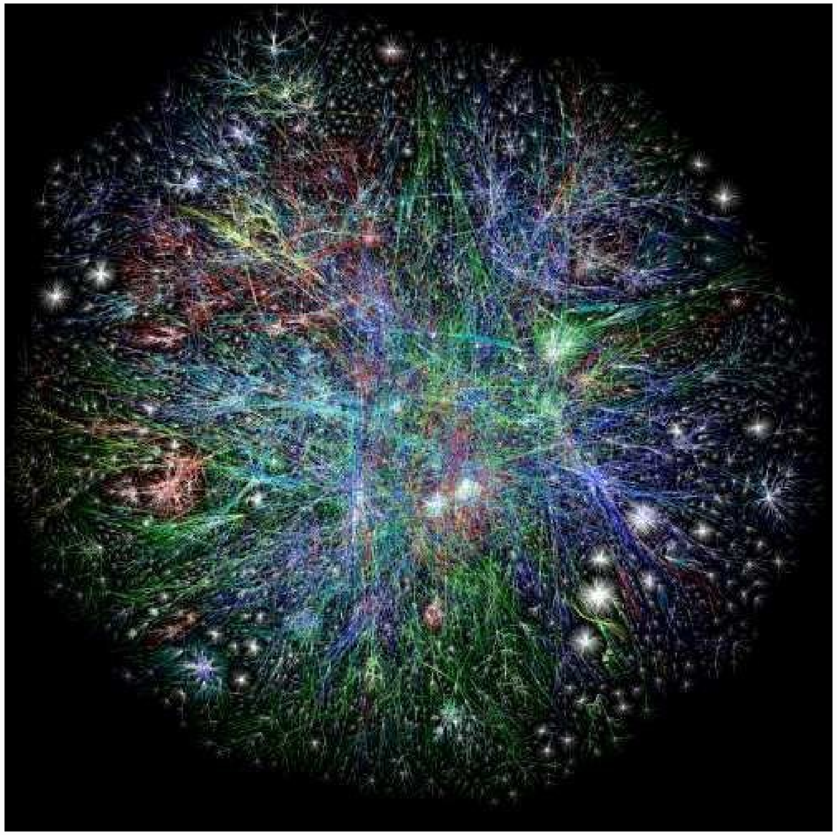
Mining graph data: Web and recommender systems



1. Introduction to Graph mining
2. Web mining and search
   - Emphasis: **Ranking algorithms**
3. Recommender systems
   - Emphasis: **Collaborative filtering**

image source: Opte project, `https://www.opte.org/the-internet`

1. Introduction to Graph mining

Data may consist of

1. one large graph
   - e.g., internet, social network
2. multiple small graphs
   - e.g., chemical compounds, biological pathways, program control flows, consumer behaviour, ...

Information to mine: interesting substructures, influential nodes, similarities, communities, clusters

Data: one large graph

Internet (Wikipedia hyperlink structure)

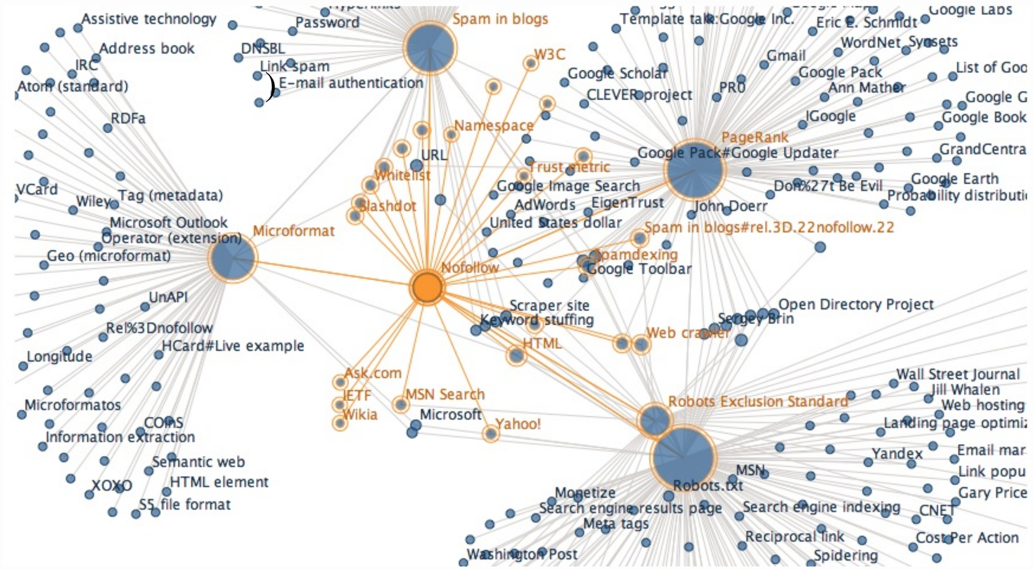
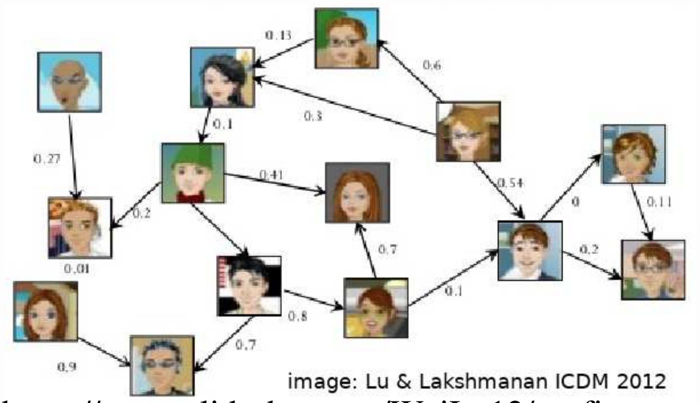


