



# Suche im Web und Ranking

Tobias Scheffer

Uwe Dick

Paul Prasse

Michael Großhans

# World Wide Web

- 1990 am CERN von Tim Berners Lee zum besseren Zugriff auf Papers entwickelt.
  - ◆ HTTP, URLs, HTML, Webserver.
- Verbindet FTP mit der Idee von Hypertext.
- Multilingual (ca. 75% englisch, je 5% japanisch und deutsch).
- Groß, verteilt.
- Volatil: Dokumente erscheinen und verschwinden.
- Unstrukturiert, heterogen.

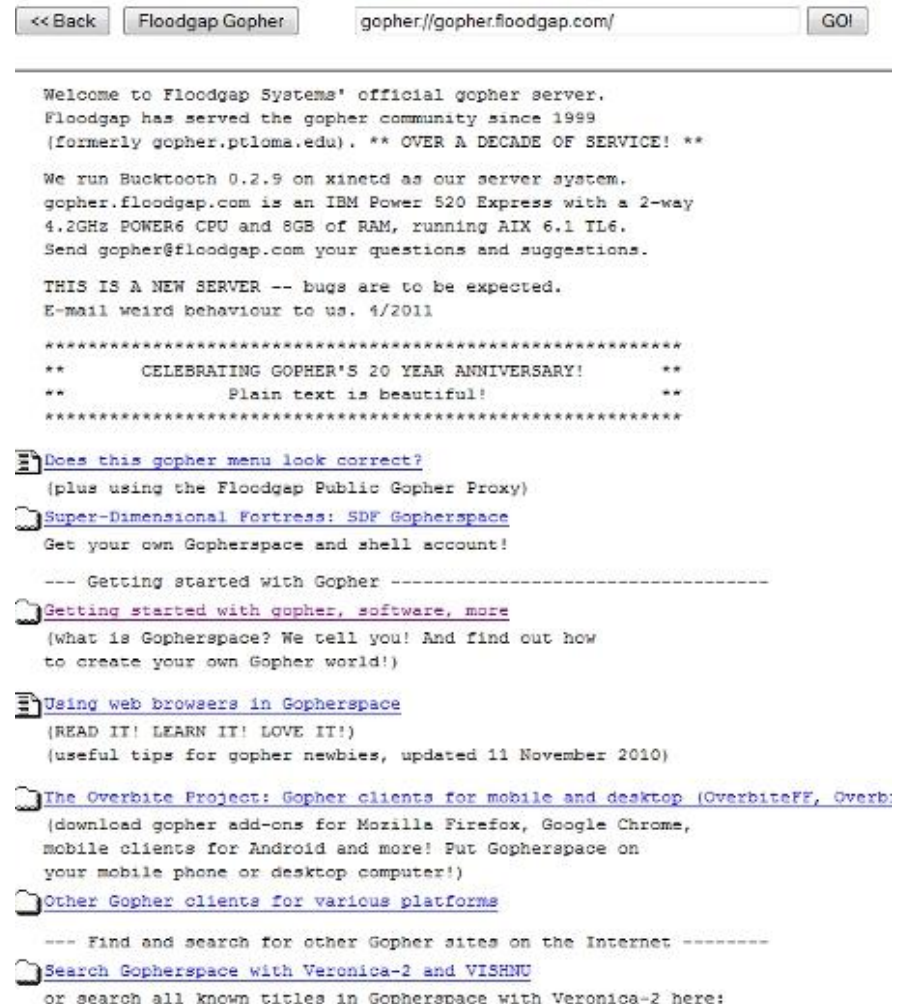
# World Wide Web

## Entwicklung

- Hypertext: 1960er Jahre.
- ARPANET / DARPANET / Internet: 1970er.
- FTP: 80er Jahre.
  - ◆ Archie: Crawler sammelt Dateinamen von FTP-Servern und ermöglicht Regex-Suche.
    - ★ Suchmaschine, die speziell zum Indizieren von FTP-Archiven entwickelt wurde.
    - ★ Problem: Da nur die Datei- und Verzeichnisnamen für die Indizierung verwendet werden, sind die Suchanfragen auf diesen Index beschränkt.
  - ◆ Gopher.

# World Wide Web Entwicklung - Gopher

- Ähnelt dem frühen WWW.
- Wurde an der Universität von Minnesota von Mark P. McCahill entwickelt.
- **Idee:** Umständliche Handhabung des FTP-Protokolls umgehen; Schaffung eines einfach zu administrierenden Informationssystems.
  - ◆ Verzeichniswechsel durch Konsolenbefehle, um gewünschte Dateien herunter laden zu können.



```
<< Back      Floodgap Gopher      gopher://gopher.floodgap.com/      GO!

Welcome to Floodgap Systems' official gopher server.
Floodgap has served the gopher community since 1999
(formerly gopher.ptloma.edu).  ** OVER A DECADE OF SERVICE! **

We run Bucktooth 0.2.9 on xinetd as our server system.
gopher.floodgap.com is an IBM Power 520 Express with a 2-way
4.2GHz POWER6 CPU and 8GB of RAM, running AIX 6.1 TL6.
Send gopher@floodgap.com your questions and suggestions.

THIS IS A NEW SERVER -- bugs are to be expected.
E-mail weird behaviour to us.  4/2011

*****
**          CELEBRATING GOPHER'S 20 YEAR ANNIVERSARY!          **
**          Plain text is beautiful!                          **
*****

[+] Does this gopher menu look correct?
    (plus using the Floodgap Public Gopher Proxy)
[-] Super-Dimensional Fortress: SDF Gopherspace
    Get your own Gopherspace and shell account!

    --- Getting started with Gopher -----
[-] Getting started with gopher, software, more
    (what is Gopherspace? We tell you! And find out how
    to create your own Gopher world!)
[-] Using web browsers in Gopherspace
    (READ IT! LEARN IT! LOVE IT!)
    (useful tips for gopher newbies, updated 11 November 2010)
[-] The Overbite Project: Gopher clients for mobile and desktop \(OverbiteFF, Overb:
    (download gopher add-ons for Mozilla Firefox, Google Chrome,
    mobile clients for Android and more! Put Gopherspace on
    your mobile phone or desktop computer!)
[-] Other Gopher clients for various platforms

    --- Find and search for other Gopher sites on the Internet -----
[-] Search Gopherspace with Veronica-2 and VISHNU
    or search all known titles in Gopherspace with Veronica-2 here:
```

# World Wide Web

## Entwicklung

- Web-Browser: Anfang der 90er von Doktoranden.
  - ◆ Mosaic: Marc Andreessen & Eric Bina.
  - ◆ Netscape und IE aus Mosaic hervorgegangen.
- Web-Directories: Verzeichnis „Yahoo“, 1994, zwei Stanford-Doktoranden.
  - ◆ Bilden Einstiegspunkt, um im Web zu surfen.
  - ◆ Meist kombiniert mit Suchmaschine.

