac{1}{30} & \frac{1}{30} \\ \frac{29}{60} & \frac{1}{30} & \frac{1}{30} \\ \frac{29}{60} & \frac{14}{15} & \frac{1}{30} \end{pmatrix} \quad \vec{p} = \begin{pmatrix} 0.091 \\ 0.203 \\ 0.705 \end{pmatrix}$$

Ranking $p_3 > p_2 > p_1$

©2022, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Link-based Ranking - 16

With the modification of rank computation using a source of rank, we obtain for our example a non-trivial ranking which appears to match intuition about the relative importance of the pages in the graph well.



Practical Computation of PageRank

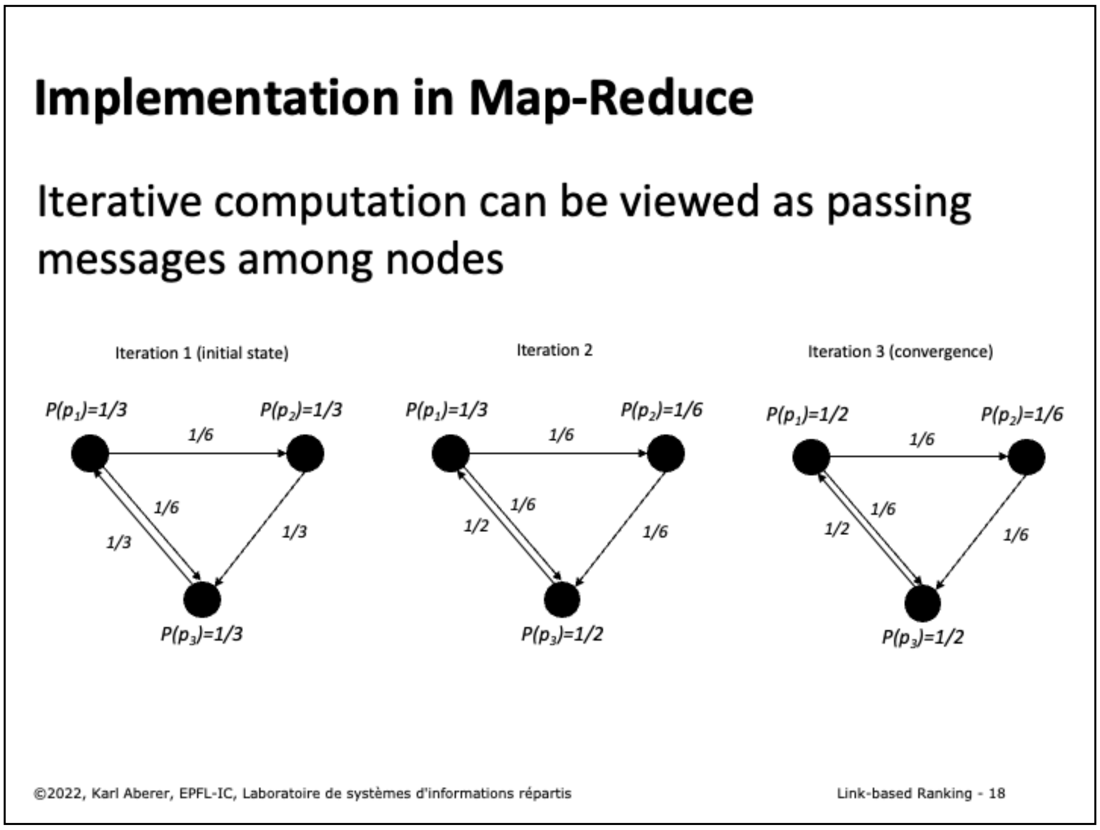
Iterative computation

```
$\vec{p}_0 \leftarrow \vec{s}$
$while\ \delta > \varepsilon$
    $\vec{p}_{i+1} \leftarrow qR \bullet \vec{p}_i$
    $\vec{p}_{i+1} \leftarrow \vec{p}_{i+1} + \frac{(1-q)}{N}\vec{e}$
    $\delta \leftarrow \left\| \vec{p}_{i+1} - \vec{p}_i \right\|_1$
```

ε termination criterion

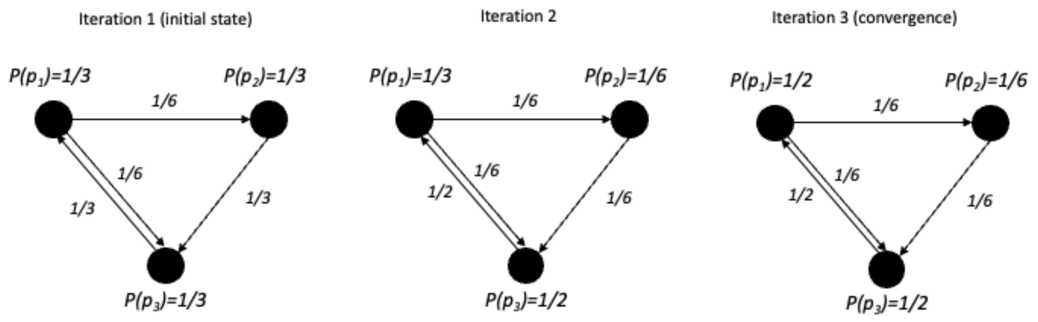
s arbitrary start vector, e.g., $s = \frac{\vec{e}}{N}$

For the practical computation of the PageRank ranking an iterative approach can be used. The vector e is used to add a source of rank. It can uniformly distribute weights to all pages, but it could also incorporate pre-existing knowledge on the importance of pages and bias the ranking towards them. The vector can also be used as initial probability distribution.



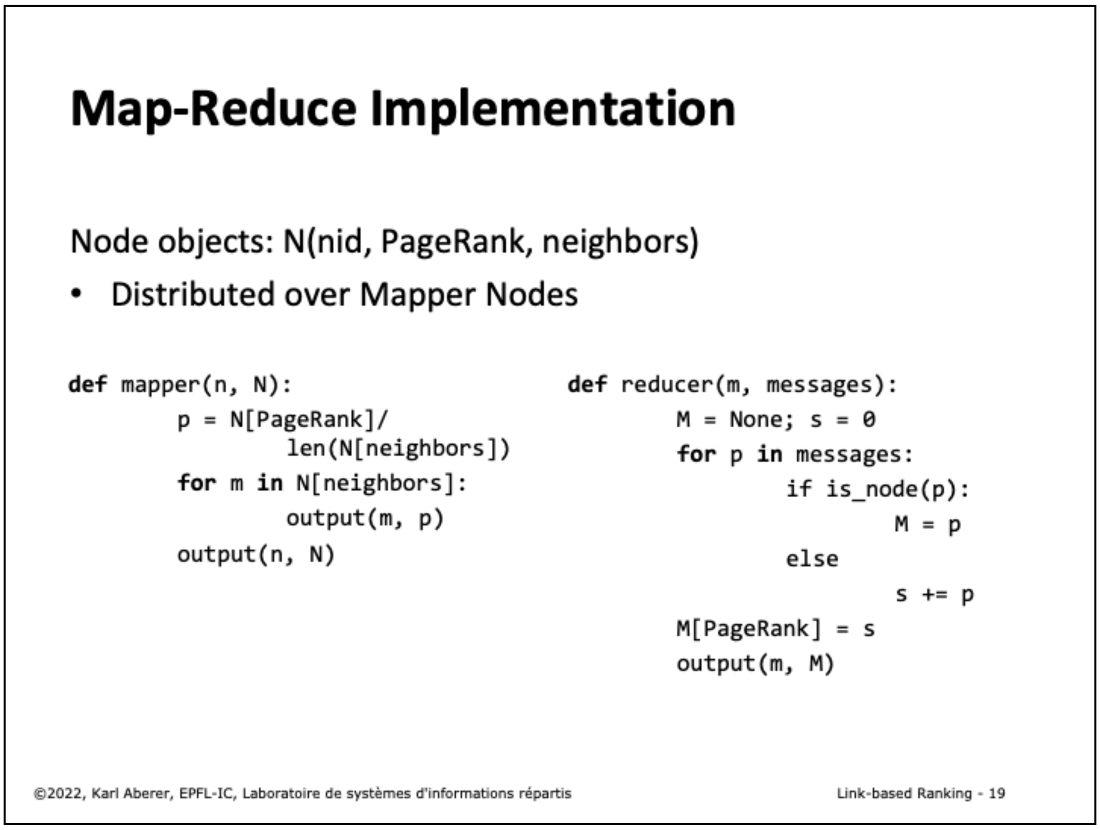
Implementation in Map-Reduce

Iterative computation can be viewed as passing messages among nodes



For practical computation of a ranking for a Web graph, the link matrices are huge and can potentially not be processed on a single node. This problem is similar to the one we have addressed for the construction of inverted files and for which one solution was the use of the map-reduce computing paradigm. We can use this paradigm as well for computing PageRank in a distributed way.

To that end, we consider the iterative computation of the PageRank values as a process of transmitting messages among the nodes. In the computation of the PageRank values a node receives in each step from the incoming links weights. It then aggregates those weights, before the next round of computation is performed. Representing the computation in this form leads directly to an approach for implementing it in the map-reduce programming model.