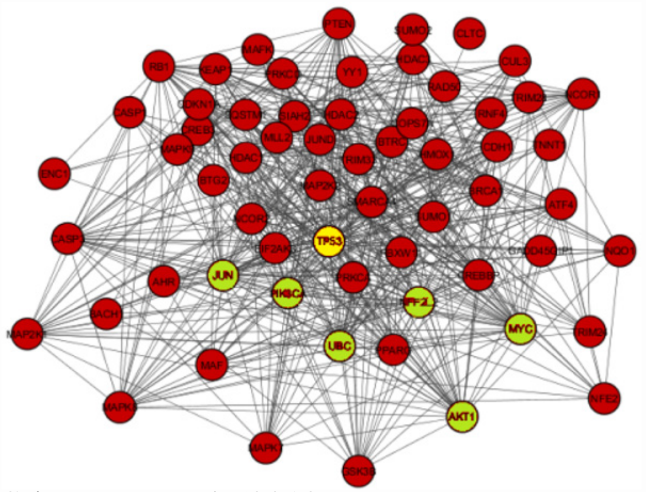
Image source:https://wiki.digitalmethods.net/Dmi/WikipediaAnalysis

Social network



https://www.slideshare.net/WeiLu12/profit–maximization–over–social–networks

Protein–protein interaction network

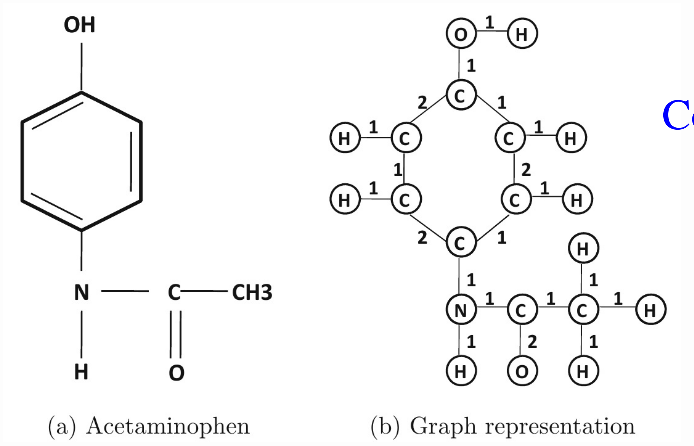


Srivastava et al. (2018)
doi 10.2174/1875036201811010240

Most influential nodes?
Communities or dense subgraphs?
Node similarity/future links?

Data: multiple (smaller) graphs

Molecular structure graphs



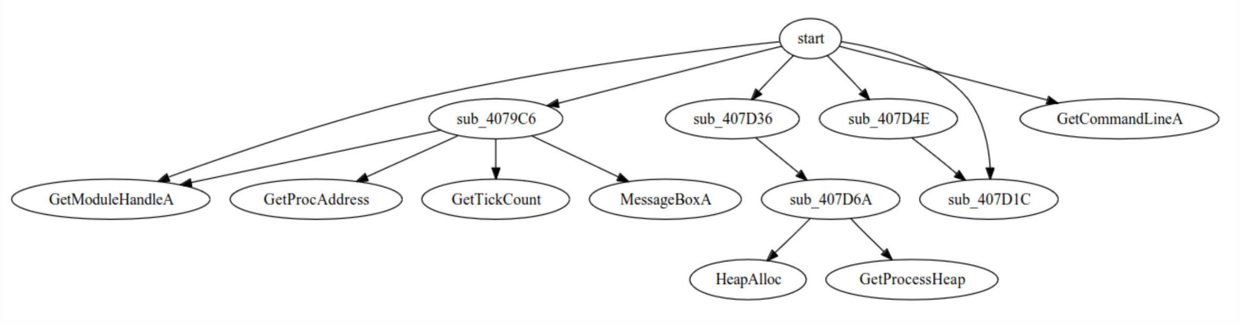
Aggarwal Fig. 17.1

Common and unique properties?

Frequent subgraphs?

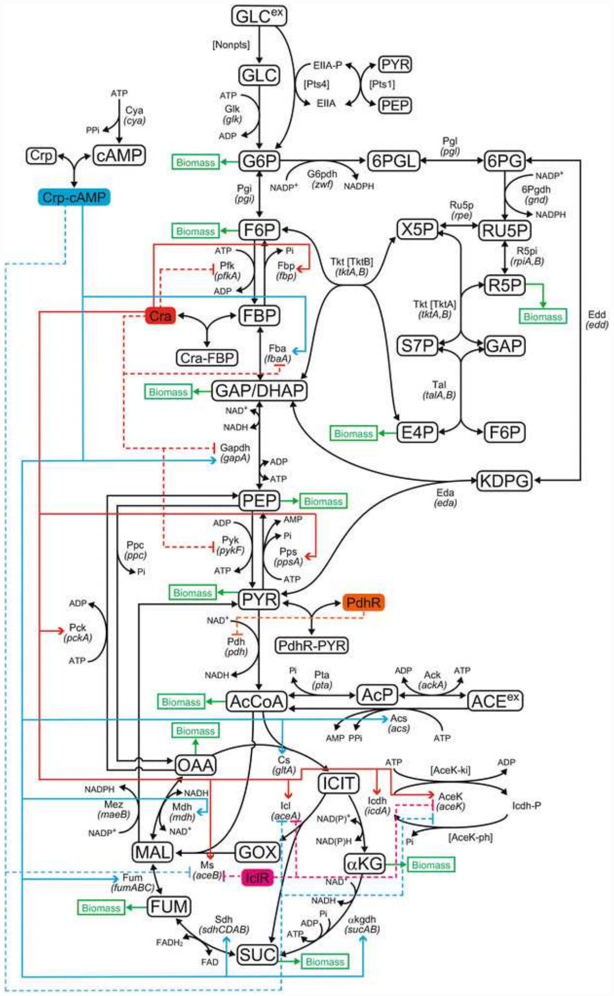
Clusters?