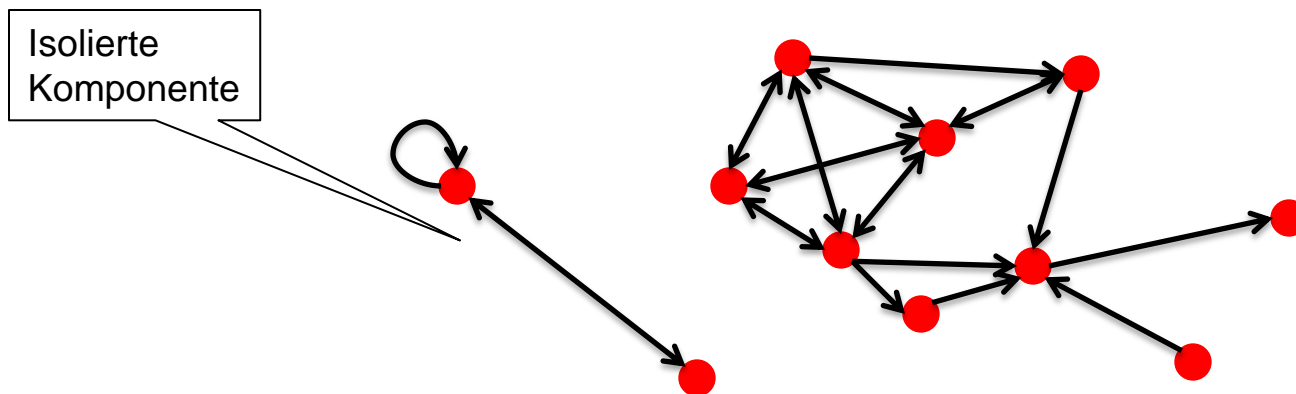
# World Wide Web

## Suche

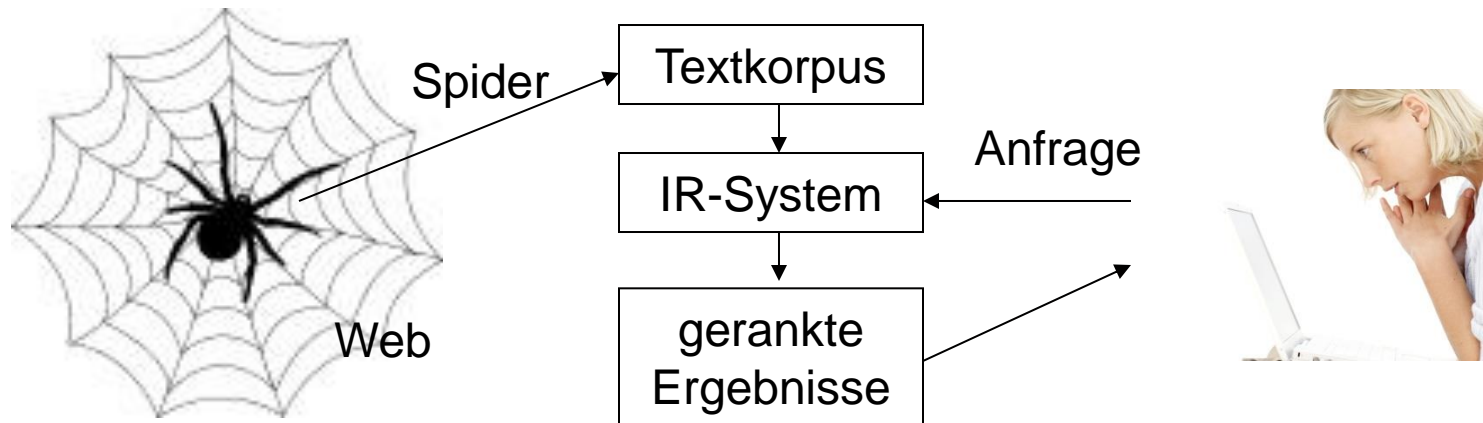
- WebCrawler: Programmier-Projekt, Uni Washington, 1994. → Excite, AOL.
- 1994: Doktorand an CMU entwickelt Lycos.
- 1995: DEC entwickelt Altavista.
- 1998: Doktoranden aus Stanford entwickeln Google.
  - ◆ Idee:
    - ★ Analyse der Linkstruktur;
    - ★ Relevantere Ergebnisse extrahieren.

# World Wide Web Graph-Struktur

- Große zentrale, stark verknüpfte Komponente.
  - ◆ Jede Seite von jeder Seite erreichbar.
- Randgebiete die nur
  - ◆ Auf die zentrale Komponente verweisen;
  - ◆ Von zentraler Komponente aus verwiesen werden.
- Isolierte Komponenten.



# World Wide Web Suche



- Unterschied zu anderen IR-Systemen:
  - ◆ Spider / Crawler sammelt Texte für Indexstruktur.
  - ◆ Analyse der Linkstruktur liefert Relevanzinformation.
  - ◆ Suchmaschinen sind stark verteilte Systeme.



# Crawler

- **Crawling:** Sammeln von Internetseiten, um sie indexieren zu können.
- **Aufgaben:**
  - ◆ Schnell neue Internetseiten besuchen.
  - ◆ Effizient arbeiten.
    - ★ So viele brauchbare Internetseiten wie möglich in kurzer Zeit bearbeiten;
    - ★ Relevante Informationen extrahieren und speichern.
- Implementierung von lauffähigen Crawlern sind im Netz frei verfügbar.

# Crawler Eigenschaften

- **Stabilität:** Crawler muss in der Lage sein, Spidertraps zu umgehen.
  - ◆ Zyklen, dynamische Internetseiten.
- **Fair:** Crawler darf eine Seite nicht beliebig oft in kurzen Zeitabständen besuchen.
  - ◆ „Denial of Service Attack“.
- **Verteilt:** Crawler sollte auf mehreren Maschinen verteilt sein.
  - ◆ Mehrere Threads laufen parallel.
- **Skalierbar:** Crawler sollte erweiterbar sein.
  - ◆ Mehrere Maschinen, höhere Bandbreite.