Map-Reduce Implementation

Node objects: N(nid, PageRank, neighbors)

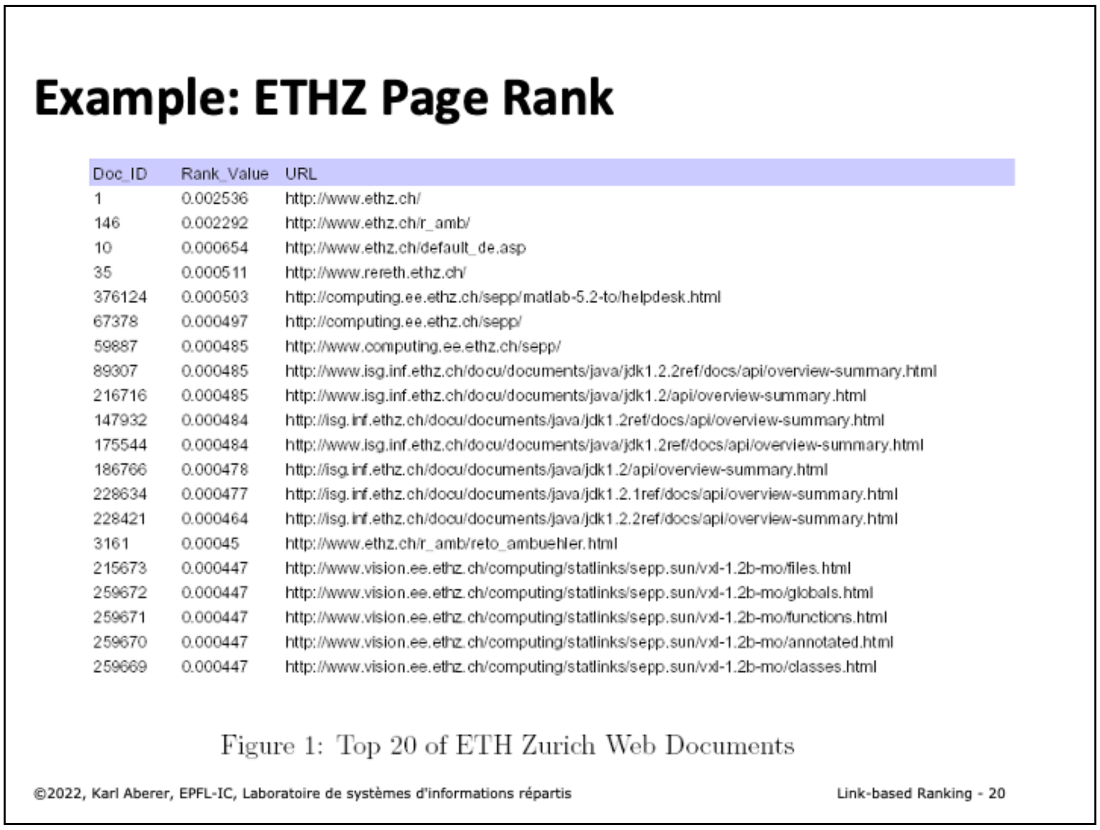
- Distributed over Mapper Nodes

```
def mapper(n, N):
        p = N[PageRank]/
                len(N[neighbors])
        for m in N[neighbors]:
                output(m, p)
        output(n, N)
```

```
def reducer(m, messages):
        M = None; s = 0
        for p in messages:
                if is_node(p):
                        M = p
                else
                        s += p
        M[PageRank] = s
        output(m, M)
```

In the map-reduce implementation two types of outputs are processed. Node objects that contain a node identifier nid, the current PageRank values, and a list of the neighbour ids. The mapper receives a node object N. It computes the contribution of its rank to its neighbours and outputs the neighbour node ids together with their weights. It also outputs the nodeid together with the node object.

The reducers collect all messages concerning a given node m. It receives both the node object and all weights. The weights are added up and used to update the node object. The reducer outputs the updated node object that will be serving as inputs to mappers in a subsequent map-reduce phase. Therefore, each execution of a mapreduce job corresponds to one iteration of the PageRank algorithm.

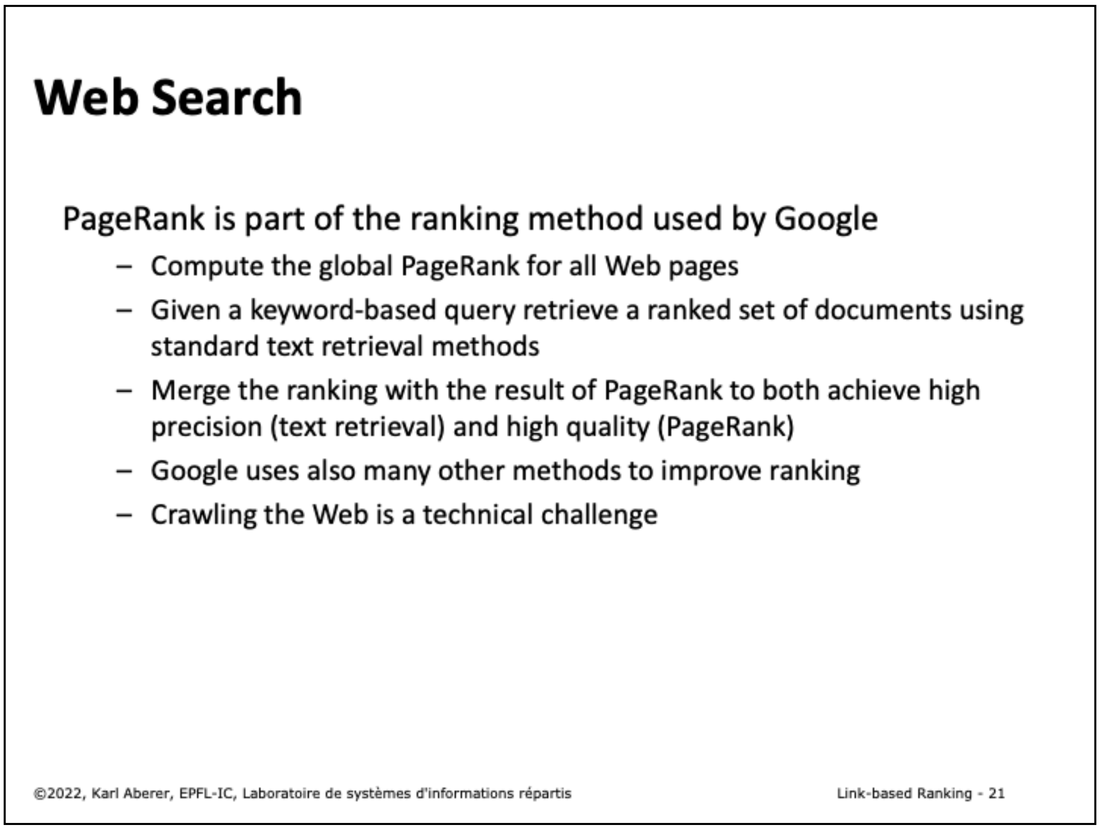


Example: ETHZ Page Rank

Doc_ID	Rank_Value	URL
1	0.002536	http://www.ethz.ch/
146	0.002292	http://www.ethz.ch/r_amb/
10	0.000654	http://www.ethz.ch/default_de.asp
35	0.000511	http://www.rereth.ethz.ch/
376124	0.000503	http://computing.ee.ethz.ch/sepp/matlab-5.2-to/helpdesk.html
67378	0.000497	http://computing.ee.ethz.ch/sepp/
59887	0.000485	http://www.computing.ee.ethz.ch/sepp/
89307	0.000485	http://www.isg.inf.ethz.ch/docu/documents/java/jdk1.2.2ref/docs/api/overview-summary.html
216716	0.000485	http://www.isg.inf.ethz.ch/docu/documents/java/jdk1.2/api/overview-summary.html
147932	0.000484	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2ref/docs/api/overview-summary.html
175544	0.000484	http://www.isg.inf.ethz.ch/docu/documents/java/jdk1.2ref/docs/api/overview-summary.html
186766	0.000478	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2/api/overview-summary.html
228634	0.000477	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2.1ref/docs/api/overview-summary.html
228421	0.000464	http://isg.inf.ethz.ch/docu/documents/java/jdk1.2.2ref/docs/api/overview-summary.html
3161	0.00045	http://www.ethz.ch/r_amb/reto_ambuehler.html
215673	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/files.html
259672	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/globals.html
259671	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/functions.html
259670	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/annotated.html
259669	0.000447	http://www.vision.ee.ethz.ch/computing/statlinks/sepp.sun/vxl-1.2b-mo/classes.html

Figure 1: Top 20 of ETH Zurich Web Documents

These are the top documents from the PageRank ranking of all Web pages at ETHZ (Data from 2001). It is interesting to see that documents related to Java documentation receive high ranking values. This is related to the fact that these documents have many internal cross-references.



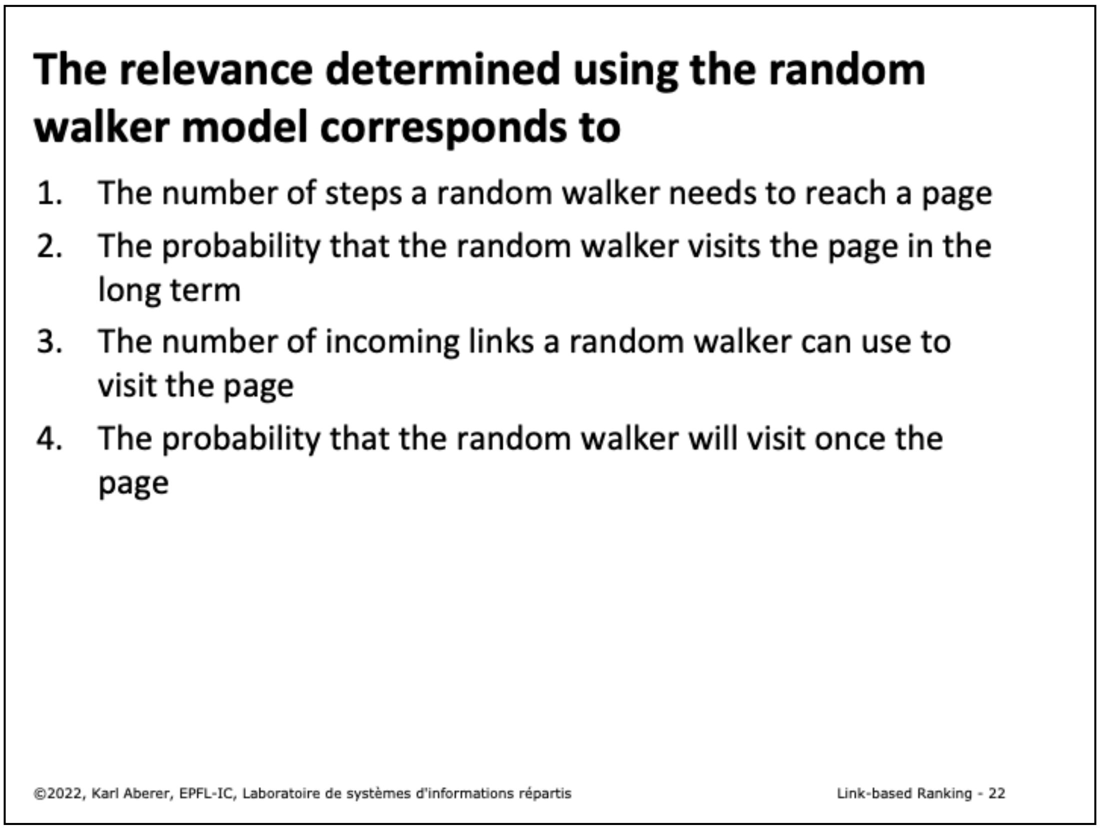
Web Search

PageRank is part of the ranking method used by Google

- Compute the global PageRank for all Web pages
- Given a keyword-based query retrieve a ranked set of documents using standard text retrieval methods
- Merge the ranking with the result of PageRank to both achieve high precision (text retrieval) and high quality (PageRank)
- Google uses also many other methods to improve ranking
- Crawling the Web is a technical challenge

PageRank is used as one metrics to rank result documents in Google. At the basis Google uses text retrieval methods to retrieve relevant documents and then applies PageRank to create a more appropriate ranking. Google uses also many other methods to improve ranking, e.g., today largely based on personal information collected from users, like search history and pages visited. The details of the ranking methods are trade secrets of the Web search engine providers.