Software call graphs (malware)



Kinable & Kostakis (2011)
doi https://doi.org/10.1007/s11416–011–0151–y

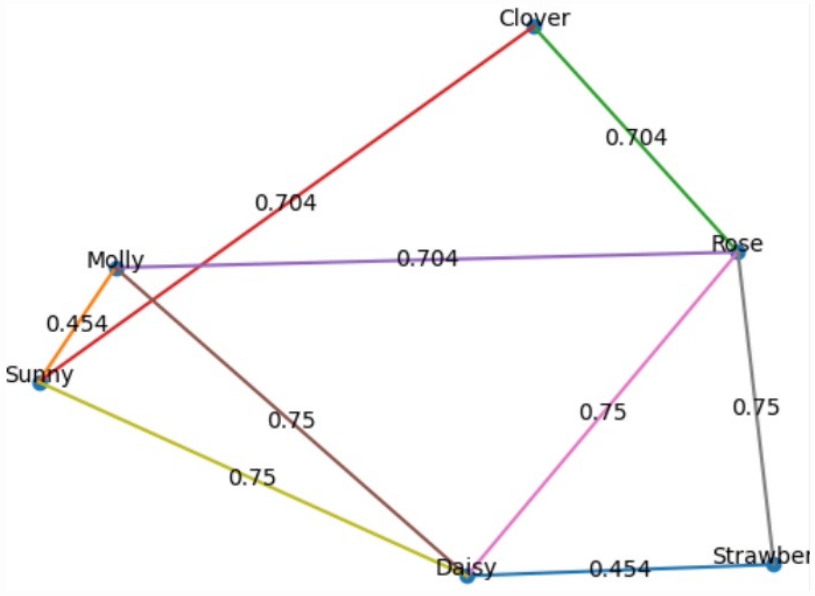
Metabolic ntwork (E. coli)



Jahan et al. (2016)
DOI:10.1186/s12934–016–0511–x

Data: graph-form presentation of other data

Similarity graph



User-item utility graph

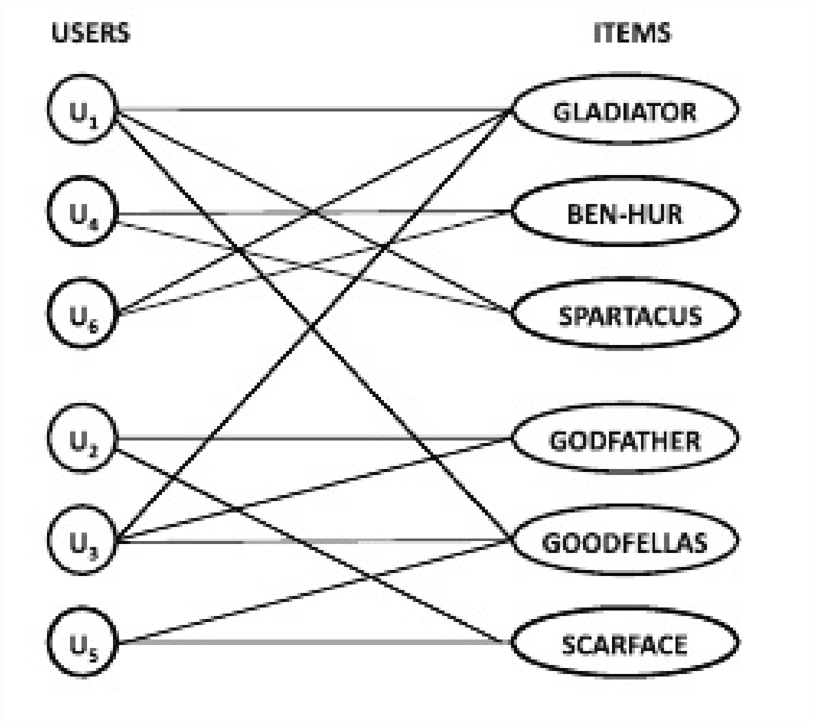


Image sources MDM2022/Laaksonen and Aggarwal Fig 18.5

2a Web mining

Data:

1. Web content = documents and links
2. Wed usage data, like buying and browsing behaviour, ratings, logs

Applications:

1. **Content-centric**: document clustering, resource discovery, web search, linkage mining
2. **Usage-centric**: recommender systems, web log analysis

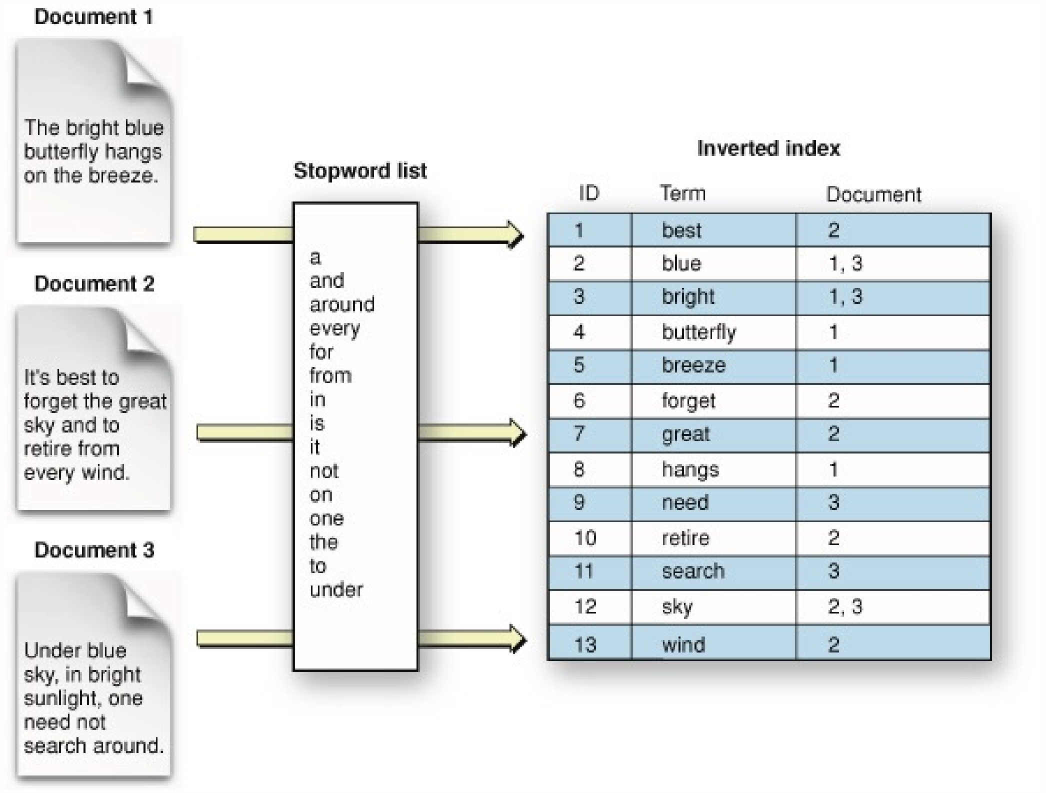
2b Web search (information retrieval)