# Crawler

## Eigenschaften

- **Effizienz:** Intelligente Nutzung des verfügbaren Speichers, der Bandbreite und des Prozessors.
  - ◆ Möglichst wenige Prozesse die „idle“ sind.
- **Qualität:** Die wichtigsten Seiten sollten zuerst besucht werden.
  - ◆ Internetseiten mit hohem Informationsgehalt wichtiger als private Internetseiten.
- **Aktualität:** Der Crawler sollte das Internet kontinuierlich crawlen.
  - ◆ Die Frequenz, mit der eine Internetseite besucht wird, sollte sich an ihrer Änderungsfrequenz orientieren.
- **Erweiterbar:** Sollte um neue Dateiformate erweitert werden können (XML,...).

# Crawler Suchstrategien

- **Strategien:**
  - ◆ Breadth first,
  - ◆ Depth first,
  - ◆ Hoher PageRank first.
- Crawlhäufigkeit in Abhängigkeit von:
  - ◆ Änderungshäufigkeit und PageRank (WallstreetJournal.com vs. HottesSockenShop.de).
  - ◆ Der Zugriffshäufigkeit auf die Seite.
- Crawling zu den richtigen Zeitpunkten bei zyklischen Updates.

# Crawler

## Arbeitsweise

### Algorithmus:

1. Beginne mit einem Pool von URLs.
  2. Besuche die URLs des Pools und extrahiere den Text und die Links der besuchten URLs.
  3. Übergebe den Text einer URL an den Indexer und füge gefundene Links zum Pool hinzu.
  4. Füge bereits besuchte URLs wieder in den Pool ein, um sie später erneut besuchen zu können.
    - ◆ Inhalte von Internetseiten können sich ändern.
- Der Graph des Internets wird dabei mehrmals traversiert.

# Crawler

## Arbeitsweise

- Um eine Milliarde Internetseiten in einem Monat zu crawlen, müssen Hunderte Internetseiten pro Sekunde besucht werden.
- Dabei ist zu beachten:
  - ◆ Links, die während des Crawlens gefunden wurden, können relative Links sein.
    - ★ Normalisierung aller relativen URLs.
  - ◆ Links können mehrmals auftauchen.
    - ★ Duplikaterkennung.
  - ◆ Links können auf Webseiten verweisen, die nicht automatisch besucht werden dürfen.
    - ★ robot.txt

# Crawler Architektur

- **URL Frontier:** Liste von URLs die besucht werden sollen.
- **DNS-Auflösung:** Bestimmung der IP-Adresse für eine gegebene URL.
- **Fetching-Modul:** Lädt Internetseiten herunter, damit sie bearbeitet werden können.
- **Parsing-Modul:** Extrahiert Text und Links einer Internetseite.
- **Dublikaterkennung:** Erkennt URLs und Inhalte, die bereits kurze Zeit vorher bearbeitet wurden.

# Crawler Architektur

- Beim Parsen einer Internetseite werden die Ankertexte als Indexterme für die verlinkte Seite abgelegt.

```
<a href="http://example.com/">Ankertext</a>
```

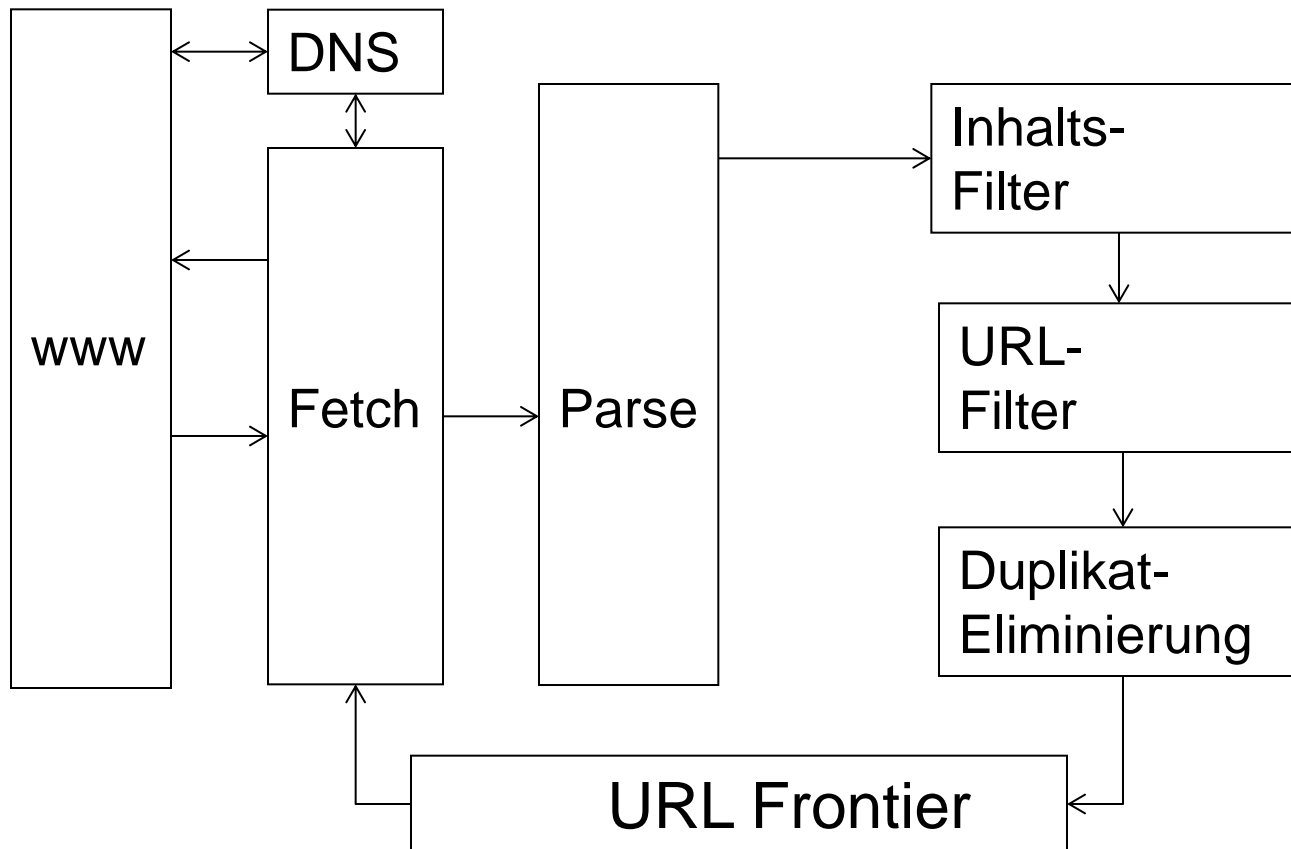
- Bevor ein Link zur URL Frontier hinzugefügt wird, muss überprüft werden:
  - ◆ Ob diese URL bereits vorhanden ist.
  - ◆ Ob die Internetseite ein inhaltliches Duplikat einer anderen Internetseite der URL Frontier ist.
  - ◆ Ob es erlaubt ist, die Internetseite zu crawlen (robot.txt).

