

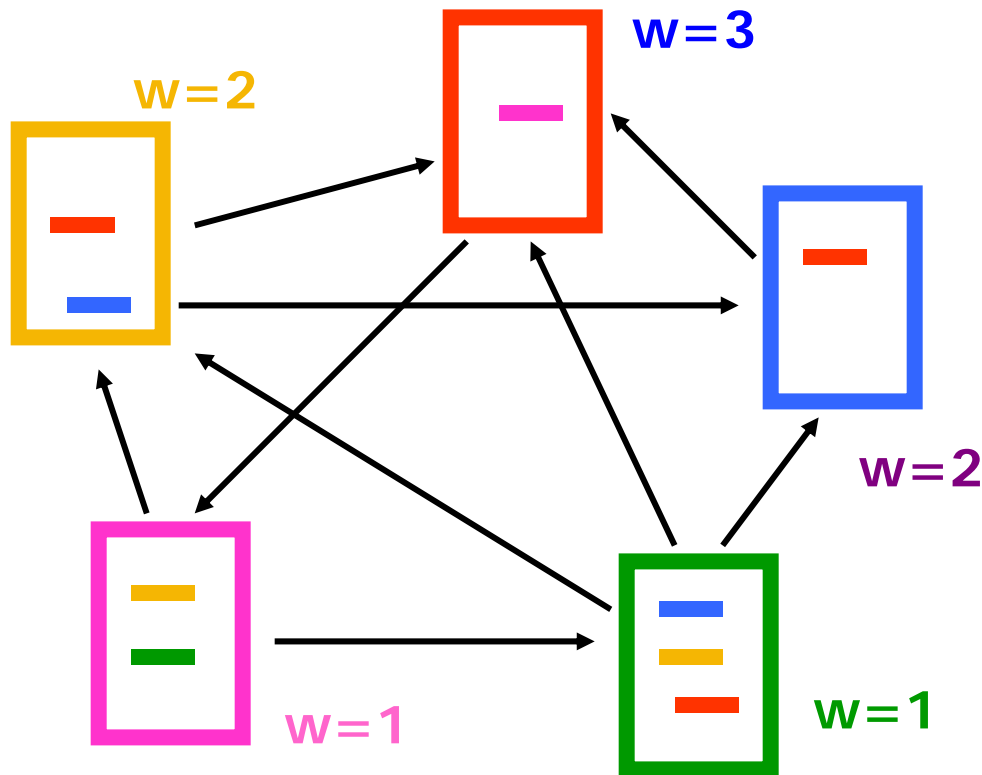
# More on Rankings

# Query-independent LAR

- Have an a-priori ordering of the web pages
- **Q**: Set of pages that contain the keywords in the query **q**
- Present the pages in **Q** ordered according to order  **$\pi$**
- **What are the advantages of such an approach?**

# InDegree algorithm

- Rank pages according to in-degree
  - $w_i = |B(i)|$

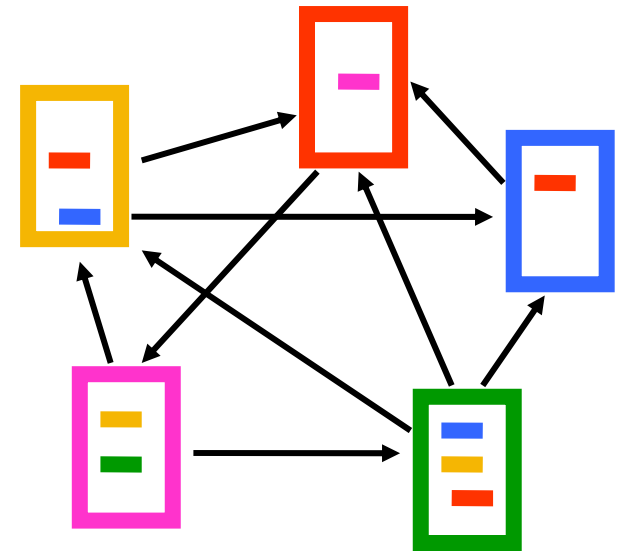


1. Red Page
2. Yellow Page
3. Blue Page
4. Purple Page
5. Green Page

# PageRank algorithm [BP98]

- **Good** authorities should be pointed by **good** authorities
- Random walk on the web graph
  - pick a page at random
  - with probability  $1 - \alpha$  jump to a random page
  - with probability  $\alpha$  follow a random outgoing link
- Rank according to the stationary distribution

$$PR(p) = \alpha \sum_{q \rightarrow p} \frac{PR(q)}{|F(q)|} + (1 - \alpha) \frac{1}{n}$$



1. Red Page
2. Purple Page
3. Yellow Page
4. Blue Page
5. Green Page

# Markov chains

- A Markov chain describes a discrete time stochastic process over a set of states