1. **Web crawling**: collect all (relevant) documents into a central location (+ keep it up to date) $\rightarrow$ Aggarwal 18.2
2. **Index construction** for the collection (offline)
3. **Query processing** (online)
   - query = set of keywords $\mathbf{q} = (w_1, \ldots, w_k)$
   - identify documents containing all/most $w_i \in \mathbf{q}$
   - **rank** documents (relevance to the query and quality)
   - return best ranked documents

Search engine: steps 2 + 3

Index construction

Preprocess documents, extract relevant tokens (words or phrases) and construct **inverted indices**



Document 1
The bright blue butterfly hangs on the breeze.
Document 2
It's best to forget the great sky and to retire from every wind.
Document 3
Under blue sky, in bright sunlight, one need not search around.
Stopword list
a
and
around
every
for
from
in
is
it
not
on
one
the
to
under
Inverted index

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Image source `https://community.hitachivantara.com/s/article/search-the-inverted-index`

Ranking documents

Combine two types of scores:

1. Content-based scores based on occurrence of keywords

- frequency of word
- where the word occurs (more weight if in the title or anchor text)
- prominence of the word (font size, colour)
- relative position of keywords (more relevant if close to each other)

How web spammer can cheat the search engine to get high content-based scores?

Ranking documents

2. Reputation-based scores utilize natural voting mechanisms

a) assumptions based on page citations

- High quality pages are pointed by many pages or
- High quality pages are pointed by many high quality pages

b) user feedback

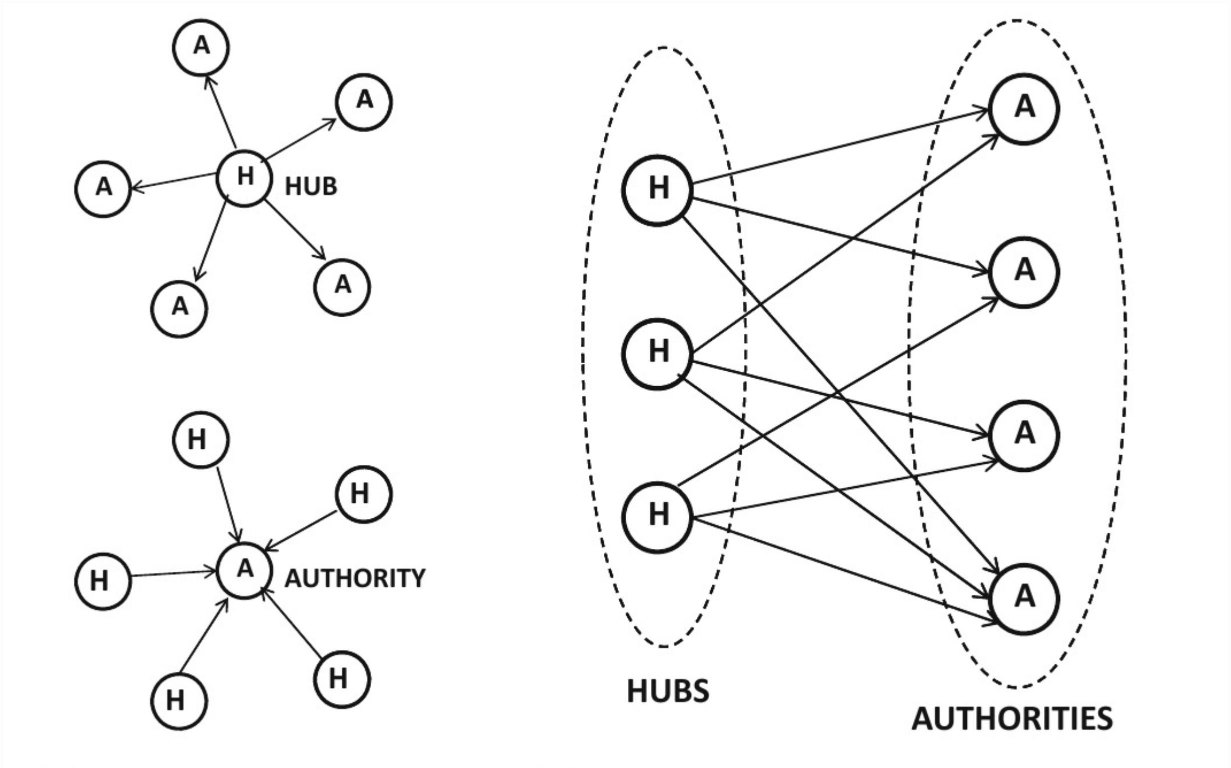
- users choose relevant pages
- return other similar pages or pages accessed by similar users (recommender systems)

Two ranking algorithms

1. HITS = Hyperlink-Induced Topic Search = ”Hubs and Authorities algorithm”
   - query-dependent
   - There are two types of good pages: good authority pages are reputable sources on topics and good hub pages offer links to good sources.
2. PageRank
   - query-independent
   - Highly reputabe pages are likely cited by other higly reputable pages.