Beispiel für robot.txt:

```
User-agent: mein-Robot  
Disallow: /quellen/dtd/  
  
User-agent: *  
Disallow: /unsinn/  
Disallow: /temp/  
Disallow: /newsticker.shtml
```



# Crawler Architektur



# Crawler

## URL Frontier

- **Anforderungen:**
  - ◆ Jede URL besitzt Score der angibt, wie relevant eine URL ist.
  - ◆ Es darf nur eine Verbindung zu einer Domain zur selben Zeit aufgebaut werden.
  - ◆ Zwischen zwei Anfragen auf eine Domain, muss eine bestimmte Zeit gewartet werden.
    - ★ „DOS-Attacken“.
  - ◆ Soll alle Threads mit URLs versorgen.
    - ★ Keine „idle“ Threads.

# Ranking

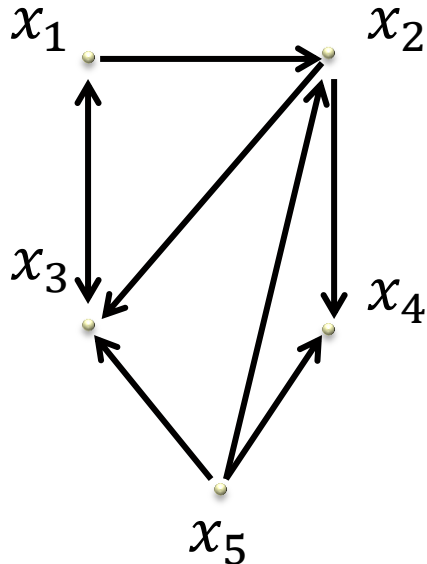
- Ende der 90er wurden alle Suchmaschinen in Web-Portale mit breitem Informations- und Unterhaltungsangebot umgebaut.
- Gegenannahme von Google: nur gute Suchfunktion ist wichtig, der Rest interessiert niemanden.
- Idee von Kleinberg (HITS) und Page & Brin (PageRank): Verweisstruktur zeigt, welche Webseiten irgend jemanden interessieren.
- Suchergebnisse werden nach PageRank bzw. HITS sortiert.

# Webgraph

- Darstellung des Webgraphen als Adjazenzmatrix

$A_{ij} = 1$  falls Link von Seite  $x_i$  nach  $x_j$

$A_{ij} = 0$  sonst



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

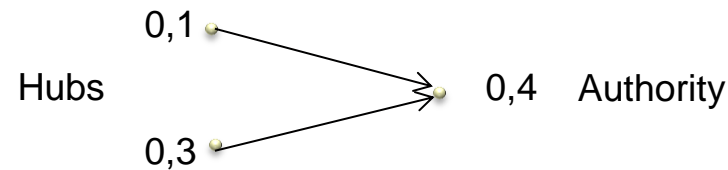
# Hubs & Authorities (HITS)

- Erst wird Menge der relevanten Dokumente bestimmt, z.B. durch invertierten Index.
- Diese werden mittels des Hub & Authority Score sortiert.
- Hubs & Authorities (HITS):
  - ◆ Jeder Knoten (Webseite) besitzt
    - ★ *Hub Score*: Wie gut sind seine Verweise auf „kompetente“ Knoten  
= ausgehende Kanten.
    - ★ *Authority Score*: Wie kompetent ist der Knoten, d.h. wie viele kompetente Knoten verweisen auf ihn  
= eingehende Kanten.

# Hubs & Authorities (HITS) Algorithmus

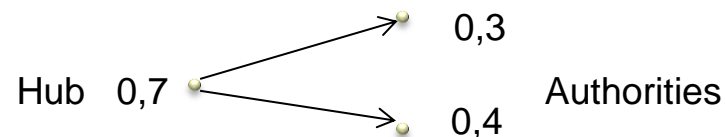
- Jeder Knoten  $x$  des Graphen hat einen Hub-Score  $h(x)$  und einen Authority-Score  $a(x)$
- **Authority-Update-Regel:** Für jeden Knoten  $x$ , berechne  $a(x)$  als Summe aller Hub-Scores von Knoten, die auf diesen zeigen.

- ◆  $a(x_i) = \sum_{j=1 \dots n} A_{i,j}^T h(x_j)$



- **Hub Update Regel:** Für jeden Knoten  $x$ , berechne  $h(x)$  als Summe der Authority-Scores aller Knoten zu denen er zeigt.

- ◆  $h(x_i) = \sum_{j=1 \dots n} A_{i,j} a(x_j)$

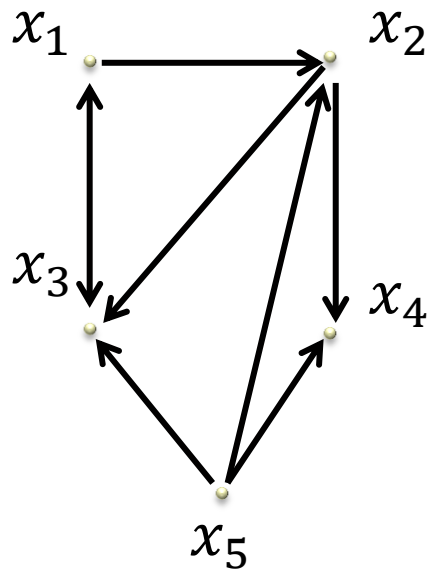


# Hubs & Authorities (HITS) Algorithmus

- Initialisiere für alle Knoten  $x$  des Graphen  $h(x) = 1$  und  $a(x) = 1$
- Iteriere  $k$  –mal:
  - ◆  $a(x_i) = \sum_{j=1\dots n} A_{i,j}^T h(x_j)$  für alle  $i$  (Authority-Update-Regel)
  - ◆  $a(x_i) = a(x_i) / \sqrt{\sum_j a(x_j)^2}$  für alle  $i$
  - ◆  $h(x_i) = \sum_{j=1\dots n} A_{i,j} a(x_j)$  für alle  $i$  (Hub-Update-Regel)
  - ◆  $h(x_i) = h(x_i) / \sqrt{\sum_j h(x_j)^2}$  für alle  $i$

# Hubs & Authorities (HITS)

■ Beispiel:



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Normalisierung

k	a					h				
	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
0	1	1	1	1	1	1	1	1	1	1
1	1	2	3	2	0	1,18	1,18	0,24	0	1,65
	0,23	0,47	0,71	0,47	0	0,5	0,5	0,1	0	0,7
2	0,1	1,2	1,7	1,2	0	1,21	1,21	0,04	0	1,71
	0,04	0,50	0,71	0,50	0	0,5	0,5	0,02	0	0,71
3	0,02	1,21	1,71	1,21	0	1,21	1,21	0,01	0	1,71
	0,01	0,50	0,71	0,50	0	0,5	0,5	0,003	0	0,71