Building a Web Search engine requires to solve several additional problems, beyond providing a ranking system. Efficient Web crawling requires algorithms that can traverse the Web avoiding redundant accesses to pages and techniques for managing large link databases.

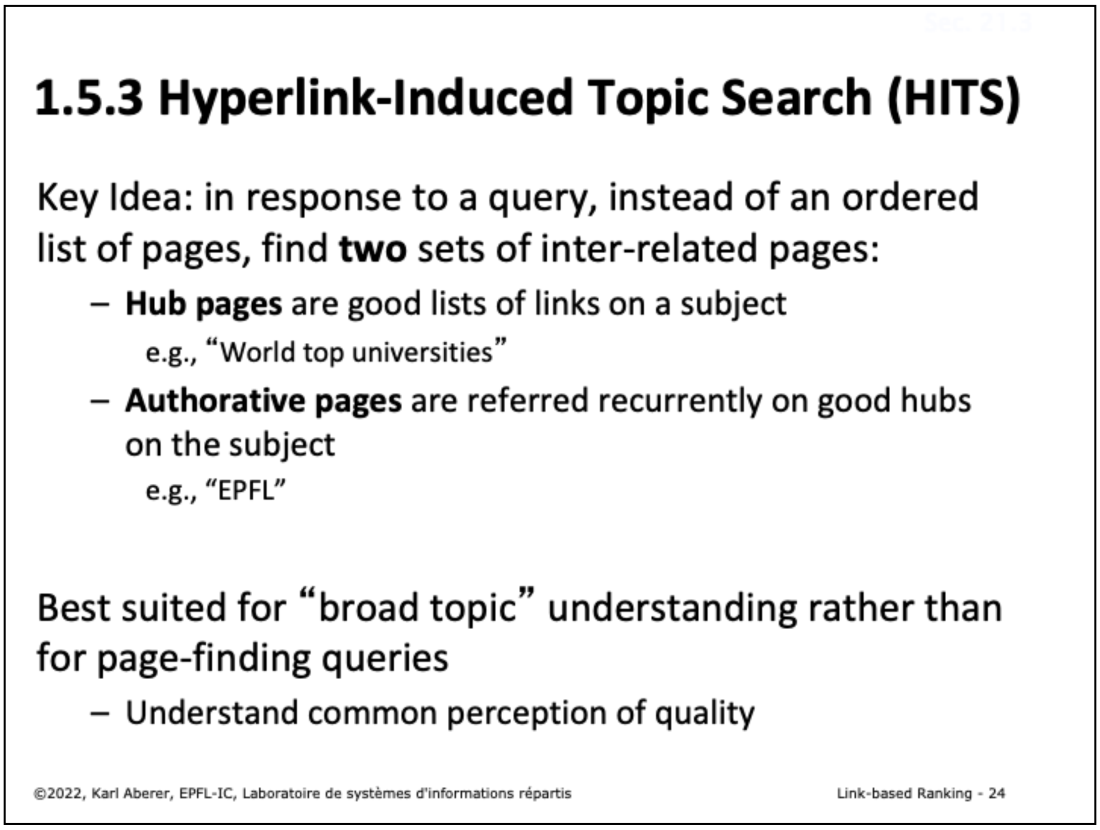


The relevance determined using the random walker model corresponds to

1. The number of steps a random walker needs to reach a page
2. The probability that the random walker visits the page in the long term
3. The number of incoming links a random walker can use to visit the page
4. The probability that the random walker will visit once the page

Consider a random jump matrix with entries 1/3 in the first column and 0 otherwise. It means

1. A random walker can always leave node 1 even without outgoing edges
2. A random walker can always reach node 1, even without incoming edges
3. A random walker can always leave node 2, even without outgoing edges
4. none of the above



Sec. 21.3

1.5.3 Hyperlink-Induced Topic Search (HITS)

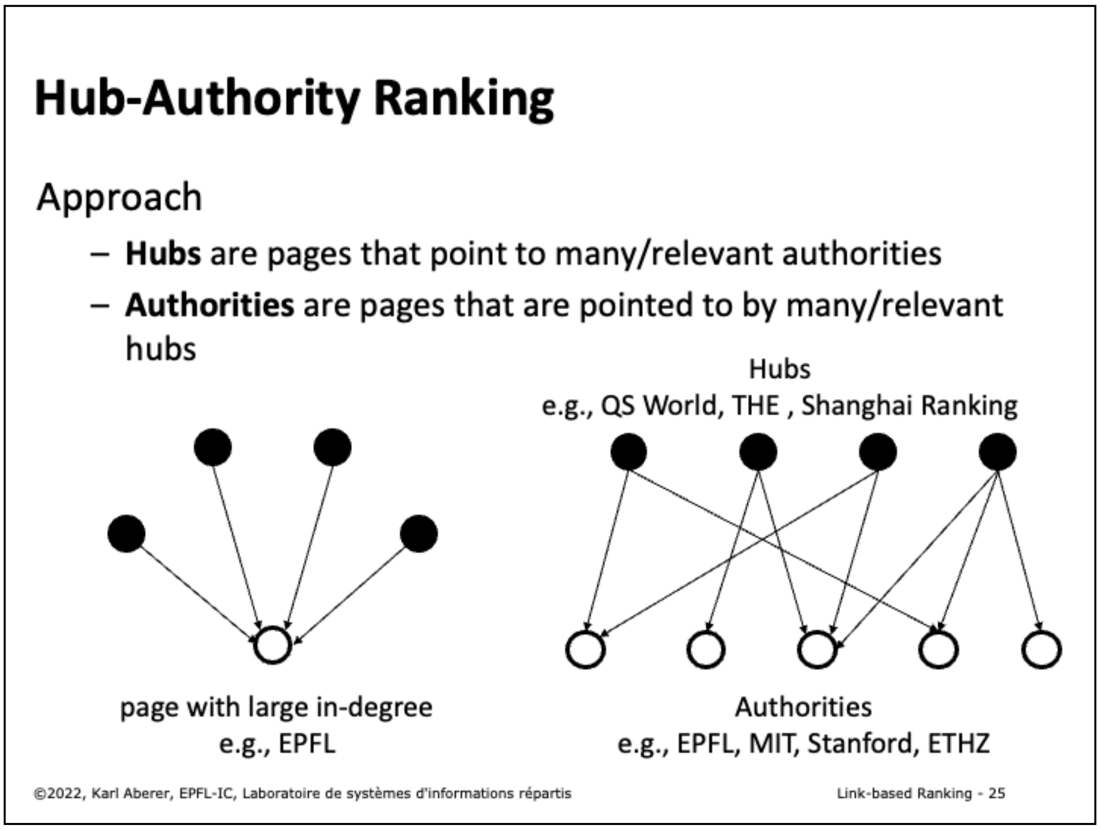
Key Idea: in response to a query, instead of an ordered list of pages, find **two** sets of inter-related pages:

- **Hub pages** are good lists of links on a subject
  e.g., “World top universities”
- **Authorative pages** are referred recurrently on good hubs on the subject
  e.g., “EPFL”

Best suited for “broad topic” understanding rather than for page-finding queries

- Understand common perception of quality

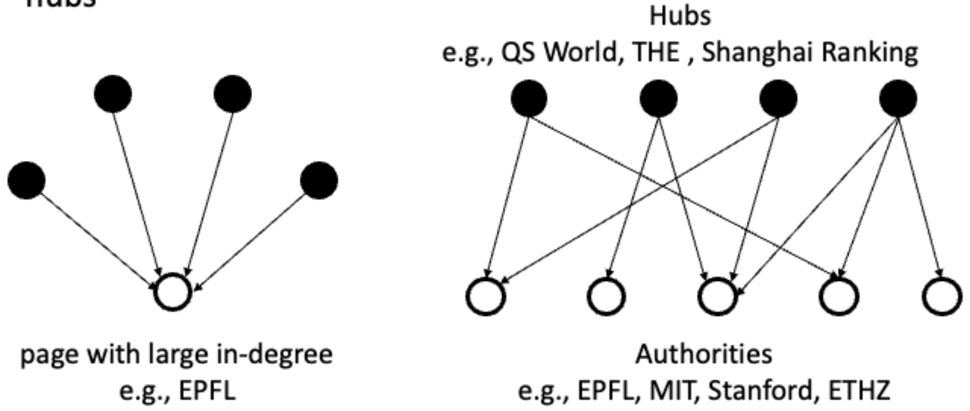
The basic idea of HITS is to apply not a single measure for link-based relevance of a document, but to distinguish two different roles documents can play. Hub pages are pages that provide references to high quality pages, whereas authority pages are high quality pages. The method has been conceived for understand a larger topic in general and obtain an overview of the essential contents related to a given topic. It can nevertheless also be used as an alternative ranking model for Web search, that provides a more refined quality evaluation of Web pages.