$$S = \{s_1, s_2, \dots, s_n\}$$

according to a transition probability matrix

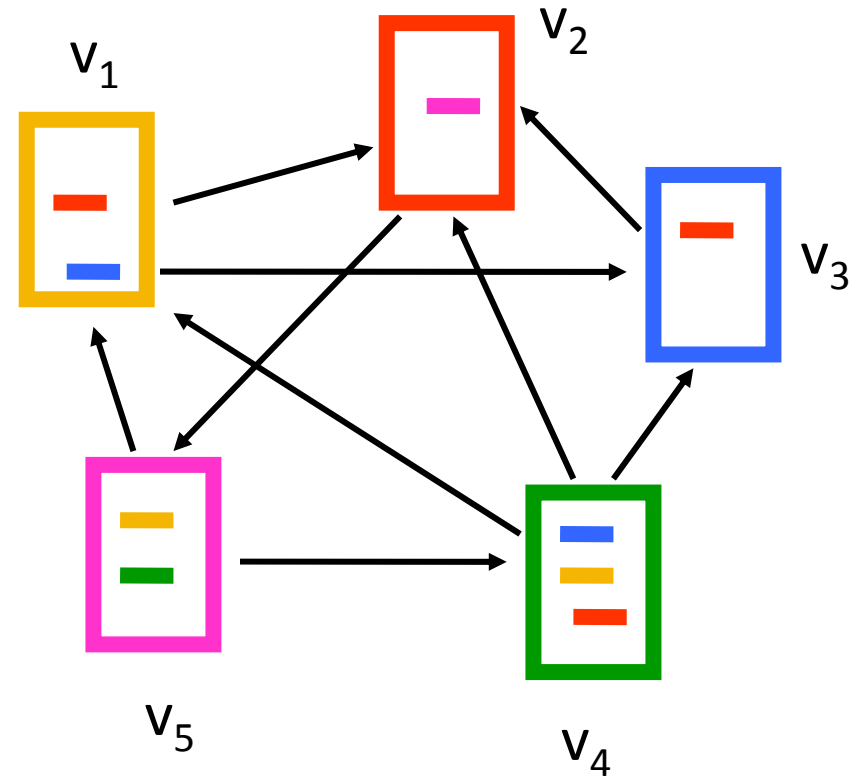
$$P = \{P_{ij}\}$$

- $P_{ij}$  = probability of moving to state  $j$  when at state  $i$ 
  - $\sum_j P_{ij} = 1$  (stochastic matrix)
- **Memorylessness property**: The next state of the chain depends only at the current state and not on the past of the process (first order MC)
  - higher order MCs are also possible

# Random walks

- Random walks on graphs correspond to Markov Chains
  - The set of states  $S$  is the set of nodes of the graph  $G$
  - The **transition probability matrix** is the probability that we follow an edge from one node to another

# An example

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$
$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 1/2 \end{bmatrix}$$


# State probability vector

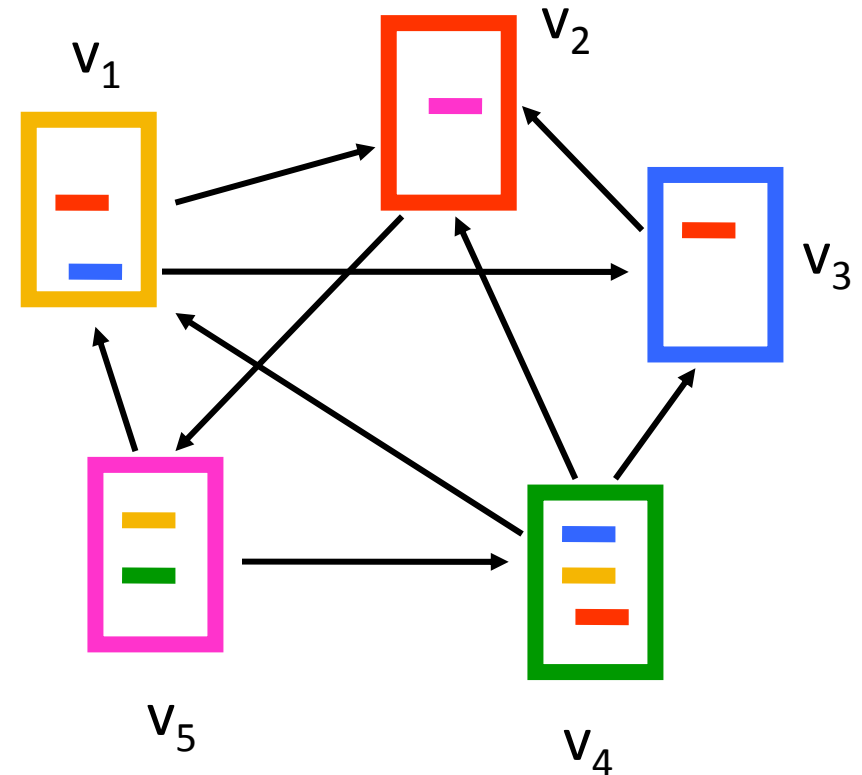
- The vector  $q^t = (q^t_1, q^t_2, \dots, q^t_n)$  that stores the probability of being at state  $i$  at time  $t$ 
  - $q^0_i$  = the probability of starting from state  $i$

$$q^t = q^{t-1} P$$



# An example

$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$



$$q_1^{t+1} = 1/3 q_4^t + 1/2 q_5^t$$

$$q_2^{t+1} = 1/2 q_1^t + q_3^t + 1/3 q_4^t$$

$$q_3^{t+1} = 1/2 q_1^t + 1/3 q_4^t$$

$$q_4^{t+1} = 1/2 q_5^t$$

$$q_5^{t+1} = q_2^t$$

# Stationary distribution

- A stationary distribution for a MC with transition matrix  $P$ , is a probability distribution  $\pi$ , such that  $\pi = \pi P$
- A MC has a unique stationary distribution if
  - it is **irreducible**
    - the underlying graph is strongly connected
  - it is **aperiodic**
    - for random walks, the underlying graph is **not** bipartite
- The probability  $\pi_i$  is the fraction of times that we visited state  $i$  as  $t \rightarrow \infty$
- The stationary distribution is an eigenvector of matrix  $P$ 
  - the principal left eigenvector of  $P$  – stochastic matrices have maximum eigenvalue 1

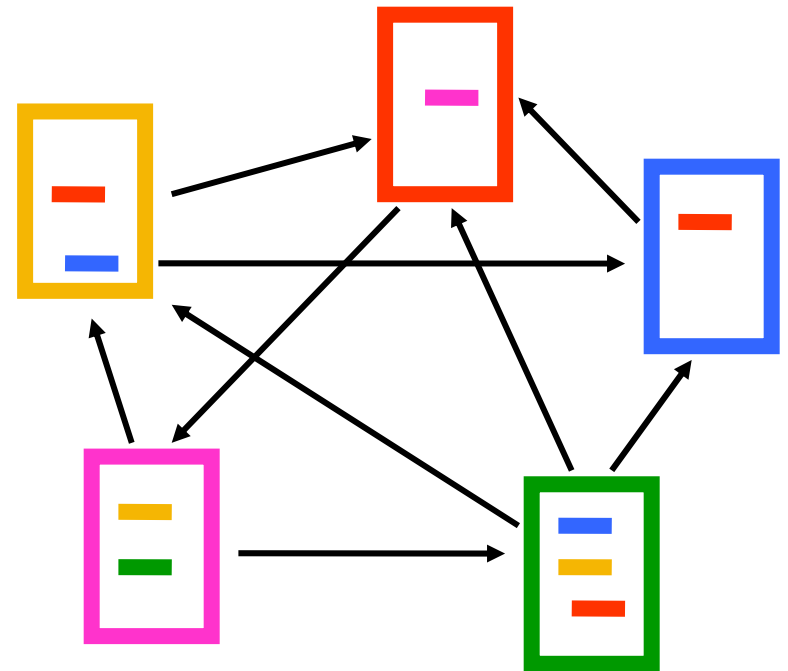
# Computing the stationary distribution

- The Power Method
  - Initialize to some distribution  $q^0$
  - Iteratively compute  $q^t = q^{t-1}P$
  - After enough iterations  $q^t \approx \pi$
  - Power method because it computes  $q^t = q^0 P^t$
- Rate of convergence
  - determined by  $\lambda_2$