# Hubs & Authorities (HITS)

- $a^{(t)}$  sei die Vektordarstellung der Authority-Scores (normalisiert) nach  $t$  Iterationen. Also  $a^{(t)}(i) = a(x_i)$  nach  $t$  Iterationen. Dann gilt:
  - ◆  $a^{(1)} = \frac{A^T \mathbf{1}}{\|A^T \mathbf{1}\|}$
  - ◆  $a^{(t)} = \frac{A^T A a^{(t-1)}}{\|A^T A a^{(t-1)}\|}$
- Der Algorithmus konvergiert gegen den Eigenvektor von  $A^T A$  mit größtem Eigenwert.
- Es gilt außerdem:
  - ◆  $h^{(1)} = \frac{A A^T \mathbf{1}}{\|A A^T \mathbf{1}\|}, h^{(t)} = \frac{A A^T h^{(t-1)}}{\|A A^T h^{(t-1)}\|}$
  - ◆  $h^{(t)} = \frac{A a^{(t-1)}}{\|A a^{(t-1)}\|}$

# Authority-Ranking

## PageRank

- Keine Unterscheidung in Hubs und Authorities.
- Ranking der Seiten nur nach Authority.
- PageRank ist umso höher, je mehr andere Seiten auf die Seiten verweisen, und je höher deren PageRank ist.
- Wird auf das ganze Web angewandt, nicht nur auf Ergebnismenge einer Suchmaschine (HITS).

# Authority-Ranking

## PageRank

- Random Surfer:
  - ◆ Beginnend bei einem beliebigen Knoten folgt der Surfer mit Wahrscheinlichkeit  $1-\varepsilon$  einem zufälligen Link.
  - ◆ Mit Wahrscheinlichkeit  $\varepsilon$  startet er neu an Zufallsknoten.
  - ◆ Wahrscheinlichkeit für Aufenthalt an einem Knoten = globaler Authority Score.

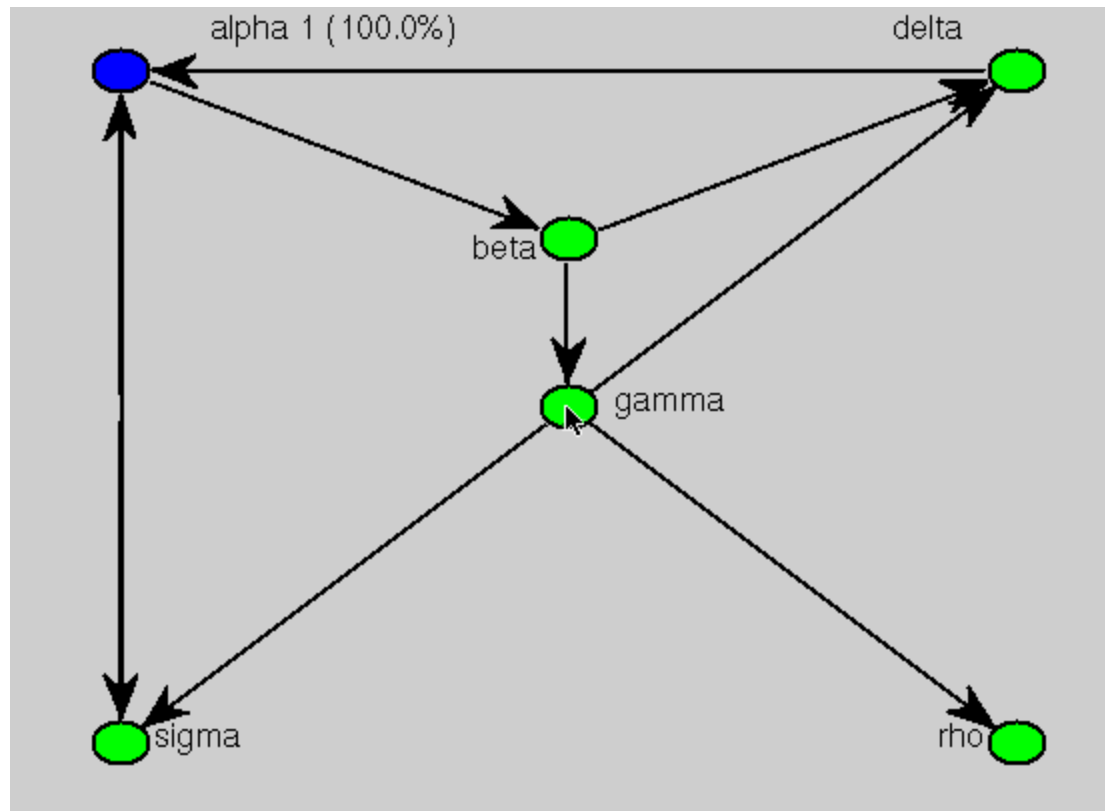
# Authority-Ranking

## PageRank

- **PageRank:** Random Surfer Modell zum Ranking von Webseiten.
  - ◆ (Ursprünglicher) Ranking-Algorithmus von Google.
  - ◆ Abhängig von Query werden relevante Webseiten gefunden und nach ihrem globalen Authority Score sortiert.
- **Annahmen:**
  - ◆ Link auf Webseite  $x_i$  verweist auf „kompetente“ Webseite  $x_j$ , d.h. Adjazenzmatrix des Webgraphen = Authority Matrix mit  $A_{ij} = 1$ .
  - ◆ Mit Wahrscheinlichkeit  $1 - \varepsilon$  folgt der Nutzer (Random Surfer) einem Link auf der Webseite.
  - ◆ Mit Wahrscheinlichkeit  $\varepsilon$  wechselt er auf eine zufällige Webseite.