Hyperlink-Induced Topic Search (HITS)

hub = page that refers to many authoritative pages
authority = authoritative page refered by many hubs



(a) Hub and authority examples (b) Network organization between hubs and authorities

image source Aggarwal Fig. 18.3

HITS basic idea

1. Construct an input graph:
   - search top-K pages most relevant to the query (e.g., top-200) $\rightarrow$ root set $\mathbf{R}$
   - add pages pointed by $v \in \mathbf{R}$ and some of pages pointing to $v \in \mathbf{R}$ (e.g., max 50/per node) $\rightarrow$ base set $\mathbf{V}$
   - construct a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ where $\mathbf{E}$ contains all links between nodes in $\mathbf{V}$

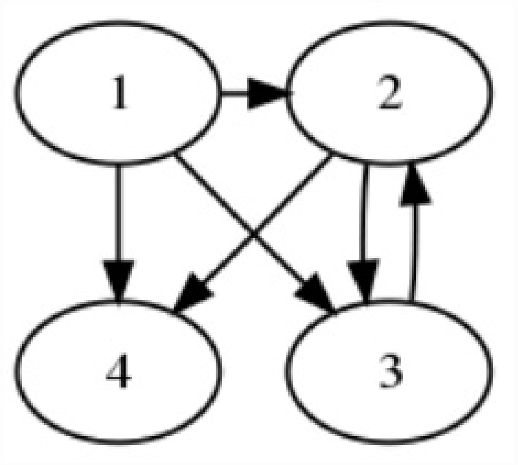
2. Assign all nodes hub and authority weights, $h(v)$ and $a(v)$
   - update weights until the system converges
   - return nodes v with highest $a(v)$

HITS algorithm on $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

$In(v) = \{u \mid (u, v) \in \mathbf{E}\}$ nodes pointing to v
$Out(v) = \{u \mid (v, u) \in \mathbf{E}\}$ nodes to which v points

1. initialize h and a such that $\sum_i h(v_i)^2 = \sum_i a(v_i)^2 = 1$ (e.g., $h(v_i) = a(v_i) = \frac{1}{\sqrt{n}}$)
2. until h and a values converge, update for all v_i:
   - $h(v_i) = \sum_{w \in Out(v_i)} a(w)$ (authorities pointed by v_i)
   - $a(v_i) = \sum_{w \in In(v_i)} h(w)$ (hubs pointing to v_i)
   - normalize weights: $h(v_i) = \frac{h(v_i)}{H}$ and $a(v_i) = \frac{a(v_i)}{A}$, where $H = \sqrt{\sum_i h(v_i)^2}$ ja $A = \sqrt{\sum_i a(v_i)^2}$
3. return v_is with largest $a(v_i)$

Example: calculation of h and a



initialize $h(i) = a(i) = \frac{1}{\sqrt{4}} = \frac{1}{2}$

round 1:

$h(1) = \frac{3}{2}$, $a(1) = 0$

$h(2) = \frac{2}{2}$, $a(2) = \frac{2}{2}$

$h(3) = \frac{1}{2}$, $a(3) = \frac{2}{2}$

$h(4) = 0$, $a(4) = \frac{2}{2}$

normalize by $H = \sqrt{\frac{7}{2}}$ and $A = \sqrt{3}$

$h(1) = \frac{3}{\sqrt{14}}$, $a(1) = 0$

$h(2) = \frac{2}{\sqrt{14}}$, $a(2) = \frac{1}{\sqrt{3}}$

$h(3) = \frac{1}{\sqrt{14}}$, $a(3) = \frac{1}{\sqrt{3}}$

$h(4) = 0$, $a(4) = \frac{1}{\sqrt{3}}$

Example: final result

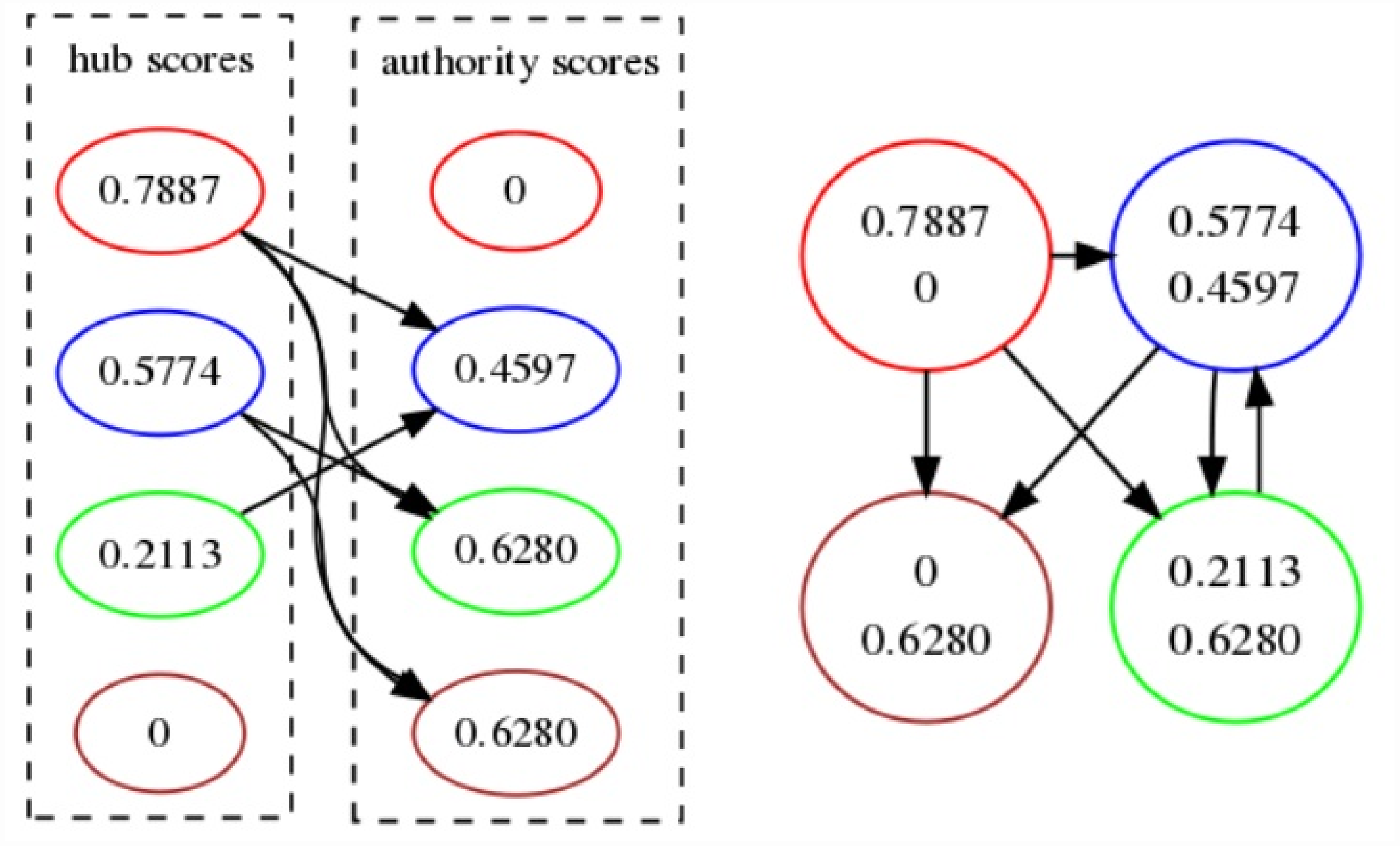


image source Pajarinen (2008): The PageRank/HITS algorithms (slides)

PageRank

Uses a **random surfer model = random walk model**

- visit random pages by selecting random links
- long term visiting frequency of a page depends on the number of in-linking pages and their visiting frequency
- $\Rightarrow$ frequency is high, if linked from other frequently visited pages
- reputation $\approx$ long term frequency of visits by a random surfer = **steady-state probability** π

$\Rightarrow$ calculate $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)^T$, where π_i = PageRank of the ith node v_i (n=number of pages)

Problem: the surfer can get trapped!

Originally, all out-going links from a node have equal probability.

Dead-end nodes don't have out-going links

⇒ add links from a dead-end node to all n pages with transition probability $\frac{1}{n}$

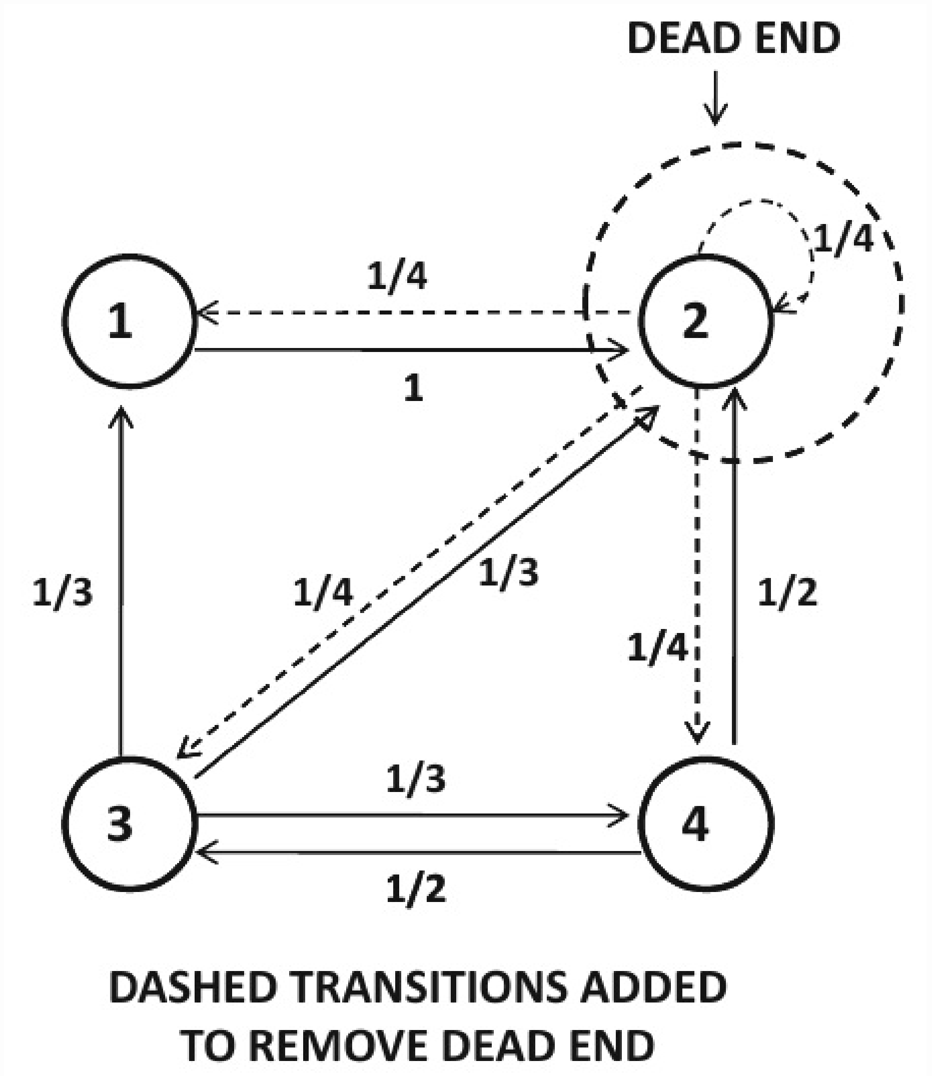


image source Aggarwal Fig. 18.2

Problem: the surfer can get trapped!

Dead-end components don't contain out-going links from the group
⇒ all steady-state probability becomes concentrated into such groups!