# The PageRank random walk

- Vanilla random walk
  - make the adjacency matrix stochastic and run a random walk

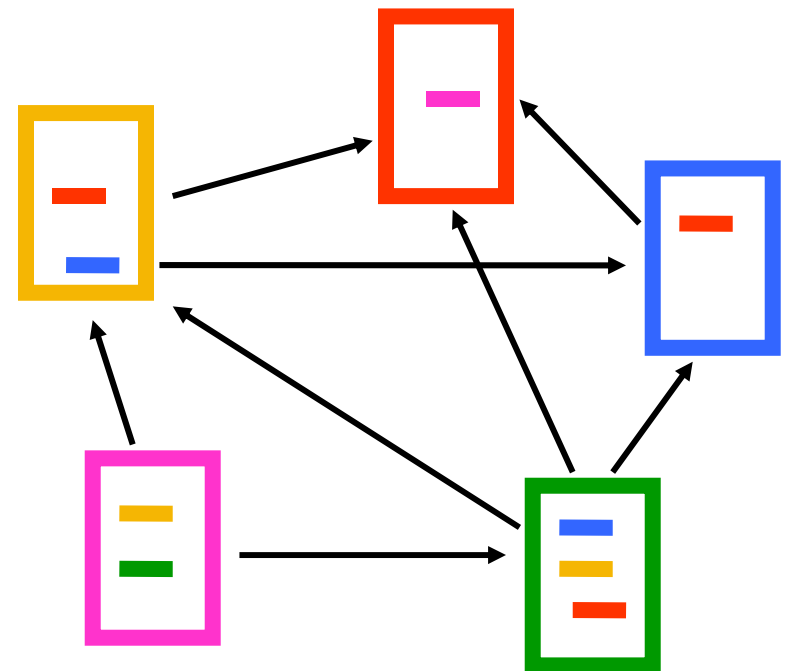
$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$



# The PageRank random walk

- What about **sink** nodes?
  - what happens when the random walk moves to a node without any outgoing links?

$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$

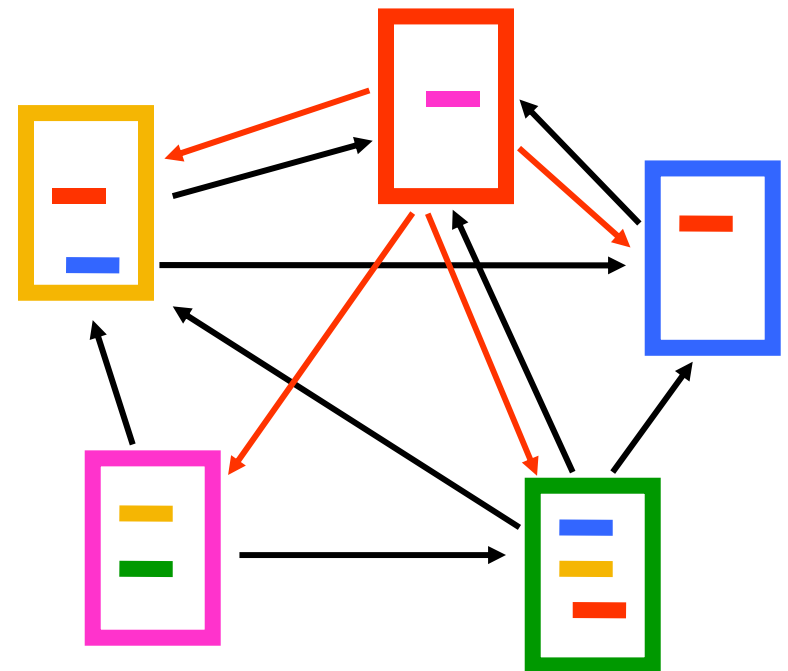


# The PageRank random walk

- Replace these row vectors with a vector  $\mathbf{v}$ 
  - typically, the uniform vector

$$P' = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$

$$P' = P + d\mathbf{v}^T \quad d = \begin{cases} 1 & \text{if } i \text{ is sink} \\ 0 & \text{otherwise} \end{cases}$$



# The PageRank random walk

- How do we guarantee irreducibility?
  - add a random jump to vector  $v$  with prob  $\alpha$ 
    - typically, to a uniform vector

$$P'' = \alpha \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 1/2 \end{bmatrix} + (1 - \alpha) \begin{bmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{bmatrix}$$

$P'' = \alpha P' + (1 - \alpha)uv^T$ , where  $u$  is the vector of all 1s

# Effects of random jump

- Guarantees irreducibility
- Motivated by the concept of random surfer
- Offers additional flexibility
  - personalization
  - anti-spam
- Controls the rate of convergence
  - the second eigenvalue of matrix  $P''$  is  $\alpha$



# A PageRank algorithm

- Performing vanilla power method is now too expensive – the matrix is not sparse

$$q^0 = v$$

$$t = 1$$

repeat

$$q^t = (P'')^T q^{t-1}$$

$$\delta = \|q^t - q^{t-1}\|$$

$$t = t + 1$$

until  $\delta < \epsilon$

Efficient computation of  $y = (P'')^T x$

$$y = \alpha P'^T x$$

$$\beta = \|x\|_1 - \|y\|_1$$

$$y = y + \beta v$$

# Random walks on undirected graphs

- In the stationary distribution of a random walk on an undirected graph, the probability of being at node  $i$  is proportional to the (weighted) degree of the vertex
- Random walks on undirected graphs are not “interesting”

# Research on PageRank

- Specialized PageRank
  - personalization [BP98]
    - instead of picking a node uniformly at random favor specific nodes that are related to the user
  - topic sensitive PageRank [H02]
    - compute many PageRank vectors, one for each topic
    - estimate relevance of query with each topic
    - produce final PageRank as a weighted combination
- Updating PageRank [Chien et al 2002]
- Fast computation of PageRank
  - numerical analysis tricks
  - node aggregation techniques
  - dealing with the “Web frontier”

# Topic-sensitive pagerank

- HITS-based scores are very inefficient to compute
- PageRank scores are independent of the queries
- Can we bias PageRank rankings to take into account query keywords?