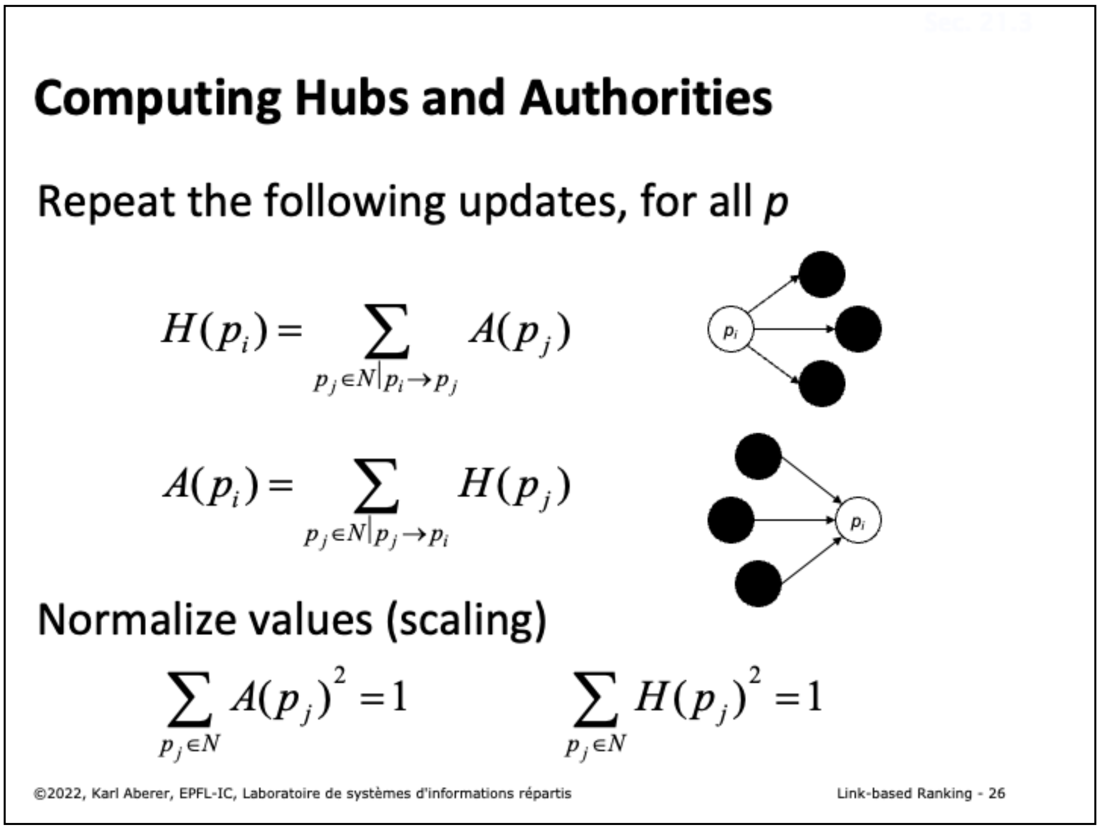
Hub-Authority Ranking

Approach

- **Hubs** are pages that point to many/relevant authorities
- **Authorities** are pages that are pointed to by many/relevant hubs


Hubs
e.g., QS World, THE , Shanghai Ranking
page with large in-degree
e.g., EPFL
Authorities
e.g., EPFL, MIT, Stanford, ETHZ

Hub-authority ranking is, like PageRank, based on a quantitative analysis of the link structure. Different to PageRank two different measures are considered. The number of incoming links as a measure for authority, and the number of links pointing to an authority as a measure for the quality of a hub. The example shows of how in this way authorative pages, such as university home pages, can be distinguished from hub pages, such as portal sites referencing universities.

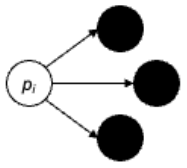


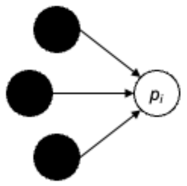
Computing Hubs and Authorities

Repeat the following updates, for all *p*

$$H(p_i) = \sum_{p_j \in N | p_i \to p_j} A(p_j)$$

$$A(p_i) = \sum_{p_j \in N | p_j \to p_i} H(p_j)$$





Normalize values (scaling)

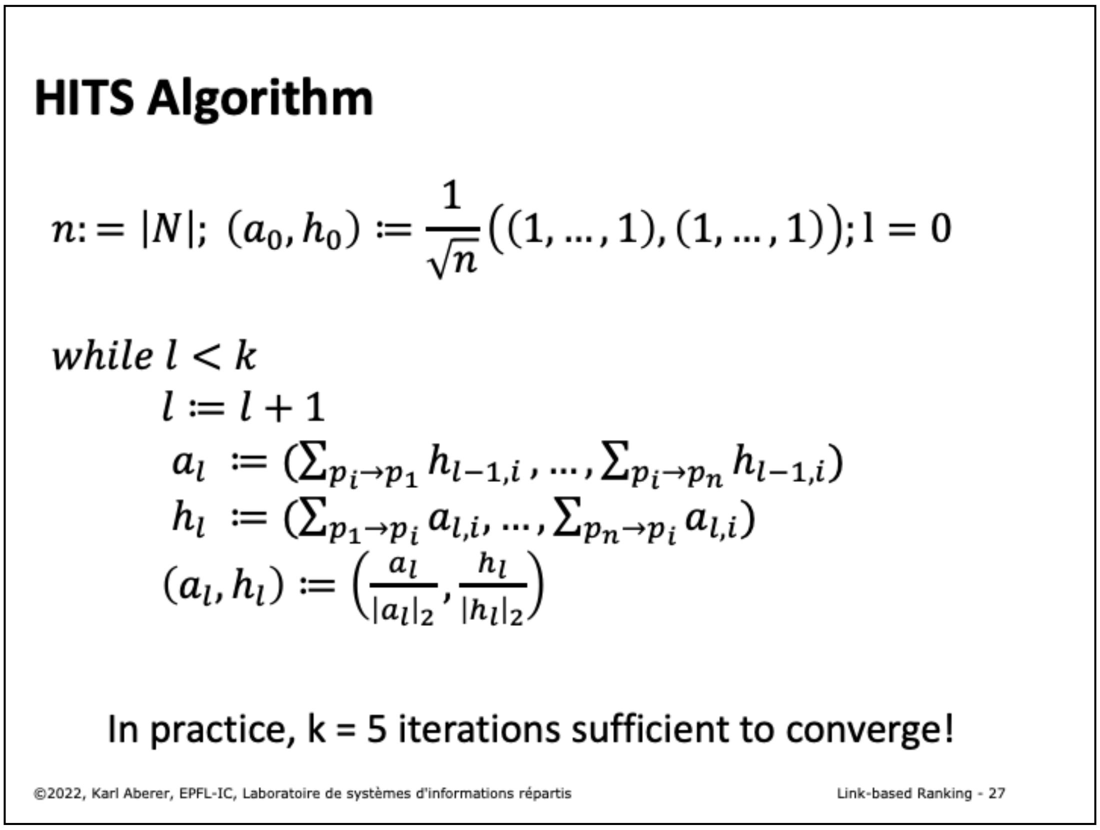
$$\sum_{p_j \in N} A(p_j)^2 = 1 \qquad \sum_{p_j \in N} H(p_j)^2 = 1$$

©2022, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Link-based Ranking - 26

As in PageRank the approach is to consider the ranking value of pages from which hyperlinks are emanating in the weighting of the influence the hyperlink has on the page it is pointing to. This results directly in a recursive formulation of the ranking values for hub and authority weights. Note that in the HITS method presented here you find subtle differences how those equations are formulated, as compared to PageRank:

1. The weights are not split among the outgoing links, but each link transfers the while hub or authority weight from the originating page
2. Since the weights are not split, the ranking values need to be normalized
3. The normalization uses L2 norm, and not L1 norm as for PageRank.



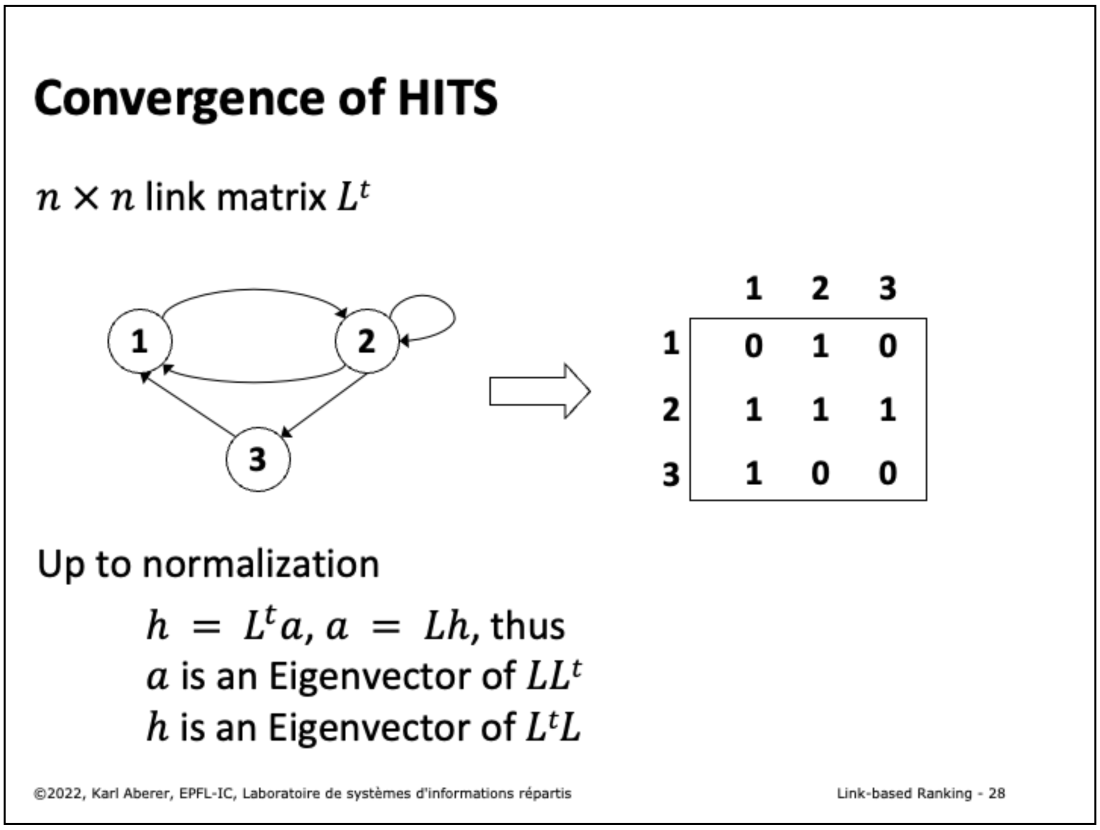
HITS Algorithm

```
$n := |N|;\ (a_0, h_0) := \frac{1}{\sqrt{n}}((1, \dots, 1), (1, \dots, 1)); l = 0$

$while\ l < k$
    $l := l + 1$
    $a_l := (\sum_{p_i \to p_1} h_{l-1,i}, \dots, \sum_{p_i \to p_n} h_{l-1,i})$
    $h_l := (\sum_{p_1 \to p_i} a_{l,i}, \dots, \sum_{p_n \to p_i} a_{l,i})$
    $(a_l, h_l) := \left(\frac{a_l}{|a_l|_2}, \frac{h_l}{|h_l|_2}\right)$
```