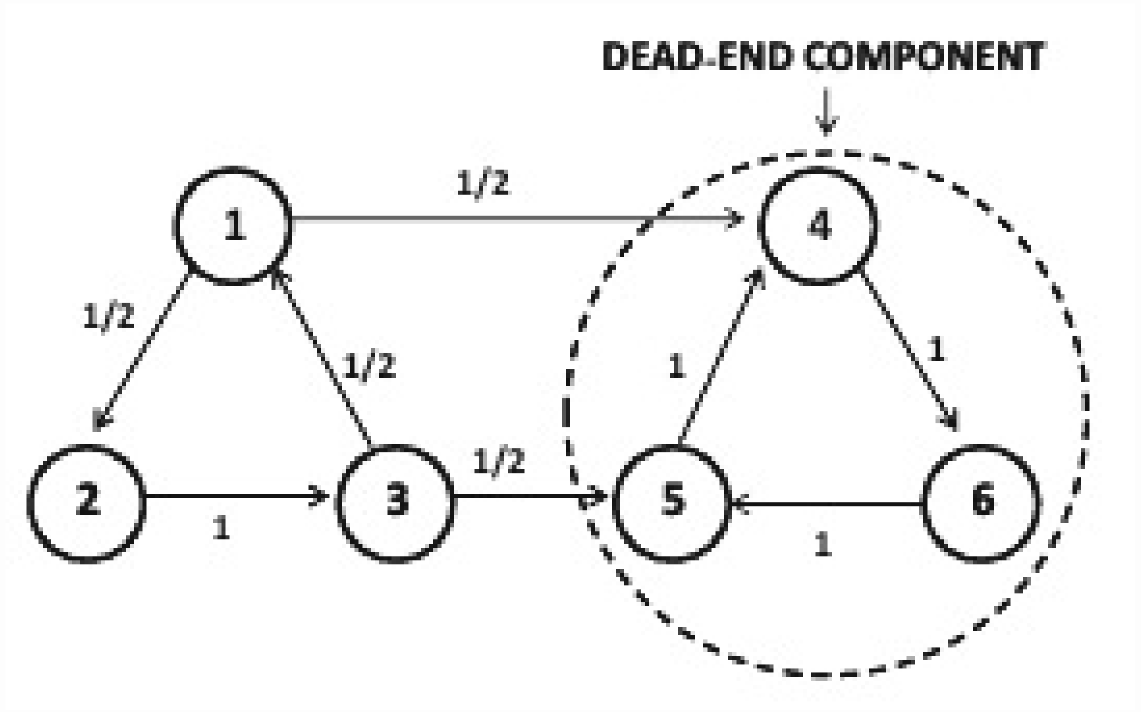


image source Aggarwal Fig. 18.2

Teleportation offers a solution to dead-end components

At each transition, a random surfer may

- jump to an arbitrary page with probability α or
- follow one of links with probability $1 - \alpha$

α = **smoothing** or **damping probability**

- typically $\alpha = 0.1$
- large α makes steady-state probability distribution more uniform
- What happens if $\alpha = 1$?

Presentations as a Markov chain

- graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, $\mathbf{V}$ = pages, $|\mathbf{V}| = n$, $\mathbf{E}$ = links
- $In(v) = \{u \mid (u, v) \in \mathbf{E}\}$ nodes pointing to v
- $Out(v) = \{u \mid (v, u) \in \mathbf{E}\}$ nodes pointed by v
- transition matrix $\mathbf{P}$, where $p_{ij} = \mathbf{P}[i, j]$ probability of transitioning $v_i \rightarrow v_j$
- set $p_{ij} = \frac{1}{|Out(v_i)|}$
- $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)^T$ steady-state probabilities

$$\pi_i = \frac{\alpha}{n} + (1 - \alpha) \sum_{v_j \in In(v_i)} p_{ji} \cdot \pi_j$$

Computation with the power iteration method

Idea: update $\boldsymbol{\pi}$ iteratively, notate by $\boldsymbol{\pi}^{(t)}$ the current $\boldsymbol{\pi}$ at time step t.

Let $\boldsymbol{\alpha} = (\alpha, \alpha, \ldots, \alpha)$ (vector of n αs)

- initialize $\pi_i^{(0)} = 1/n$, $t = 0$
- until $\boldsymbol{\pi}^{(t)}$ converges do
  i) calculate $\boldsymbol{\pi}^{(t+1)}$:

$$\boldsymbol{\pi}^{(t+1)} = \frac{\boldsymbol{\alpha}}{n} + (1 - \alpha)\mathbf{P}^T \boldsymbol{\pi}^{(t)}$$

  ii) normalize such that $\sum \pi_i = 1$

Computation with the power iteration method

Given convergation threshold θ:

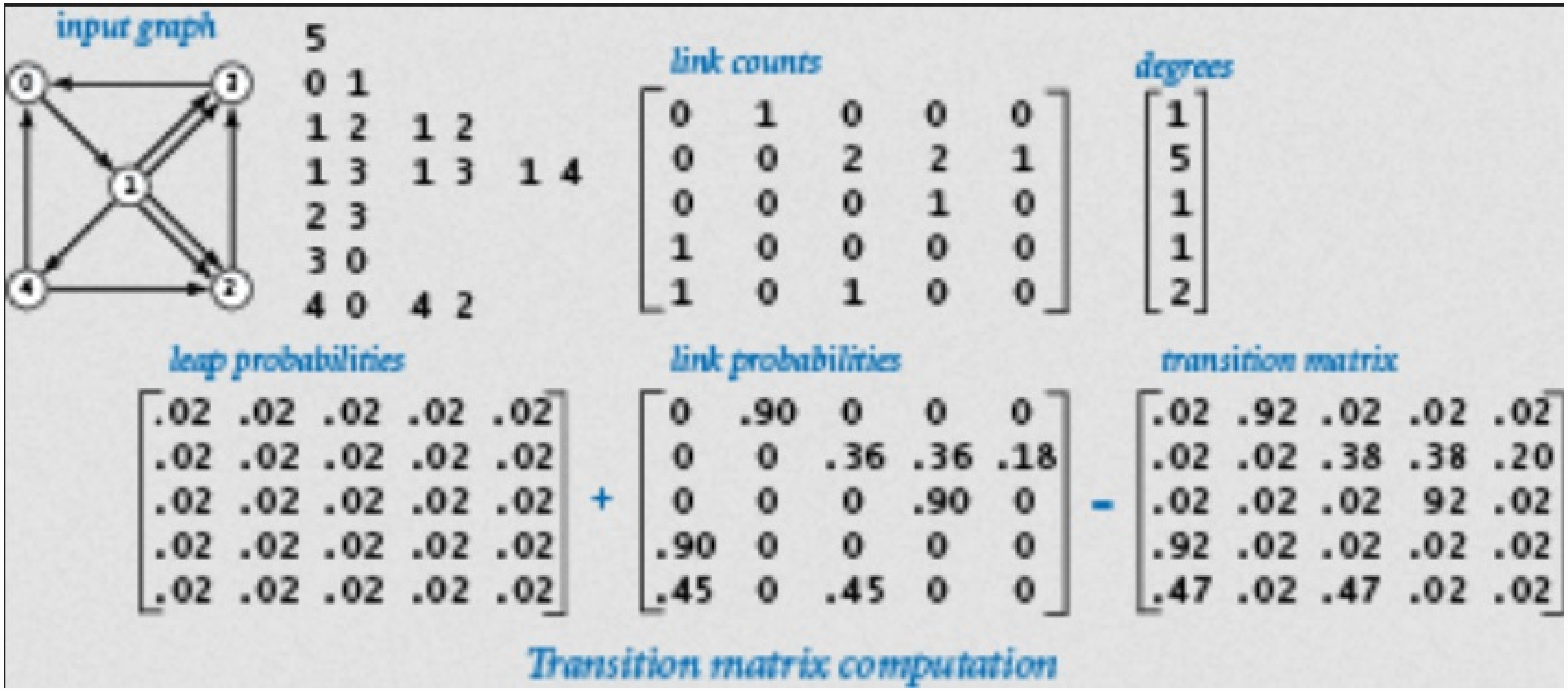
- initialize $\pi_i^{(0)} = 1/n$, $t = 0$
- until ($\|\boldsymbol{\pi}^{(t)} - \boldsymbol{\pi}^{(t+1)}\| \leq \theta$) do
    - for all $i = 1, \ldots, n$