# Authority-Ranking PageRank

- Beispiel:



# Authority-Ranking

## PageRank - Formal

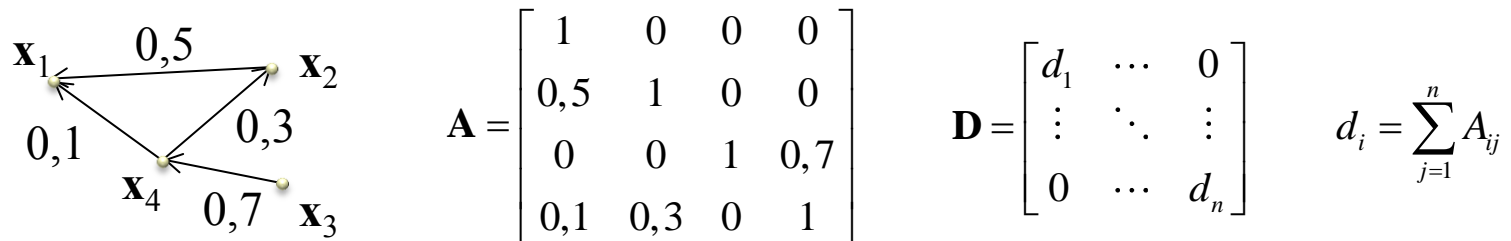
- **Gegeben:** Instanzen (Webseiten)  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  mit gegebenen lokalen Authority Scores (Kompetenz-Bewertungen):
  - ◆ Authority Score  $A_{ij} = a(\mathbf{x}_i, \mathbf{x}_j)$  gibt an wie „kompetent“  $\mathbf{x}_j$  aus Sicht von  $\mathbf{x}_i$  ist.
  - ◆ Link oder kein Link, Link-Position.
- **Gesucht:** Modell  $f : A \in R^{n \times n} \mapsto s \in R^n$  welches für Instanzen  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  globale Authority Scores (Ranking)  $s_i$  liefert.
  - Annahme: Je kompetenter  $\mathbf{x}_i$  und je höher der Authority Score  $A_{ij}$ , desto kompetenter  $\mathbf{x}_j$ .

Authority Matrix

# Authority-Ranking

## PageRank - Modellierung

- Modellierung der (nicht-symmetrischen) Kompetenz-Bewertungen als gerichteter Graph:
  - ◆ Instanzen sind Knoten, Authority Scores sind Kanten-Gewichte  
⇒ Authority Matrix = Adjazenzmatrix  $\mathbf{A}$ .



- ◆ Intuition: (normierter) Graph beschreibt mit welcher Wahrscheinlichkeit Knoten  $x_i$  Knoten  $x_j$  als „Experten“ nennen würde.
- ◆ Beispiel:  $x_4$  hält  $x_2$  für 3-mal so kompetent wie  $x_1$ ,  $x_3$  ist aus seiner Sicht kein Experte.

# Authority-Ranking

## PageRank

- **Gegeben:** Wahrscheinlichkeit dafür, dass der Nutzer von Webseite  $x_i$  zu Webseite  $x_j$  wechselt ist

$$P_{ij} = p(\mathbf{x}_j | \mathbf{x}_i) = (1 - \varepsilon) \frac{A_{ij}}{\sum_{k=1}^n A_{ik}} + \varepsilon \frac{1}{n}$$

Transitionswahrscheinlichkeit

und somit  $\mathbf{P} = (1 - \varepsilon)\mathbf{D}^{-1}\mathbf{A} + \varepsilon\mathbf{U}$  mit  $U_{ij} = \frac{1}{n}$ .

- **Gesucht:** Wahrscheinlichkeit dafür, dass man auf Webseite  $x_i$  ist, d.h.  $s_i = p(\mathbf{x}_i)$ .

Aufenthaltswahrscheinlichkeit



# Authority-Ranking

## PageRank

- **Algorithmus:** Beginnend mit initialem Ranking Scores  $\mathbf{s}$  iterativ neue Scores bestimmen mit

$$\mathbf{s}' = \frac{1}{c} \mathbf{P}^T \mathbf{s} \quad \text{wobei} \quad c = \|\mathbf{P}^T \mathbf{s}\| \Rightarrow \|\mathbf{s}'\| = 1$$

- Konvergenz von PageRank bei  $\mathbf{s}' = \mathbf{s}$ , sodass gilt

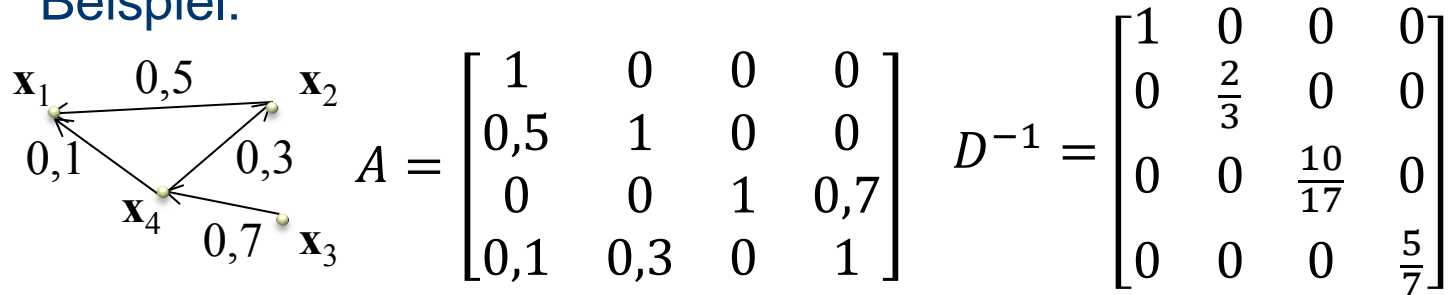
$$\mathbf{s} = \frac{1}{\lambda} \mathbf{P}^T \mathbf{s}$$

d.h.  $\mathbf{s}$  ist ein Eigenvektor von  $\mathbf{P}^T$  mit Eigenwert  $\lambda$ .

- Man kann zeigen, dass  $\mathbf{s}$  der Eigenvektor mit größtem Eigenwert  $\lambda$  ist.

# Authority-Ranking PageRank

■ Beispiel:



$$P = (1 - \varepsilon)D^{-1}A + \varepsilon U = \begin{bmatrix} \frac{397}{400} & \frac{1}{400} & \frac{1}{400} & \frac{1}{400} \\ \frac{133}{400} & \frac{53}{80} & \frac{1}{400} & \frac{1}{400} \\ \frac{1}{400} & \frac{1}{400} & \frac{417}{713} & \frac{865}{2109} \\ \frac{41}{560} & \frac{601}{2800} & \frac{1}{400} & \frac{655}{923} \end{bmatrix}, \text{ mit } \varepsilon = 0.01$$

$$s' = \frac{1}{\|P^T s\|} P^T s = \begin{bmatrix} 0,6711 \\ 0,4227 \\ 0,2838 \\ 0,5389 \end{bmatrix} \xrightarrow{\text{rekursiv einsetzen}} \begin{bmatrix} 0,997 \\ 0,0186 \\ 0,0062 \\ 0,0176 \end{bmatrix}, \text{ mit } s_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

# Authority-Ranking

## PageRank

### ■ Vorteile:

- ◆ Leicht und effizient berechenbar.
- ◆ Existenz & Eindeutigkeit der Lösung sowie Konvergenz des Algorithmus ist garantiert für  $0 < \varepsilon < 1$ .