**Topic-sensitive PageRank**

# Topic-sensitive PageRank

- Conventional PageRank computation:
- $r^{(t+1)}(v) = \sum_{u \in N(v)} r^{(t)}(u) / d(v)$
- $N(v)$ : neighbors of  $v$
- $d(v)$ : degree of  $v$
- $r = Mx$
- $M' = (1-\alpha)P' + \alpha \left[ \frac{1}{n} \right]_{n \times n}$
- $r = (1-\alpha)P'r + \alpha \left[ \frac{1}{n} \right]_{n \times n} r = (1-\alpha)P'r + \alpha p$
- $p = \left[ \frac{1}{n} \right]_{n \times 1}$

# Topic-sensitive PageRank

- $r = (1-\alpha)P'r + \alpha p$
- **Conventional PageRank:**  $p$  is a uniform vector with values  $1/n$
- Topic-sensitive PageRank uses a **non-uniform** personalization vector  $p$
- Not simply a post-processing step of the PageRank computation
- Personalization vector  $p$  introduces bias in all iterations of the iterative computation of the PageRank vector

# Personalization vector

- In the random-walk model, the personalization vector represents the addition of a set of transition edges, where the probability of an artificial edge  $(u,v)$  is  $\alpha p_v$
- Given a graph the result of the PageRank computation only depends on  $\alpha$  and  $p$  :  
 $PR(\alpha,p)$

# Topic-sensitive PageRank: Overall approach

- Preprocessing
  - Fix a set of  $k$  topics
  - For each topic  $c_j$  compute the PageRank scores of page  $u$  wrt to the  $j$ -th topic:  $r(u,j)$
- Query-time processing:
  - For query  $q$  compute the total score of page  $u$  wrt  $q$  as  $\text{score}(u,q) = \sum_{j=1\dots k} \text{Pr}(c_j|q) r(u,j)$



# Topic-sensitive PageRank: Preprocessing

- Create **k** different biased PageRank vectors using some pre-defined set of **k** categories **(c<sub>1</sub>, ..., c<sub>k</sub>)**
- **T<sub>j</sub>**: set of URLs in the **j**-th category
- Use non-uniform personalization vector **p=w<sub>j</sub>** such that:

$$w_j(v) = \begin{cases} \frac{1}{|T_j|}, & v \in T_j \\ 0, & \text{o/w} \end{cases}$$

# Topic-sensitive PageRank: Query-time processing

- $D_j$ : class term vectors consisting of all the terms appearing in the  $k$  pre-selected categories

$$\Pr(c_j | q) = \frac{\Pr(c_j) \Pr(q | c_j)}{\Pr(q)} \propto \Pr(c_j) \prod_i \Pr(q_i | c_j)$$

- How can we compute  $P(c_j)$ ?
- How can we compute  $\Pr(q_i | c_j)$ ?

- Comparing results of Link Analysis Ranking algorithms
- Comparing and aggregating rankings

# Comparing LAR vectors



$$w_1 = [ 1 \quad 0.8 \quad 0.5 \quad 0.3 \quad 0 ]$$

$$w_2 = [ 0.9 \quad 1 \quad 0.7 \quad 0.6 \quad 0.8 ]$$

- How close are the LAR vectors  $w_1, w_2$ ?

# Distance between LAR vectors

- Geometric distance: how close are the **numerical weights** of vectors  $w_1, w_2$ ?

$$d_1(w_1, w_2) = \sum |w_1[i] - w_2[i]|$$



$$w_1 = [ 1.0 \quad 0.8 \quad 0.5 \quad 0.3 \quad 0.0 ]$$

$$w_2 = [ 0.9 \quad 1.0 \quad 0.7 \quad 0.6 \quad 0.8 ]$$

$$d_1(w_1, w_2) = 0.1 + 0.2 + 0.2 + 0.3 + 0.8 = 1.6$$

# Distance between LAR vectors

- Rank distance: how close are the **ordinal rankings** induced by the vectors  $w_1, w_2$ ?
  - Kendal's  $\tau$  distance

$$d_r(w_1, w_2) = \frac{\text{pairs ranked in a different order}}{\text{total number of distinct pairs}}$$

# Outline

- Rank Aggregation
  - Computing aggregate scores
  - Computing aggregate rankings - voting

# Rank Aggregation

- Given a set of rankings  $R_1, R_2, \dots, R_m$  of a set of objects  $X_1, X_2, \dots, X_n$  produce a single ranking  $R$  that is in agreement with the existing rankings



# Examples

- Voting
  - rankings  $R_1, R_2, \dots, R_m$  are the voters, the objects  $X_1, X_2, \dots, X_n$  are the candidates.

# Examples

- Combining multiple scoring functions
  - rankings  $R_1, R_2, \dots, R_m$  are the scoring functions, the objects  $X_1, X_2, \dots, X_n$  are data items.
    - Combine the PageRank scores with term-weighting scores
    - Combine scores for multimedia items
      - color, shape, texture
    - Combine scores for database tuples
      - find the best hotel according to price and location

# Examples

- Combining multiple sources
  - rankings  $R_1, R_2, \dots, R_m$  are the sources, the objects  $X_1, X_2, \dots, X_n$  are data items.
    - meta-search engines for the Web
    - distributed databases
    - P2P sources

# Variants of the problem

- Combining scores
  - we know the scores assigned to objects by each ranking, and we want to compute a single score
- Combining ordinal rankings
  - the scores are not known, only the ordering is known
  - the scores are known but we do not know how, or do not want to combine them
    - e.g. price and star rating

# Combining scores

- Each object  $X_i$  has  $m$  scores  
 $(r_{i1}, r_{i2}, \dots, r_{im})$
- The score of object  $X_i$  is  
computed using an  
aggregate scoring function  
 $f(r_{i1}, r_{i2}, \dots, r_{im})$