$$\pi_i^{(t+1)} = \frac{\alpha}{n} + (1 - \alpha) \sum_{v_j \in In(v_i)} p_{ji} \pi_j^{(t)}$$

    - for all $i = 1, \ldots, n$ normalize $\pi_i^{(t+1)}$s

Note: heavy computation! $\rightarrow$ calculate beforehand for all pages

Alternatively integrate teleportation probabilities into the transition matrix (here $\alpha = 0.1$)



```
5
0 1
1 2  1 2
1 3  1 3  1 4
2 3
3 0
4 0  4 2
```

link counts

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

degrees

$$\begin{bmatrix} 1 \\ 5 \\ 1 \\ 1 \\ 2 \end{bmatrix}$$

leap probabilities + link probabilities = transition matrix

$$\begin{bmatrix} .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \\ .02 & .02 & .02 & .02 & .02 \end{bmatrix} + \begin{bmatrix} 0 & .90 & 0 & 0 & 0 \\ 0 & 0 & .36 & .36 & .18 \\ 0 & 0 & 0 & .90 & 0 \\ .90 & 0 & 0 & 0 & 0 \\ .45 & 0 & .45 & 0 & 0 \end{bmatrix} = \begin{bmatrix} .02 & .92 & .02 & .02 & .02 \\ .02 & .02 & .38 & .38 & .20 \\ .02 & .02 & .02 & 92 & .02 \\ .92 & .02 & .02 & .02 & .02 \\ .47 & .02 & .47 & .02 & .02 \end{bmatrix}$$

Transition matrix computation

image source
`https://introcs.cs.princeton.edu/python/16pagerank/`

Other PageRank style methods

1. Topic-sensitive PageRank favours certain topics (like user's interests)

- fix a list of base topics
- search high quality sample of pages from each topic
- restrict teleportation only to these pages

2. SimRank measures similarity between nodes

Intuition: Two random surfers walking backwards from nodes i and j take $L(i, j)$ steps, until they meet. Then SimRank(i, j) is expected value of $c^{L(i,j)}$, where c = decay constant.

SimRank

$$SimRank(v_i, v_j) =$$

$$\begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } |In(v_i)| = 0 \text{ or } |In(v_j)| = 0 \\ \frac{c}{|In(v_i)| \cdot |In(v_j)|} \sum_{p \in In(v_i)} \sum_{q \in In(v_j)} SimRank(p, q) & \text{otherwise} \end{cases}$$

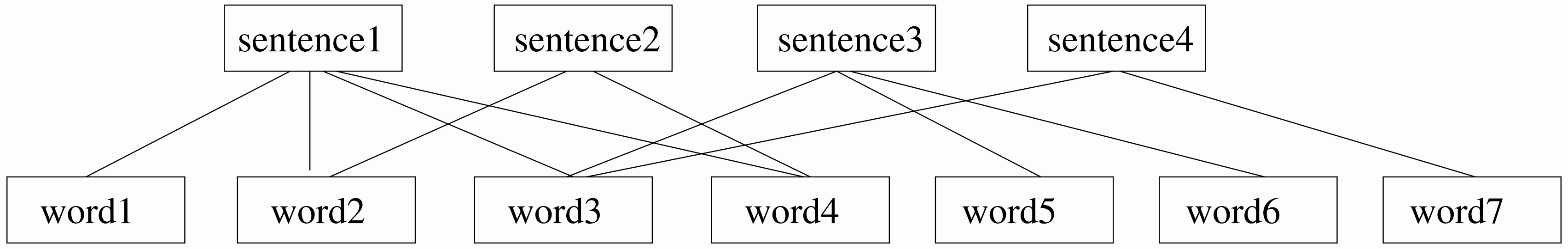
Further reading Aggarwal Ch 18.4.1.2

Summary

- Important problem how to rank documents!
- Content-based scores based on page contents
- Reputation-based scores utilize e.g., link structure
- Ranking algorithms:
    - HITS: query-dependent $\rightarrow$ smaller input graph
    - PageRank: query-independent $\rightarrow$ calculate for all pages (heavy!)

Extra: Applying HITS to sentences and words

Given a keyword or words, create a bipartite graph:



- e.g., all sentences where the given word occurs, all their words, all their sentences and randomly some of their words
- sentences (s) are given S weights and words (w) W weights
- initialize uniformly such that the square sum is 1

Extra: Applying HITS to sentences and words

- update rule: $S(s) = \sum_{w_i \in s} W(w_i)$ and $W(w) = \sum_{w \in s_i} S(s_i)$ + normalization
- in the end the words with largest S-weight are most similar
- analogously search the most similar sentences with a given one
- quality of results varies a lot depending on the documents, language, preprocessing, and how the graph was constructed!