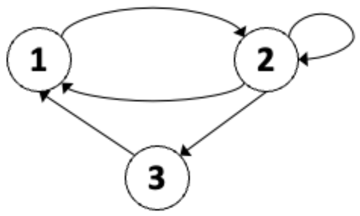
In practice, k = 5 iterations sufficient to converge!

Similarly, as for PageRank, the equations can be solved using iteration. Here we show a possible realization of such an iterative computation, using uniformly distributed weights for initialization.



Convergence of HITS

$n \times n$ link matrix L^t



$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 1 & 0 \\ 2 & 1 & 1 & 1 \\ 3 & 1 & 0 & 0 \end{array}$$

Up to normalization

$h = L^t a, a = Lh$, thus

a is an Eigenvector of LL^t

h is an Eigenvector of $L^t L$

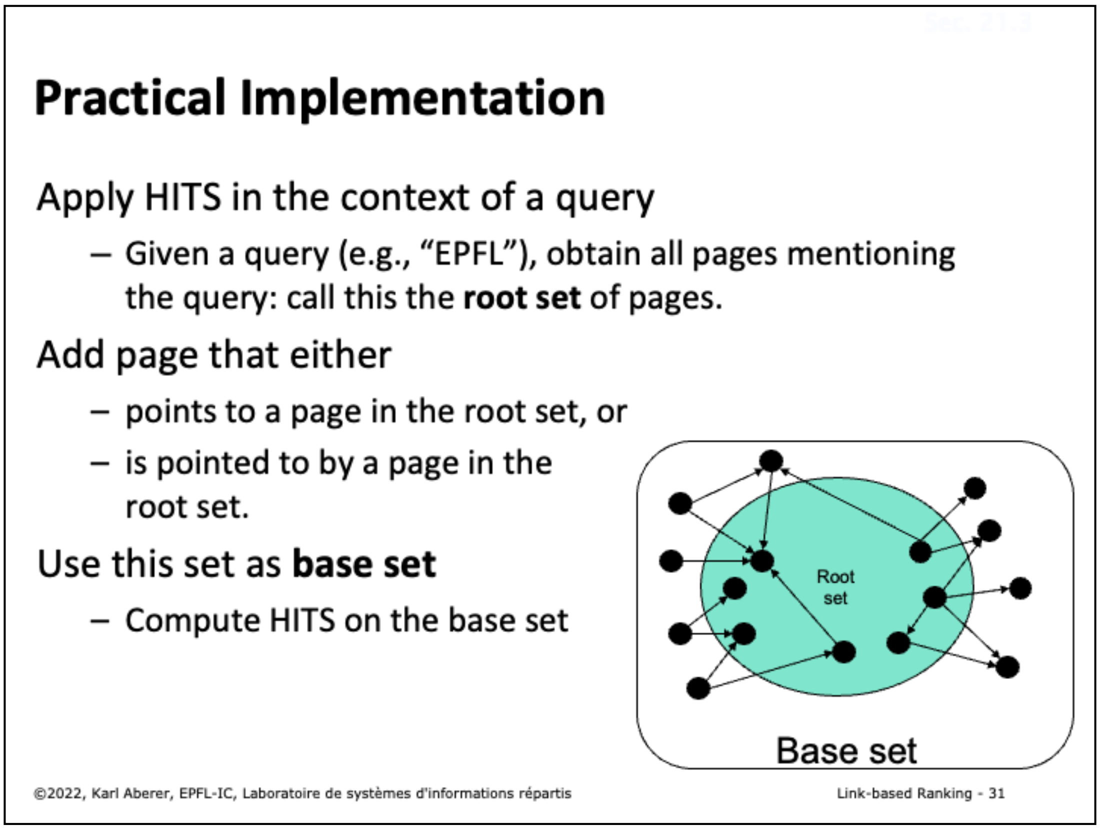
When formulating the HITS equations in matrix form, using the link matrix L, we see that the authority and hub weights correspond to the Eigenvectors of the matrices LL^t and L^tL This shows that the iterative computation with normalization of the hub and authority values will converge to the principal Eigenvectors of the matrices LL^t and L^tL

When computing HITS, the initial values

1. Are set all to 1
2. Are set all to $\frac{1}{n}$
3. Are set all to $\frac{1}{\sqrt{n}}$
4. Are chosen randomly

If the first column of matrix L is (0,1,1,1) and all other entries are 0 then the authority values

1. $(0,1,1,1)$
2. $(0, 1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$
3. $(1, 1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})$
4. $(1,0,0,0)$



Practical Implementation

Apply HITS in the context of a query

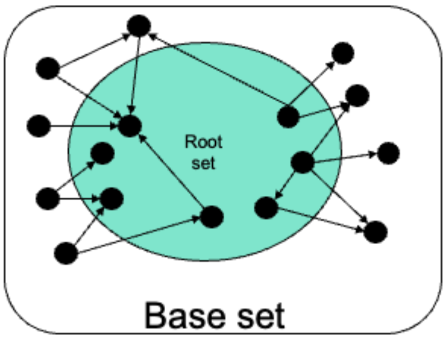
- Given a query (e.g., "EPFL"), obtain all pages mentioning the query: call this the **root set** of pages.

Add page that either

- points to a page in the root set, or
- is pointed to by a page in the root set.

Use this set as **base set**

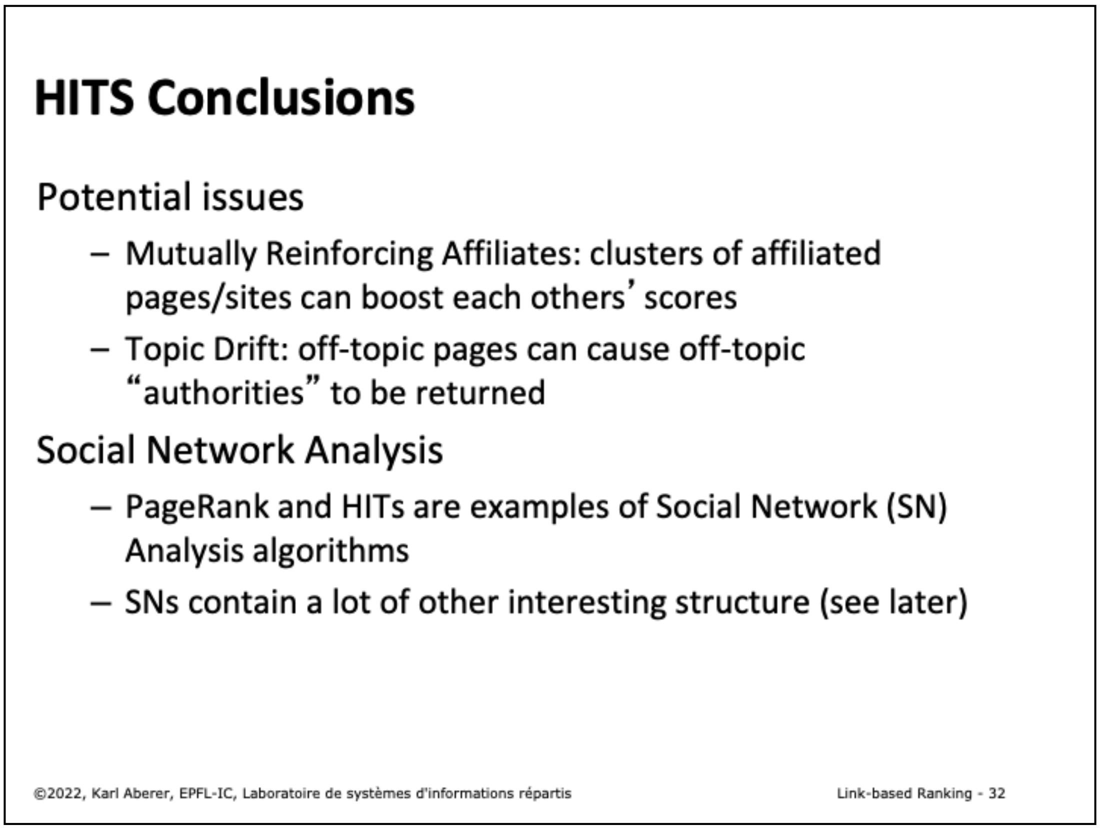
- Compute HITS on the base set



©2022, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis

Link-based Ranking - 31

One possible application of HITS is to compute the ranking on the complete Web Graph, as it is done with PageRank. Another way to use it (and this is how it was initially conceived), is to apply it in the context of a given query, to rerank the results by promoting results with high authority and hub values. In order to perform this operation, first all results for a query are retrieved (using a standard text retrieval model). Then the neighboring pages (either pointing to a result page, or referred by a result page) are added to the set of pages, which is then called the base set. HITS is then computed on the base set. This makes sense, as in this way we both consider referred pages and referring pages for the relevant documents, which helps to identify both hubs and authorities.