	$R_1$	$R_2$	$R_3$
$X_1$	1	0.3	0.2
$X_2$	0.8	0.8	0
$X_3$	0.5	0.7	0.6
$X_4$	0.3	0.2	0.8
$X_5$	0.1	0.1	0.1

# Combining scores

- Each object  $X_i$  has  $m$  scores  $(r_{i1}, r_{i2}, \dots, r_{im})$
- The score of object  $X_i$  is computed using an **aggregate scoring function**  $f(r_{i1}, r_{i2}, \dots, r_{im})$ 
  - $f(r_{i1}, r_{i2}, \dots, r_{im}) = \min\{r_{i1}, r_{i2}, \dots, r_{im}\}$

	$R_1$	$R_2$	$R_3$	$R$
$X_1$	1	0.3	0.2	0.2
$X_2$	0.8	0.8	0	0
$X_3$	0.5	0.7	0.6	0.5
$X_4$	0.3	0.2	0.8	0.2
$X_5$	0.1	0.1	0.1	0.1

# Combining scores

- Each object  $X_i$  has  $m$  scores  $(r_{i1}, r_{i2}, \dots, r_{im})$
- The score of object  $X_i$  is computed using an **aggregate scoring function**  $f(r_{i1}, r_{i2}, \dots, r_{im})$ 
  - $f(r_{i1}, r_{i2}, \dots, r_{im}) = \max\{r_{i1}, r_{i2}, \dots, r_{im}\}$

	$R_1$	$R_2$	$R_3$	$R$
$X_1$	1	0.3	0.2	1
$X_2$	0.8	0.8	0	0.8
$X_3$	0.5	0.7	0.6	0.7
$X_4$	0.3	0.2	0.8	0.8
$X_5$	0.1	0.1	0.1	0.1

# Combining scores

- Each object  $X_i$  has  $m$  scores  $(r_{i1}, r_{i2}, \dots, r_{im})$
- The score of object  $X_i$  is computed using an **aggregate scoring function**  $f(r_{i1}, r_{i2}, \dots, r_{im})$ 
  - $f(r_{i1}, r_{i2}, \dots, r_{im}) = r_{i1} + r_{i2} + \dots + r_{im}$

	$R_1$	$R_2$	$R_3$	$R$
$X_1$	1	0.3	0.2	1.5
$X_2$	0.8	0.8	0	1.6
$X_3$	0.5	0.7	0.6	1.8
$X_4$	0.3	0.2	0.8	1.3
$X_5$	0.1	0.1	0.1	0.3



# Top-k

- Given a set of  $n$  objects and  $m$  scoring lists **sorted** in decreasing order, find the **top-k** objects according to a scoring function  $f$
- **top-k**: a set  $T$  of  $k$  objects such that  $f(r_{j1}, \dots, r_{jm}) \leq f(r_{i1}, \dots, r_{im})$  for every object  $X_i$  in  $T$  and every object  $X_j$  not in  $T$
- **Assumption**: The function  $f$  is monotone
  - $f(r_1, \dots, r_m) \leq f(r_1', \dots, r_m')$  if  $r_i \leq r_i'$  for all  $i$
- **Objective**: Compute top-k with the minimum cost

# Cost function

- We want to minimize the number of accesses to the scoring lists
- **Sorted accesses**: sequentially access the objects in the order in which they appear in a list
  - cost  $C_s$
- **Random accesses**: obtain the cost value for a specific object in a list
  - cost  $C_r$
- If  $s$  sorted accesses and  $r$  random accesses minimize  $s C_s + r C_r$

# Example

$R_1$	
$X_1$	1
$X_2$	0.8
$X_3$	0.5
$X_4$	0.3
$X_5$	0.1

$R_2$	
$X_2$	0.8
$X_3$	0.7
$X_1$	0.3
$X_4$	0.2
$X_5$	0.1

$R_3$	
$X_4$	0.8
$X_3$	0.6
$X_1$	0.2
$X_5$	0.1
$X_2$	0

- Compute top-2 for the **sum** aggregate function

# Fagin's Algorithm

1. Access sequentially all lists in parallel until there are  $k$  objects that have been seen in **all** lists

$R_1$	
$X_1$	1
$X_2$	0.8
$X_3$	0.5
$X_4$	0.3
$X_5$	0.1

$R_2$	
$X_2$	0.8
$X_3$	0.7
$X_1$	0.3
$X_4$	0.2
$X_5$	0.1

$R_3$	
$X_4$	0.8
$X_3$	0.6
$X_1$	0.2
$X_5$	0.1
$X_2$	0

# Fagin's Algorithm

1. Access sequentially all lists in parallel until there are  $k$  objects that have been seen in **all** lists

$R_1$			$R_2$			$R_3$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1
$X_5$	0.1		$X_5$	0.1		$X_2$	0

# Fagin's Algorithm

1. Access sequentially all lists in parallel until there are  $k$  objects that have been seen in **all** lists

$R_1$			$R_2$			$R_3$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1
$X_5$	0.1		$X_5$	0.1		$X_2$	0

# Fagin's Algorithm

1. Access sequentially all lists in parallel until there are  $k$  objects that have been seen in **all** lists

$R_1$			$R_2$			$R_3$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1
$X_5$	0.1		$X_5$	0.1		$X_2$	0

# Fagin's Algorithm

1. Access sequentially all lists in parallel until there are  $k$  objects that have been seen in **all** lists

$R_1$			$R_2$			$R_3$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1
$X_5$	0.1		$X_5$	0.1		$X_2$	0



# Fagin's Algorithm

2. Perform random accesses to obtain the scores of all seen objects

$R_1$			$R_2$			$R_3$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1
$X_5$	0.1		$X_5$	0.1		$X_2$	0

# Fagin's Algorithm

3. Compute score for all objects and find the top-k

$R_1$			$R_2$			$R_3$			$R$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8		$X_3$	1.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6		$X_2$	1.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2		$X_1$	1.5
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1		$X_4$	1.3
$X_5$	0.1		$X_5$	0.1		$X_2$	0			

# Fagin's Algorithm

- $X_5$  cannot be in the top-2 because of the monotonicity property
  - $f(X_5) \leq f(X_1) \leq f(X_3)$

$R_1$	
$X_1$	1
$X_2$	0.8
$X_3$	0.5
$X_4$	0.3
$X_5$	0.1

$R_2$	
$X_2$	0.8
$X_3$	0.7
$X_1$	0.3
$X_4$	0.2
$X_5$	0.1

$R_3$	
$X_4$	0.8
$X_3$	0.6
$X_1$	0.2
$X_5$	0.1
$X_2$	0

$R$	
$X_3$	1.8
$X_2$	1.6
$X_1$	1.5
$X_4$	1.3

# Fagin's Algorithm

- The algorithm is cost optimal under some probabilistic assumptions for a restricted class of aggregate functions