### ■ Nachteile:

- ◆ Links können schlechte Indikatoren für Kompetenz-Bewertung sein:
  - ★ Kompetenz zweier Instanzen kann sich gegenseitig verstärken.
  - ★ Automatisch generierte Links haben kaum Aussagekraft.
  - ★ „Künstliche“ Links (z.B. Link-Spam) verändern Ranking.
- ◆ Eigenschaften der Instanzen fließen nicht ins Ranking ein.

# PageRank

## Link Spam

- PageRank ist beeinflussbar.
- Link Farmen:
  - ◆ Felder künstlich generierter Seiten, deren PageRank „geerntet“ und zu Zielseite geleitet wird.
- Guestbook Spam:
  - ◆ Generierte Einträge in Gästebüchern und Blogs, die Verweis auf Zielseite enthalten.
- Link Exchange Services:
  - ◆ Seiten mit Links auf (thematisch nicht verwandte) Seiten, meist gegen Geld.
- Partner-Programme: z.B. Amazon, Ebay.
  - ◆ Link auf Produktseite gegen Umsatzbeteiligung.

# Link Spam

## BadRank

- Wenn PageRank beeinflussbar wird, dann verliert er seine Korrelation zur Relevanz der Seiten.
- Link Spam sollte bei der Berechnung des Page Ranks nicht so berücksichtigt werden wie „natürliche“ Links.
- Suchmaschinenbetreiber haben „Blacklists“.
  - ◆ URLs von Link-Spam-Seiten.
  - ◆ Werden manuell erstellt.

# Link Spam

## BadRank

- Invertierter PageRank-Algorithmus, „bestraft“ Seiten, die auf Link Spam verweisen.
- Initialisierung:  $B(u) = 1$ , wenn  $u$  auf Blacklist.
- Für alle Seiten:

$$B'(u) = \sum_{v:u \rightarrow v} \frac{B(v)}{N_u}$$

Gemittelter BadRank der Links eines Knoten

$$B(u) = \frac{B'(u)}{|B'(u)|}$$

Normalisierung

- BadRank wirkt wie negativer PageRank.

# World Wide Web

## Benutzerschnittstellen

- Advanced Search:
  - ◆ Wird fast nie benutzt.
  - ◆ Ähnliche Seiten finden,
  - ◆ Links auf Seite finden,
  - ◆ Maschinelle Übersetzung, cross-language retrieval.
- Clusterung von Seiten

# World Wide Web

## Retrieval - Websuche

- Suchterme  $\rightarrow$  Index  $\rightarrow$  Liste von URLs.
- Texte mit mehreren Suchtermen durch Mengenoperationen.
- Aufbau eines Ähnlichkeitsvektors mit verschiedenen Merkmalen

$$\Phi(x, q) = \begin{pmatrix} \text{sim}(x, q) \text{ im Vektorraummodell} \\ \# \text{ gleicher Terme in } \langle \text{h1} \rangle \text{-Tags} \\ \dots \\ \text{PageRank}(x) \end{pmatrix}$$

- Rückgabe der Seiten  $\arg \max_x w\Phi(x, q)$
- Manuelle Konstruktion von  $w$ .
- Lernen von  $w$  aus Klickdaten.



# RankSVM

- **Idee:** Lernen des Gewichtsvektors durch Klickdaten.

## 1. SVM-Support Vector Machines

[www.support-vector-machines.org/](http://www.support-vector-machines.org/)

## 2. SVM-Light Support Vector Machine

[svmlight.joachims.org/](http://svmlight.joachims.org/)

## 3. Kernel-Machines.Org — Kernel Machines

[www.kernel-machines.org/](http://www.kernel-machines.org/)

## 4. Support Vector Machines - The Book

[www.support-vector.net/](http://www.support-vector.net/)

## 5. Support Vector Machines

[www.svms.org/](http://www.svms.org/)

## 6. SVM - Support Vector Machines

[www.dtreg.com/svm.htm](http://www.dtreg.com/svm.htm)



$$l_1 > l_2$$

$$l_1 > l_3$$

$$l_1 > l_5$$

$$l_1 > l_6$$

$$l_4 > l_2$$

$$l_4 > l_3$$

$$l_4 > l_5$$

$$l_4 > l_6$$

# RankSVM

## ■ Eingabe:

- ◆ Paare von Suchanfragen mit partiellem **Ranking**:

- ★  $(q_1, r'_1), (q_2, r'_2), \dots, (q_n, r'_n)$

- ★  $r'_i = \{(d_1, d_2), (d_1, d_3), \dots, (d_j, d_k)\}$

$$\begin{array}{l} d_1 > d_2 \\ d_1 > d_3 \\ \vdots \\ d_j > d_k \end{array}$$

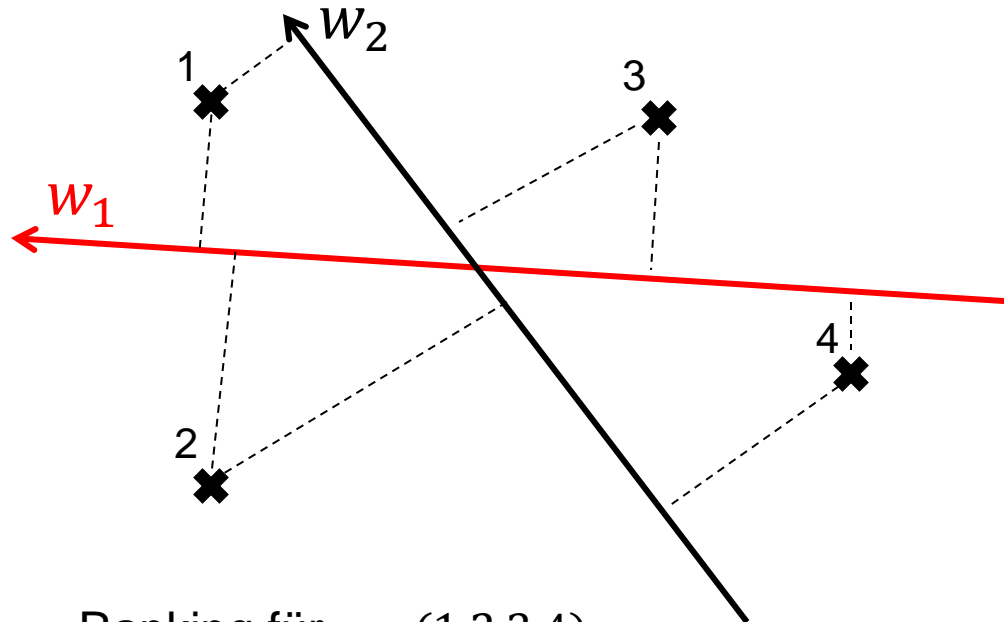
- ## ■ Ziel: Gewichtsvektor $w$ , so dass für alle Relationen $(d_i, d_j)$ einer Anfrage $q$ gilt:

- ◆  $w^T \Phi(d_i, q) > w^T \Phi(d_j, q)$

$$\begin{array}{l} w^T \Phi(d_1, q_i) > w^T \Phi(d_2, q_i) \\ w^T \Phi(d_1, q_i) > w^T \Phi(d_3, q_i) \\ \vdots \\ w^T \Phi(d_j, q_i) > w^T \Phi(d_k, q_i) \end{array}$$

# RankSVM

- Beispiel:



Ranking für  $w_1$ : (1,2,3,4)

Ranking für  $w_2$ : (1,3,2,4)

# RankSVM

- RankSVM Optimierungsproblem:

Minimiere:  $\frac{1}{2} \|w\|^2 + C \sum \xi_{i,j,k}$

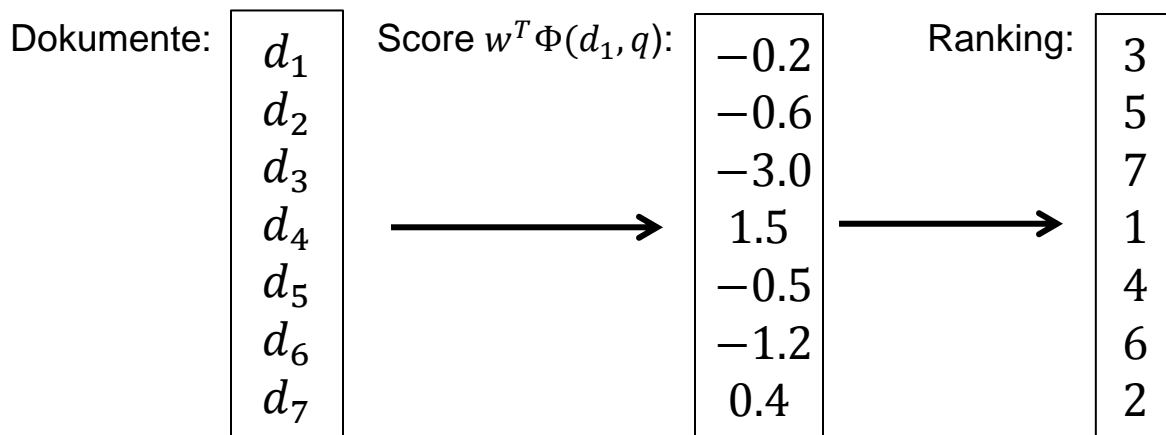
so, dass:  $\forall (d_i, d_j) \in r'_1: w^T \Phi(d_i, q_1) > w^T \Phi(d_j, q_1) + 1 - \xi_{i,j,1}$

⋮

$\forall (d_i, d_j) \in r'_k: w^T \Phi(d_i, q_k) > w^T \Phi(d_j, q_k) + 1 - \xi_{i,j,k}$

$\forall i \forall j \forall k: \xi_{i,j,k} \geq 0$

- Bestimmung des optimalen Rankings für Anfrage  $q$ :



# RankSVM Evaluation

- Kendall's  $\tau$ : Bestimmt den Grad der Übereinstimmung zweier Rankings.

- $$\tau = \frac{2P}{\frac{1}{2}n(n-1)} - 1$$

Anzahl von Übereinstimmungen

Anzahl von Relationen

- $\tau$  kann Werte von -1 (keine Übereinstimmung) bis +1 (perfekte Übereinstimmung) annehmen.

- Beispiel:

- ◆ Ranking 1: A,B,C,D

*A > B; A > C; A > D; B > C; B > D; C > D*

- ◆ Ranking 2: B,C,A,D

*B > C; B > A; B > D; C > A; C > D; A > D*

- ◆ 
$$\tau = \frac{2*4}{\frac{1}{2}4(4-1)} - 1 = \frac{1}{3}$$

# Browsing Webverzeichnisse

- Dmoz.org,
- Yahoo,
- Google.

**dmoz** In partnership with **AOL.**

Follow @dmoz | [about dmoz](#) | [dmoz blog](#) | [suggest URL](#) | [help](#) | [link](#) | [editor login](#)

Search *advanced*

**Arts**  
[Movies](#), [Television](#), [Music](#)...

**Business**  
[Jobs](#), [Real Estate](#), [Investing](#)...

**Computers**  
[Internet](#), [Software](#), [Hardware](#)...

**Games**  
[Video Games](#), [RPGs](#), [Gambling](#)...

**Health**  
[Fitness](#), [Medicine](#), [Alternative](#)...

**Home**  
[Family](#), [Consumers](#), [Cooking](#)...

**Kids and Teens**  
[Arts](#), [School Time](#), [Teen Life](#)...

**News**  
[Media](#), [Newspapers](#), [Weather](#)...

**Recreation**  
[Travel](#), [Food](#), [Outdoors](#), [Humor](#)...

**Reference**  
[Maps](#), [Education](#), [Libraries](#)...

**Regional**  
[US](#), [Canada](#), [UK](#), [Europe](#)...

**Science**  
[Biology](#), [Psychology](#), [Physics](#)...

**Shopping**  
[Clothing](#), [Food](#), [Gifts](#)...

**Society**  
[People](#), [Religion](#), [Issues](#)...

**Sports**  
[Baseball](#), [Soccer](#), [Basketball](#)...

**World**  
[Català](#), [Česky](#), [Dansk](#), [Deutsch](#), [Español](#), [Esperanto](#), [Français](#), [Galego](#), [Hrvatski](#), [Italiano](#), [Lietuvių](#),  
[Magyar](#), [Nederlands](#), [Norsk](#), [Polski](#), [Português](#), [Română](#), [Slovensky](#), [Suomi](#), [Svenska](#), [Türkçe](#),  
[Български](#), [Ελληνικά](#), [Русский](#), [Українська](#), [العربية](#), [עברית](#), [עברית](#), [עברית](#), [日本語](#), [简体中文](#), [繁體中文](#), ...

**Become an Editor** Help build the largest human-edited directory of the web

Copyright © 1998-2015 AOL Inc.

4,055,069 sites - 90,374 editors - over 1,024,763 categories

Build 2.1.10-783858 Wed May 27 23:54:09 EDT 2015

# Fragen?