HITS Conclusions

Potential issues

- Mutually Reinforcing Affiliates: clusters of affiliated pages/sites can boost each others' scores
- Topic Drift: off-topic pages can cause off-topic "authorities" to be returned

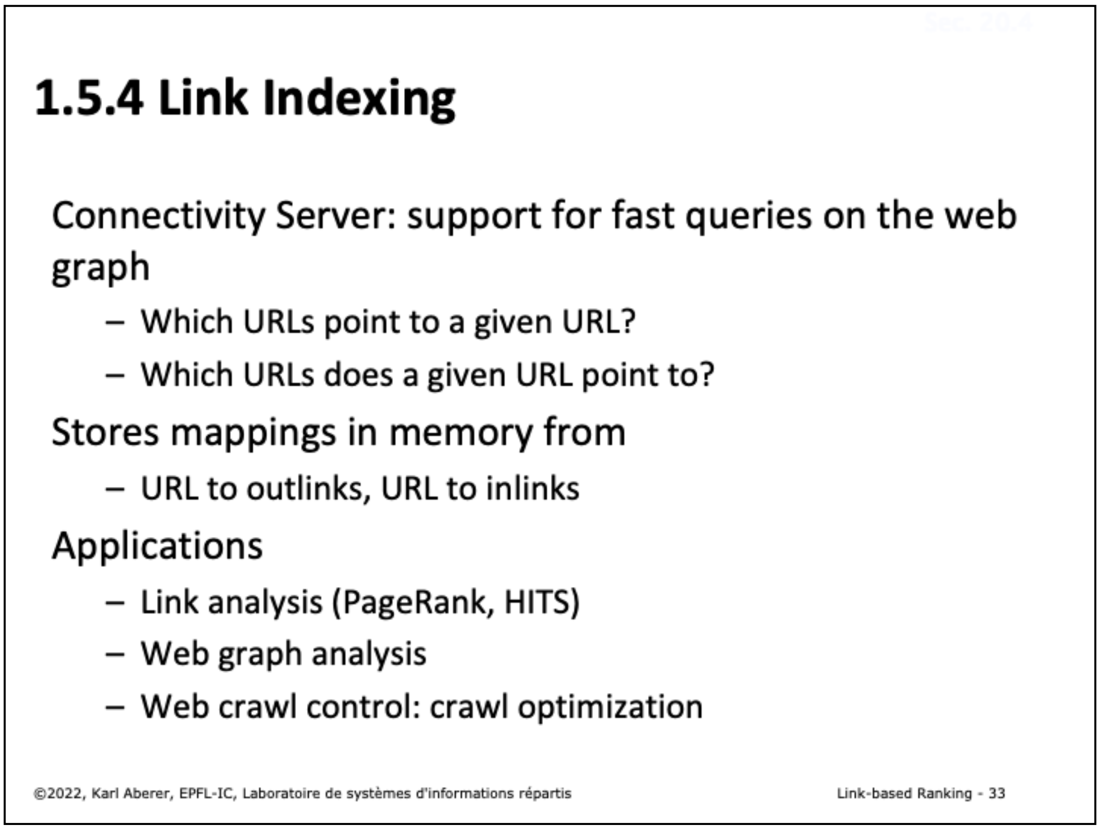
Social Network Analysis

- PageRank and HITs are examples of Social Network (SN) Analysis algorithms
- SNs contain a lot of other interesting structure (see later)

HITS suffers from similar potential problems related to the manipulation of the link structure through link spamming as PageRank. In addition, when performing a broad topic search and computing a base set for analysis, topic drift may occur, e.g., through the introduction of off-topic hubs. This is a problem that is similar to the issues of topic drift in pseudo-relevance feedback that we have observed earlier.

Both, HITS and PageRank are examples of social network analysis algorithms. We will introduce later other types of algorithms for this purpose, aiming at community detection.

For efficient implementation, link-based ranking algorithms require an efficient representation of the Web graph. This is a topic that we will explore next.



1.5.4 Link Indexing

Connectivity Server: support for fast queries on the web graph

- Which URLs point to a given URL?
- Which URLs does a given URL point to?

Stores mappings in memory from

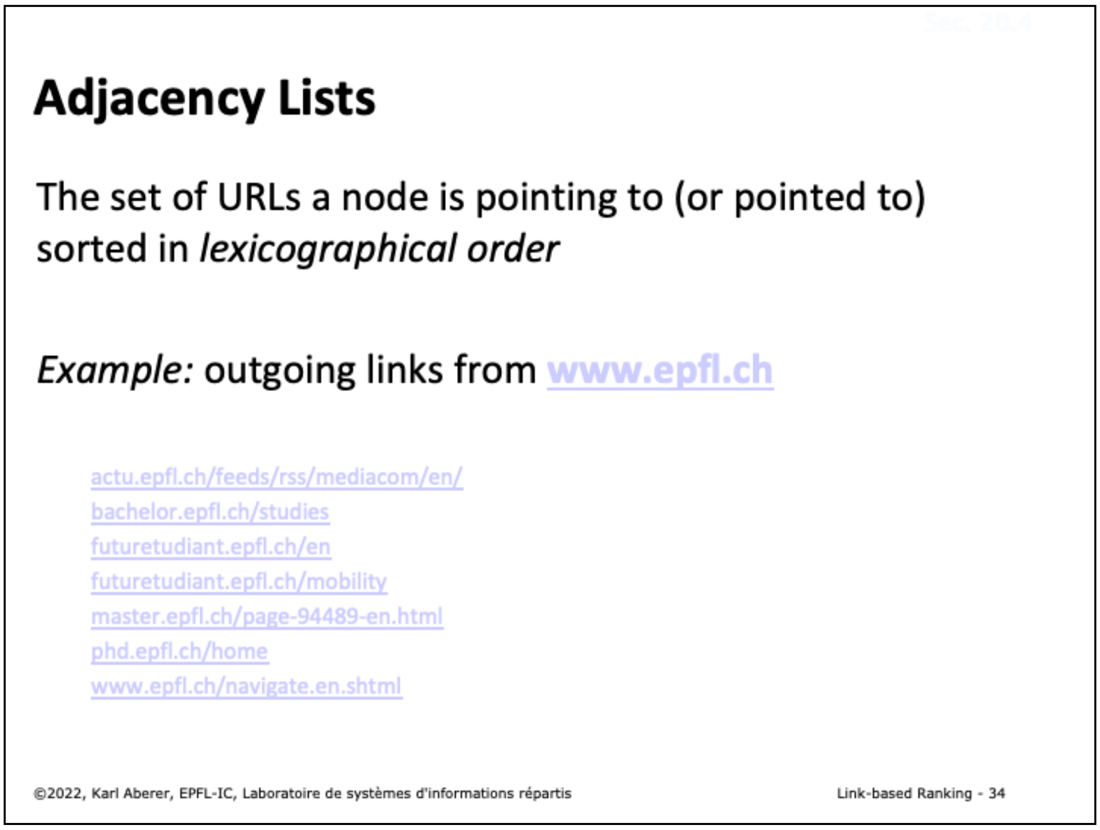
- URL to outlinks, URL to inlinks

Applications

- Link analysis (PageRank, HITS)
- Web graph analysis
- Web crawl control: crawl optimization

For text retrieval we introduced inverted files as an indexing structure for fast lookup of documents based on text queries. Using such an index it becomes possible to efficiently compute that statistics needed for the ranking model.

Similarly as for link-based ranking, for computing statistics on the Web graph we need an efficient indexing structure for the link graph. This is called a connectivity server. It allows to answer efficiently the queries relevant for Web graph analysis, namely which URLs a page points to and is pointed to. Beyond performing Web graph analysis such a connectivity server has also other applications, especially for controlling Web crawling.



Sec. 20.4

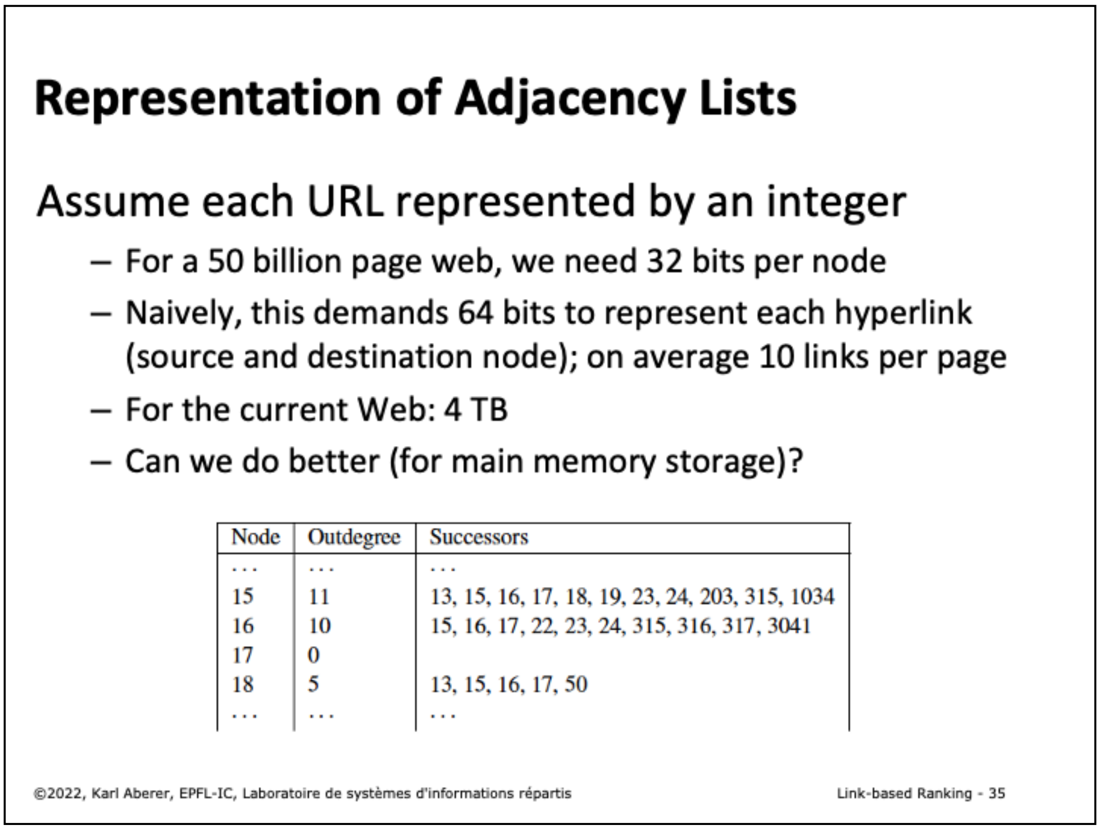
Adjacency Lists

The set of URLs a node is pointing to (or pointed to) sorted in *lexicographical order*

Example: outgoing links from www.epfl.ch

actu.epfl.ch/feeds/rss/mediacom/en/
bachelor.epfl.ch/studies
futuretudiant.epfl.ch/en
futuretudiant.epfl.ch/mobility
master.epfl.ch/page-94489-en.html
phd.epfl.ch/home
www.epfl.ch/navigate.en.shtml

A connectivity server has to store all outgoing (and incoming) links to a web page. For example, the home page of EPFL contains a large set of outgoing links, some of which are shown here. As a first step, the lists of links are sorted in lexicographical order. As a result, we obtain the adjacency list for a Web page, which we can consider as the equivalent to the posting list of a document in text indexing.