# Threshold algorithm

1. Access the elements sequentially

$R_1$	
$X_1$	1
$X_2$	0.8
$X_3$	0.5
$X_4$	0.3
$X_5$	0.1

$R_2$	
$X_2$	0.8
$X_3$	0.7
$X_1$	0.3
$X_4$	0.2
$X_5$	0.1

$R_3$	
$X_4$	0.8
$X_3$	0.6
$X_1$	0.2
$X_5$	0.1
$X_2$	0

# Threshold algorithm

1. At each sequential access
  - a. Set the threshold  $t$  to be the aggregate of the scores seen in this access

$R_1$			$R_2$			$R_3$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1
$X_5$	0.1		$X_5$	0.1		$X_2$	0

$t = 2.6$

# Threshold algorithm

1. At each sequential access
  - b. Do random accesses and compute the score of the objects seen

$R_1$	
$X_1$	1
$X_2$	0.8
$X_3$	0.5
$X_4$	0.3
$X_5$	0.1

$R_2$	
$X_2$	0.8
$X_3$	0.7
$X_1$	0.3
$X_4$	0.2
$X_5$	0.1

$R_3$	
$X_4$	0.8
$X_3$	0.6
$X_1$	0.2
$X_5$	0.1
$X_2$	0

$t = 2.6$

$X_1$	1.5
$X_2$	1.6
$X_4$	1.3

# Threshold algorithm

1. At each sequential access
  - c. Maintain a list of top-k objects seen so far

$R_1$			$R_2$			$R_3$				
$X_1$	1		$X_2$	0.8		$X_4$	0.8		$t = 2.6$	
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6		$X_2$	1.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2		$X_1$	1.5
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1			
$X_5$	0.1		$X_5$	0.1		$X_2$	0			



# Threshold algorithm

1. At each sequential access
  - d. When the scores of the top-k are greater or equal to the threshold, stop

$R_1$			$R_2$			$R_3$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1
$X_5$	0.1		$X_5$	0.1		$X_2$	0

$t = 2.1$

$X_3$	1.8
$X_2$	1.6

# Threshold algorithm

1. At each sequential access
  - d. When the scores of the top-k are greater or equal to the threshold, stop

$R_1$			$R_2$			$R_3$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1
$X_5$	0.1		$X_5$	0.1		$X_2$	0

$t = 1.0$

$X_3$	1.8
$X_2$	1.6

# Threshold algorithm

2. Return the top-k seen so far

$R_1$			$R_2$			$R_3$	
$X_1$	1		$X_2$	0.8		$X_4$	0.8
$X_2$	0.8		$X_3$	0.7		$X_3$	0.6
$X_3$	0.5		$X_1$	0.3		$X_1$	0.2
$X_4$	0.3		$X_4$	0.2		$X_5$	0.1
$X_5$	0.1		$X_5$	0.1		$X_2$	0

$t = 1.0$

$X_3$	1.8
$X_2$	1.6

# Threshold algorithm

- From the monotonicity property for any object not seen, the score of the object is less than the threshold
  - $f(X_5) \leq t \leq f(X_2)$
- The algorithm is **instance cost-optimal**
  - within a constant factor of the best algorithm on any database

# Combining rankings

- In many cases the scores are not known
  - e.g. meta-search engines – scores are proprietary information
- ... or we do not know how they were obtained
  - one search engine returns score 10, the other 100. What does this mean?
- ... or the scores are incompatible
  - apples and oranges: does it make sense to combine price with distance?
- In this cases we can only work with the rankings

# The problem

- Input: a set of rankings  $R_1, R_2, \dots, R_m$  of the objects  $X_1, X_2, \dots, X_n$ . Each ranking  $R_i$  is a **total ordering** of the objects
  - for every pair  $X_i, X_j$  either  $X_i$  is ranked above  $X_j$  or  $X_j$  is ranked above  $X_i$
- Output: A total ordering  $R$  that **aggregates** rankings  $R_1, R_2, \dots, R_m$

# Voting theory

- A voting system is a rank aggregation mechanism
- Long history and literature
  - criteria and axioms for good voting systems

# What is a good voting system?

- The **Condorcet criterion**
  - if object **A** defeats every other object in a pairwise majority vote, then **A** should be ranked first
- **Extended Condorcet criterion**
  - if the objects in a **set** X defeat in pairwise comparisons the objects in the set Y then the objects in X should be ranked above those in Y
- Not all voting systems satisfy the Condorcet criterion!



# Pairwise majority comparisons

- Unfortunately the Condorcet winner does not always exist
  - irrational behavior of groups

	$V_1$	$V_2$	$V_3$
1	A	B	C
2	B	C	A
3	C	A	B

A > B

B > C

C > A

# Pairwise majority comparisons

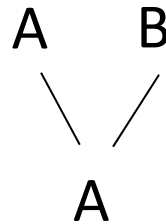
- Resolve cycles by imposing an agenda

	$V_1$	$V_2$	$V_3$
1	A	D	E
2	B	E	A
3	C	A	B
4	D	B	C
5	E	C	D

# Pairwise majority comparisons

- Resolve cycles by imposing an agenda

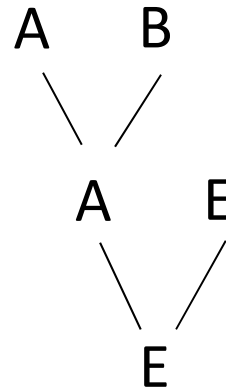
	$V_1$	$V_2$	$V_3$
1	A	D	E
2	B	E	A
3	C	A	B
4	D	B	C
5	E	C	D



# Pairwise majority comparisons

- Resolve cycles by imposing an agenda

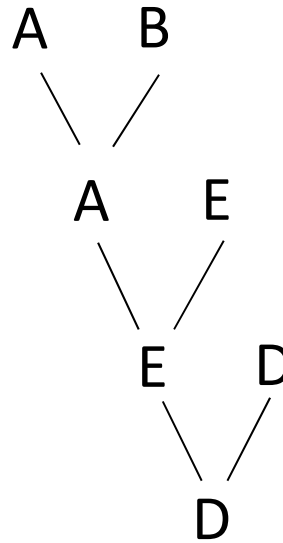
	$V_1$	$V_2$	$V_3$
1	A	D	E
2	B	E	A
3	C	A	B
4	D	B	C
5	E	C	D



# Pairwise majority comparisons

- Resolve cycles by imposing an agenda

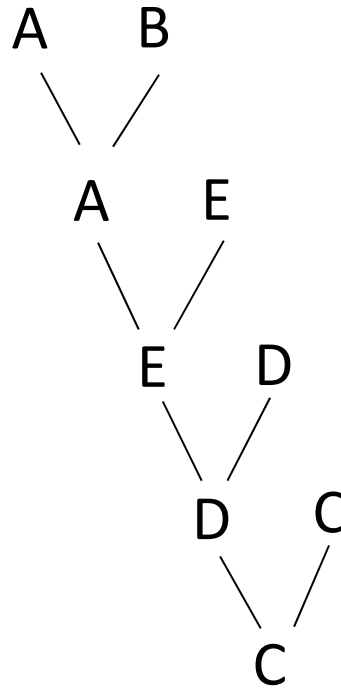
	$V_1$	$V_2$	$V_3$
1	A	D	E
2	B	E	A
3	C	A	B
4	D	B	C
5	E	C	D



# Pairwise majority comparisons

- Resolve cycles by imposing an agenda

	$V_1$	$V_2$	$V_3$
1	A	D	E
2	B	E	A
3	C	A	B
4	D	B	C
5	E	C	D

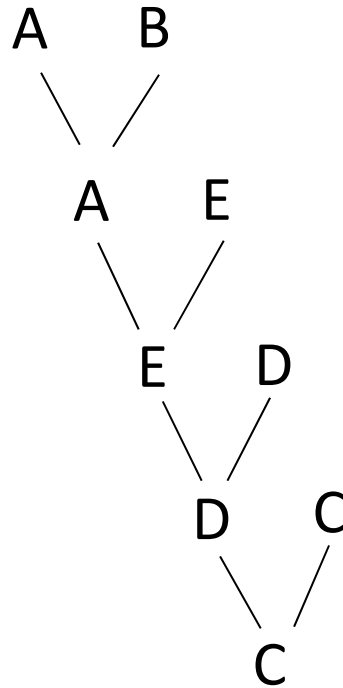


- C is the winner

# Pairwise majority comparisons

- Resolve cycles by imposing an agenda

	$V_1$	$V_2$	$V_3$
1	A	D	E
2	B	E	A
3	C	A	B
4	D	B	C
5	E	C	D



- But everybody prefers A or B over C

# Pairwise majority comparisons

- The voting system is not **Pareto optimal**
  - there exists another ordering that everybody prefers
- Also, it is sensitive to the order of voting



# Plurality vote

- Elect first whoever has more 1st position votes

voters	10	8	7
1	A	C	B
2	B	A	C
3	C	B	A

- Does not find a Condorcet winner (C in this case)

# Plurality with runoff

- If no-one gets more than 50% of the 1st position votes, take the majority winner of the first two

voters	10	8	7	2
1	A	C	B	B
2	B	A	C	A
3	C	B	A	C

first round: A 10, B 9, C 8

second round: A 18, B 9

winner: A

# Plurality with runoff

- If no-one gets more than 50% of the 1st position votes, take the majority winner of the first two

voters	10	8	7	2
1	A	C	B	A
2	B	A	C	B
3	C	B	A	C

change the order of  
A and B in the last  
column

first round: A 12, B 7, C 8

second round: A 12, C 15

winner: C!

# Positive Association axiom

- Plurality with runoff violates the **positive association axiom**
- **Positive association axiom**: positive changes in preferences for an object should not cause the ranking of the object to decrease

# Borda Count

- For each ranking, assign to object  $X$ , number of points equal to the number of objects it defeats
  - first position gets  $n-1$  points, second  $n-2$ , ..., last  $0$  points
- The total weight of  $X$  is the number of points it accumulates from all rankings

# Borda Count

voters	3	2	2
1 (3p)	A	B	C
2 (2p)	B	C	D
3 (1p)	C	D	A
4 (0p)	D	A	B

$$A: 3*3 + 2*0 + 2*1 = 11p$$

$$B: 3*2 + 2*3 + 2*0 = 12p$$

$$C: 3*1 + 2*2 + 2*3 = 13p$$

$$D: 3*0 + 2*1 + 2*2 = 6p$$

BC
C
B
A
D

- Does not always produce Condorcet winner

# Borda Count

- Assume that D is removed from the vote

voters	3	2	2
1 (2p)	A	B	C
2 (1p)	B	C	A
3 (0p)	C	A	B

$$A: 3*2 + 2*0 + 2*1 = 7p$$

$$B: 3*1 + 2*2 + 2*0 = 7p$$

$$C: 3*0 + 2*1 + 2*2 = 6p$$

BC
B
A
C

- Changing the position of D changes the order of the other elements!

# Independence of Irrelevant Alternatives

- The relative ranking of  $X$  and  $Y$  should not depend on a third object  $Z$ 
  - heavily debated axiom



# Borda Count

- The Borda Count of an object  $X$  is the aggregate number of pairwise comparisons that the object  $X$  wins
  - follows from the fact that in one ranking  $X$  wins all the pairwise comparisons with objects that are under  $X$  in the ranking

# Voting Theory

- Is there a voting system that does not suffer from the previous shortcomings?