Representation of Adjacency Lists

Assume each URL represented by an integer

- For a 50 billion page web, we need 32 bits per node
- Naively, this demands 64 bits to represent each hyperlink (source and destination node); on average 10 links per page
- For the current Web: 4 TB
- Can we do better (for main memory storage)?

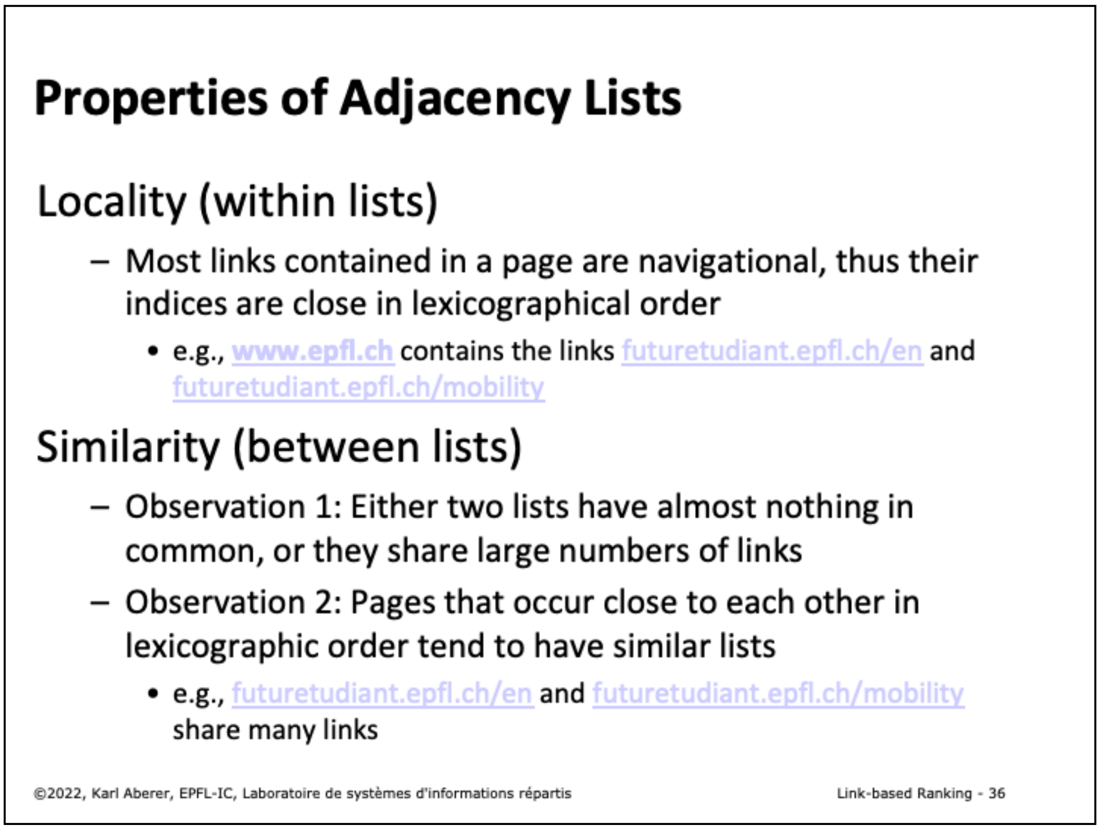
Node	Outdegree	Successors
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

As a first optimization of the representation of adjacency lists, we represent each URL by an integer, instead of storing it in its textual form. Using such an approach we can estimate the total size of adjacency lists for the current Web:

- The current (crawled) Web has around 50 billion pages (http://www.worldwidewebsize.com/, 2022)
- It is estimated that a page contains on average 10 links
- We need 32 bits for each URL, which demands 64 bits for the storage of a single link.

Therefore, the required storage is 4 TB.

Even with large memory sizes available today, this still is a significant index size. In the following, we will show how to reduce required storage to approximately ~3 bits/link which makes the size of the index much more manageable, i.e., about 20 times less storage. This will be achieved by systematically compressing the adjacency lists.



Properties of Adjacency Lists

Locality (within lists)

- Most links contained in a page are navigational, thus their indices are close in lexicographical order
  - e.g., www.epfl.ch contains the links futuretudiant.epfl.ch/en and futuretudiant.epfl.ch/mobility

Similarity (between lists)

- Observation 1: Either two lists have almost nothing in common, or they share large numbers of links
- Observation 2: Pages that occur close to each other in lexicographic order tend to have similar lists
  - e.g., futuretudiant.epfl.ch/en and futuretudiant.epfl.ch/mobility share many links

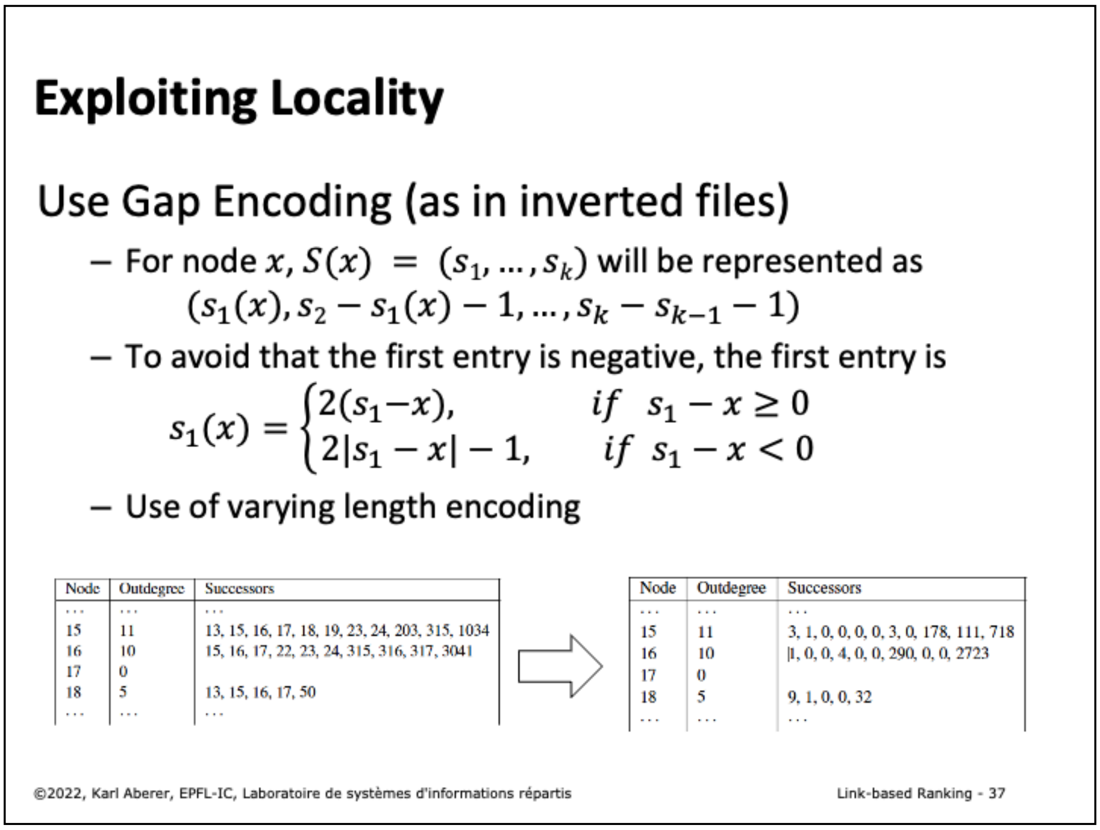
©2022, Karl Aberer, EPFL-IC, Laboratoire de systèmes d'informations répartis   Link-based Ranking - 36

For compressing adjacency lists we can exploit several observations on typical properties of Web pages.

Locality: Most links contained in a page are for navigating withing the same Web site. If we compare the source and target URLs of these links, we observe that as a result they often share a long common prefix. Thus, if URLs are sorted lexicographically, the index of the URL of a Web page and the URLs of the targets of the links of the Web page are close to each other. Locality is a property of one adjacency list, thus is an intra-list similarity property.

Similarity: In general, we can assume that either pages have many common links, because the belong to the same web site, or they have almost nothing in common, because they are from different Web sites. Furthermore, pages that are from the same Website will have URLs that are similar in lexicographical order, and therefore it is more likely to find pages with many common outgoing links close to each other in lexicographic order. Similarity is a property of different adjacency lists, this an inter-list similarity property.

We will now show of how to exploit these two properties.



Exploiting Locality

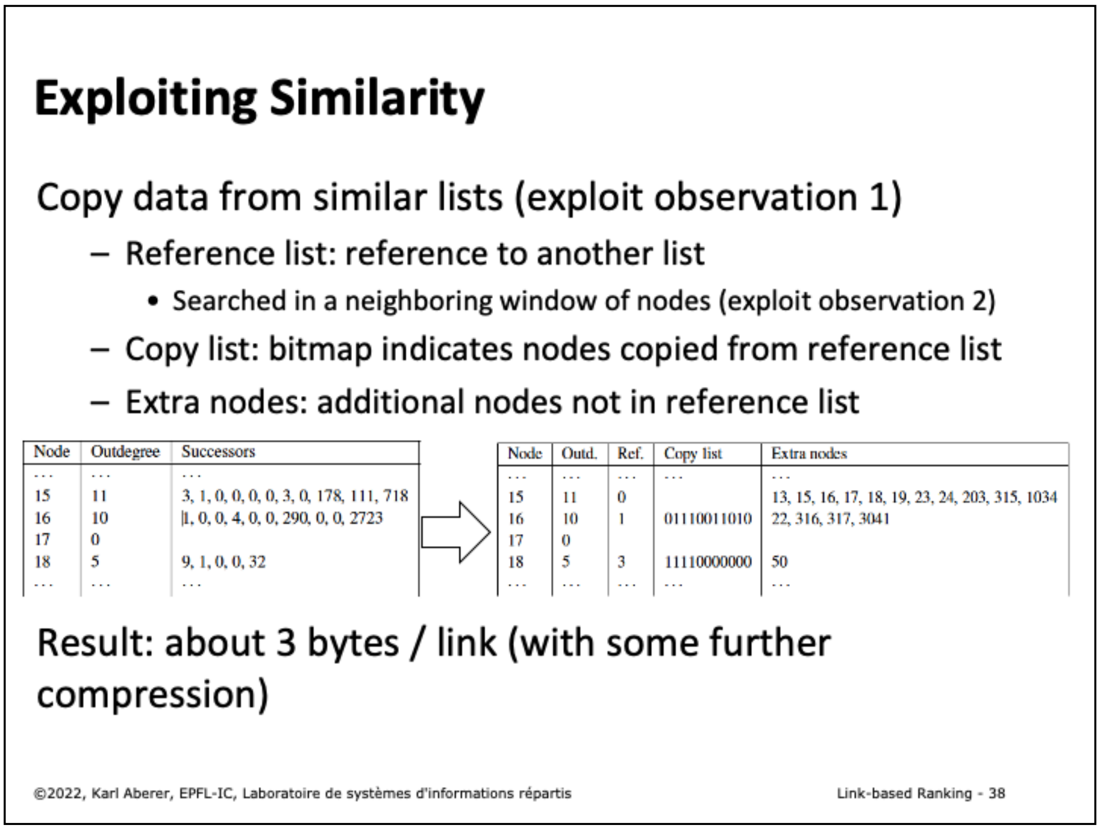
Use Gap Encoding (as in inverted files)

- For node x, $S(x) \;=\; (s_1, \dots, s_k)$ will be represented as
  $(s_1(x), s_2 - s_1(x) - 1, \dots, s_k - s_{k-1} - 1)$
- To avoid that the first entry is negative, the first entry is
  $$s_1(x) = \begin{cases} 2(s_1 - x), & if \;\; s_1 - x \geq 0 \\ 2|s_1 - x| - 1, & if \;\; s_1 - x < 0 \end{cases}$$
- Use of varying length encoding

Node	Outdegree	Successors
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

Node	Outdegree	Successors
...	...	...
15	11	3, 1, 0, 0, 0, 0, 3, 0, 178, 111, 718
16	10	1, 0, 0, 4, 0, 0, 290, 0, 0, 2723
17	0	
18	5	9, 1, 0, 0, 32
...	...	...

Locality can be exploited in a way analogous of how compression of posting lists for text indexing has been performed. Instead of storing the absolute integer indices of the URL identifiers, their differences are stored. In other words, we perform gap encoding. The resulting differences are then encoded using a varying length compression scheme, such as gamma coding, as it has been applied with inverted files.



Exploiting Similarity

Copy data from similar lists (exploit observation 1)

- Reference list: reference to another list
  - Searched in a neighboring window of nodes (exploit observation 2)
- Copy list: bitmap indicates nodes copied from reference list
- Extra nodes: additional nodes not in reference list

Node	Outdegree	Successors
...	...	...
15	11	3, 1, 0, 0, 0, 0, 3, 0, 178, 111, 718
16	10	1, 0, 0, 4, 0, 0, 290, 0, 0, 2723
17	0	
18	5	9, 1, 0, 0, 32
...	...	...

Node	Outd.	Ref.	Copy list	Extra nodes
...	...	...	...	...
15	11	0		13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	1	01110011010	22, 316, 317, 3041
17	0			
18	5	3	11110000000	50
...	...	...	...	...

Result: about 3 bytes / link (with some further compression)

For exploiting similarity, we exploit the redundancy among similar lists. Based on the observation that similar lists are more likely to occur in a lexicographical neighborhood, in a first step a sliding window of neighboring lists that have already been processed is searched for a most similar list. If such a list is found, it is called reference list. Then for the given adjacency list to be compressed, only the data necessary to reconstruct the list from the reference list will be stored. For this it is necessary to store two types of data:

1. Copy data: this is a bitlist that indicates for every entry in the reference list whether the URL is also part of the given adjacency list. This covers all URLs that the given list can “inherit” from the reference list
2. Extra nodes: the given list can also contain URLs that are not part of the reference list. For those the indices of the URLs need to be stored explicitly.

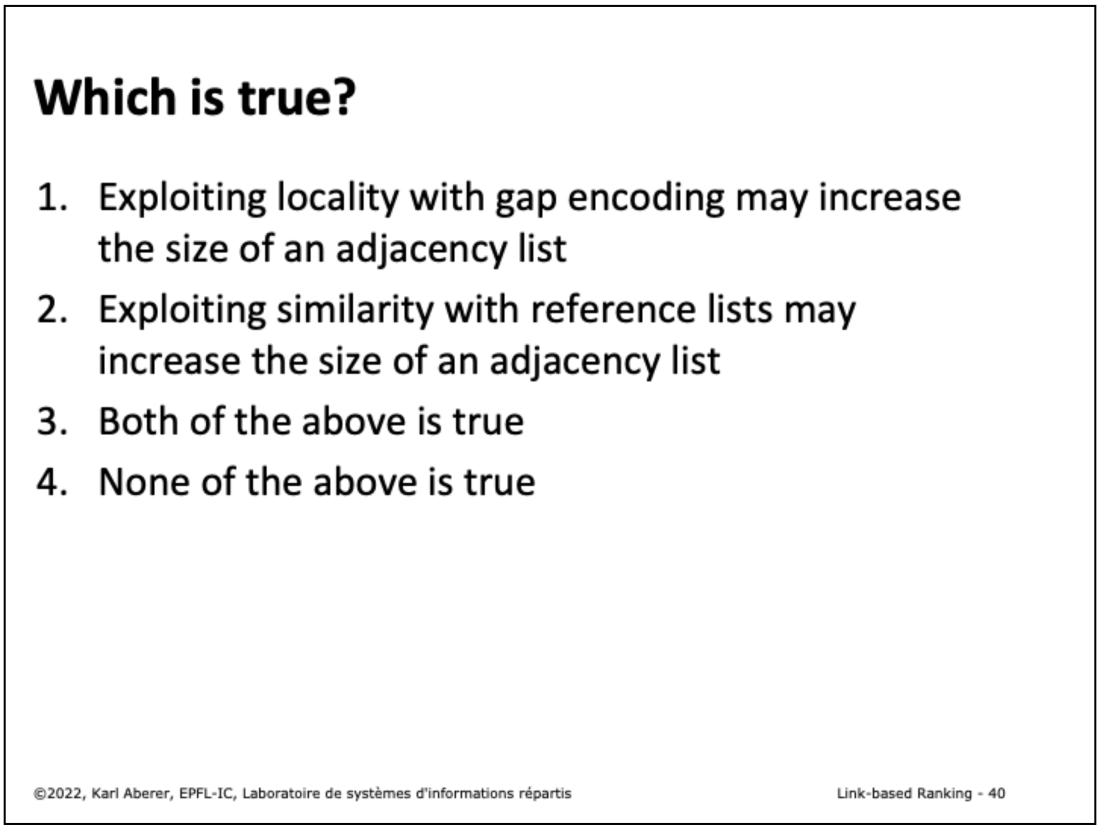
In the example we use Node 15 for the reference list and compress Node 16 and 18. By comparing the lists we see that for the case of Node 18 all, but one URL appear in the reference list of Node 15. There are indicated in the bitlist. The adjacency list of Node 18 contains also one URLs that does not appear in the reference list, namely 50. This is listed in the list of extra nodes.

Candidates for potential reference lists are searched among neighboring lists using a window of predefined size. The choice of the window size is important, as larger windows increase chances of finding good candidates, but also increase the cost of compression.

Together with some further compression applied to the copy lists and the extra nodes, this index compression scheme achieves about 3 Bytes/link cost in the representation of the Web graph.

When compressing the adjacency list of a given URL, a reference list

1. Is chosen from neighboring URLs that can be reached in a small number of hops
2. May contain URLs not occurring in the adjacency list of the given URL
3. Lists all URLs not contained in the adjacency list of given URL
4. All of the above



Which is true?

1. Exploiting locality with gap encoding may increase the size of an adjacency list
2. Exploiting similarity with reference lists may increase the size of an adjacency list
3. Both of the above is true
4. None of the above is true

References

Course material based on

- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008 (http://www-nlp.stanford.edu/IR-book/)

Relevant articles

- Sergey Brin , Lawrence Page, The anatomy of a large-scale hypertextual Web search engine, Computer Networks and ISDN Systems, v.30 n.1-7, p.107-117, April 1, 1998.
- Jon M. Kleinberg: Authoritative Sources in a Hyperlinked Environment. JACM 46(5): 604-632 (1999)
- Boldi, Paolo, and Sebastiano Vigna. "The webgraph framework I: compression techniques." Proceedings of the 13th international conference on World Wide Web. ACM, 2004.
- Jimmy Lin and Chris Dyer. Data-Intensive Text Processing with MapReduce, Morgan & Claypool Synthesis, 2011.