# Arrow's Impossibility Theorem

- There is no voting system that satisfies the following axioms
  - Universality
    - all inputs are possible
  - Completeness and Transitivity
    - for each input we produce an answer and it is meaningful
  - Positive Association
    - Promotion of a certain option cannot lead to a worse ranking of this option.
  - Independence of Irrelevant Alternatives
    - Changes in individuals' rankings of irrelevant alternatives (ones outside a certain subset) should have no impact on the societal ranking of the subset.
  - Non-imposition
    - Every possible societal preference order should be achievable by some set of individual preference orders
  - Non-dictatorship
- **KENNETH J. ARROW** *Social Choice and Individual Values* (1951). Won Nobel Prize in 1972

# Kemeny Optimal Aggregation

- Kemeny distance  $K(R_1, R_2)$ : The number of pairs of nodes that are ranked in a different order (Kendall-tau)
  - number of bubble-sort swaps required to transform one ranking into another
- Kemeny optimal aggregation minimizes

$$K(R, R_1, \dots, R_m) = \sum_{i=1}^m K(R, R_i)$$

- Kemeny optimal aggregation satisfies the Condorcet criterion and the extended Condorcet criterion
  - maximum likelihood interpretation: produces the ranking that is most likely to have generated the observed rankings
- ...but it is NP-hard to compute
  - easy 2-approximation by obtaining the best of the input rankings, but it is not “interesting”

# Locally Kemeny optimal aggregation

- A ranking  $R$  is **locally Kemeny optimal** if there is no bubble-sort swap that produces a ranking  $R'$  such that  $K(R', R_1, \dots, R_m) \leq K(R, R_1, \dots, R_m)$
- Locally Kemeny optimal is not necessarily Kemeny optimal
- Definitions apply for the case of partial lists also

# Locally Kemeny optimal aggregation

- Locally Kemeny optimal aggregation can be computed in polynomial time
  - At the  $i$ -th iteration insert the  $i$ -th element  $x$  in the bottom of the list, and bubble it up until there is an element  $y$  such that the majority places  $y$  over  $x$
- Locally Kemeny optimal aggregation satisfies the Condorcet and extended Condorcet criterion

# Rank Aggregation algorithm [DKNS01]

- Start with an aggregated ranking and make it into a locally Kemeny optimal aggregation
- How do we select the initial aggregation?
  - Use another aggregation method
  - Create a Markov Chain where you move from an object  $X$ , to another object  $Y$  that is ranked higher by the majority

# Spearman's footrule distance

- Spearman's footrule distance: The difference between the ranks  $R(i)$  and  $R'(i)$  assigned to object  $i$

$$F(R, R') = \sum_{i=1}^n |R(i) - R'(i)|$$

- Relation between Spearman's footrule and Kemeny distance

$$K(R, R') \leq F(R, R') \leq 2K(R, R')$$



# Spearman's footrule aggregation

- Find the ranking  $R$ , that minimizes

$$F(R, R_1, \dots, R_m) = \sum_{i=1}^m F(R, R_i)$$

- The optimal Spearman's footrule aggregation can be computed in polynomial time
  - It also gives a 2-approximation to the Kemeny optimal aggregation
- If the median ranks of the objects are unique then this ordering is optimal

# Example

$R_1$	
1	A
2	B
3	C
4	D

$R_2$	
1	B
2	A
3	D
4	C

$R_3$	
1	B
2	C
3	A
4	D

$R$	
1	B
2	A
3	C
4	D

A: ( 1, 2, 3 )  
B: ( 1, 1, 2 )  
C: ( 3, 3, 4 )  
D: ( 3, 4, 4 )

# The MedRank algorithm

- Access the rankings sequentially

$R_1$	
1	A
2	B
3	C
4	D

$R_2$	
1	B
2	A
3	D
4	C

$R_3$	
1	B
2	C
3	A
4	D

$R$	
1	
2	
3	
4	

# The MedRank algorithm

- Access the rankings sequentially
  - when an element has appeared in more than half of the rankings, output it in the aggregated ranking

$R_1$		$R_2$		$R_3$		$R$	
1	A	1	B	1	B	1	B
2	B	2	A	2	C	2	
3	C	3	D	3	A	3	
4	D	4	C	4	D	4	

# The MedRank algorithm

- Access the rankings sequentially
  - when an element has appeared in more than half of the rankings, output it in the aggregated ranking

$R_1$			$R_2$			$R_3$			$R$	
1	A		1	B		1	B		1	B
2	B		2	A		2	C		2	A
3	C		3	D		3	A		3	
4	D		4	C		4	D		4	

# The MedRank algorithm

- Access the rankings sequentially
  - when an element has appeared in more than half of the rankings, output it in the aggregated ranking

$R_1$			$R_2$			$R_3$			$R$	
1	A		1	B		1	B		1	B
2	B		2	A		2	C		2	A
3	C		3	D		3	A		3	C
4	D		4	C		4	D		4	

# The MedRank algorithm

- Access the rankings sequentially
  - when an element has appeared in more than half of the rankings, output it in the aggregated ranking

$R_1$			$R_2$			$R_3$			$R$	
1	A		1	B		1	B		1	B
2	B		2	A		2	C		2	A
3	C		3	D		3	A		3	C
4	D		4	C		4	D		4	D

# The Spearman's rank correlation

- Spearman's rank correlation

$$S(R, R') = \sum_{i=1}^n (R(i) - R'(i))^2$$

- Computing the optimal rank aggregation with respect to Spearman's rank correlation is the same as computing Borda Count
  - Computable in polynomial time



# Extensions and Applications

- Rank distance measures between partial orderings and top-k lists
- Similarity search
- Ranked Join Indices
- Analysis of Link Analysis Ranking algorithms
- Connections with machine learning

# References

- A. Borodin, G. Roberts, J. Rosenthal, P. Tsaparas, [Link Analysis Ranking: Algorithms, Theory and Experiments](#), ACM Transactions on Internet Technologies (TOIT), 5(1), 2005
- Ron Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, Erik Vee, [Comparing and aggregating rankings with ties](#) , PODS 2004
- M. Tennenholtz, and Alon Altman, "[On the Axiomatic Foundations of Ranking Systems](#)", Proceedings of IJCAI, 2005
- Ron Fagin, Amnon Lotem, Moni Naor. [Optimal aggregation algorithms for middleware](#), J. Computer and System Sciences 66 (2003), pp. 614-656. Extended abstract appeared in Proc. 2001 ACM Symposium on Principles of Database Systems (PODS '01), pp. 102-113.
- Alex Tabbarok [Lecture Notes](#)
- Ron Fagin, Ravi Kumar, D. Sivakumar [Efficient similarity search and classification via rank aggregation](#), Proc. 2003 ACM SIGMOD Conference (SIGMOD '03), pp. 301-312.
- Cynthia Dwork, Ravi Kumar, Moni Naor, D. Sivakumar. [Rank Aggregation Methods for the Web](#). 10th International World Wide Web Conference, May 2001.
- C. Dwork, R. Kumar, M. Naor, D. Sivakumar, "[Rank Aggregation Revisited](#)," WWW10; selected as Web Search Area highlight, 2001.