Analysis of massive data sets

http://www.fer.hr/predmet/avsp

Prof. dr. sc. Siniša Srbljić

Doc. dr. sc. Dejan Škvorc

Doc. dr. sc. Ante Đerek

Faculty of Electrical Engineering and Computing
Consumer Computing Laboratory

Analysis of Massive Data Sets: Mining Large Graphs – Link Analysis

Marin Šilić, PhD

Overview

- Motivation
- Link Analysis
- Flow Formulation
- Matrix Formulation
- Power Iteration Method
- Google Formulation
- Implementation Details

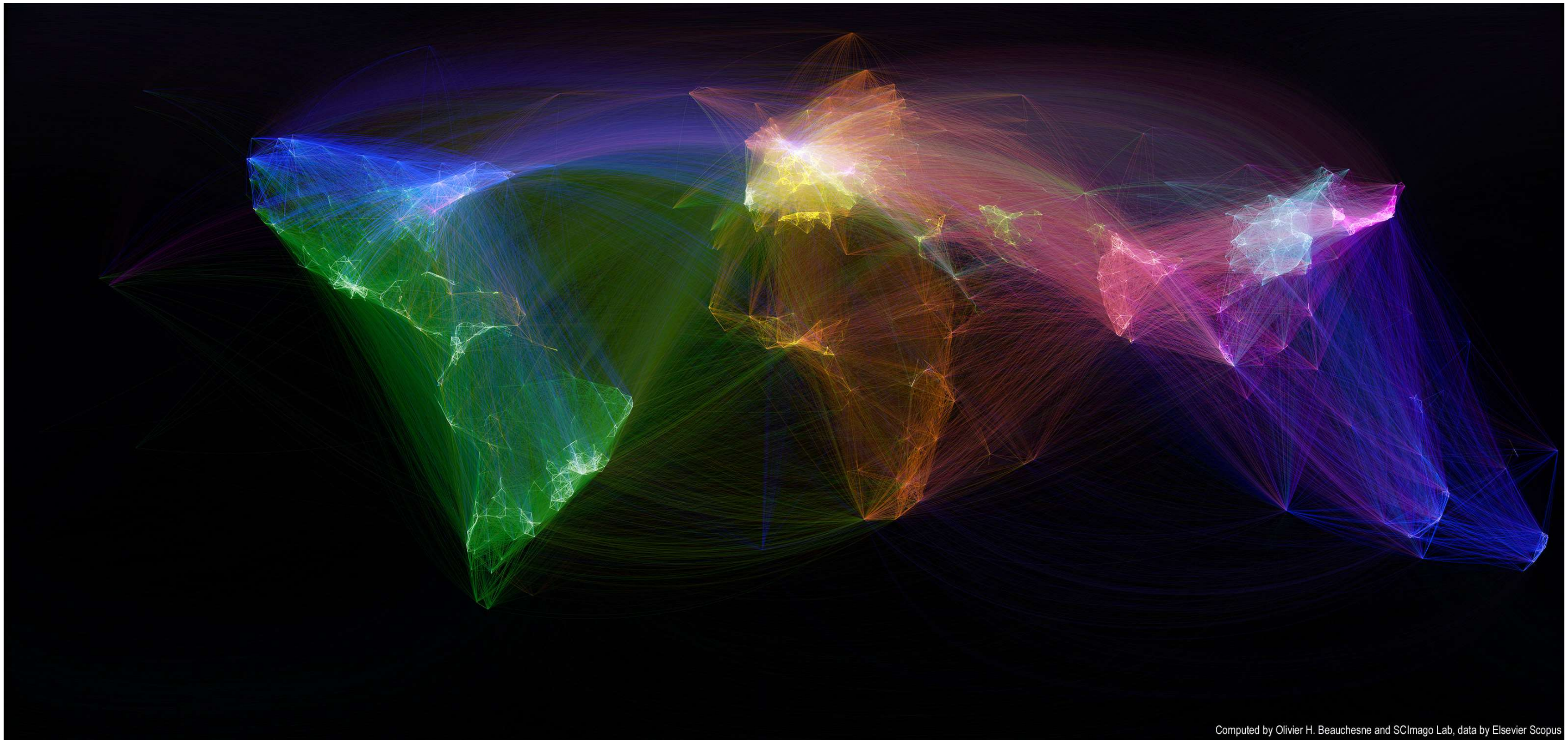
Motivation

- **Data Naturally Represented as Graph**
  - Social Networks

Facebook's Social Network Graph: Paul Butler, intern at Facebook, Visualization of the locality of friendship, 2010

Motivation

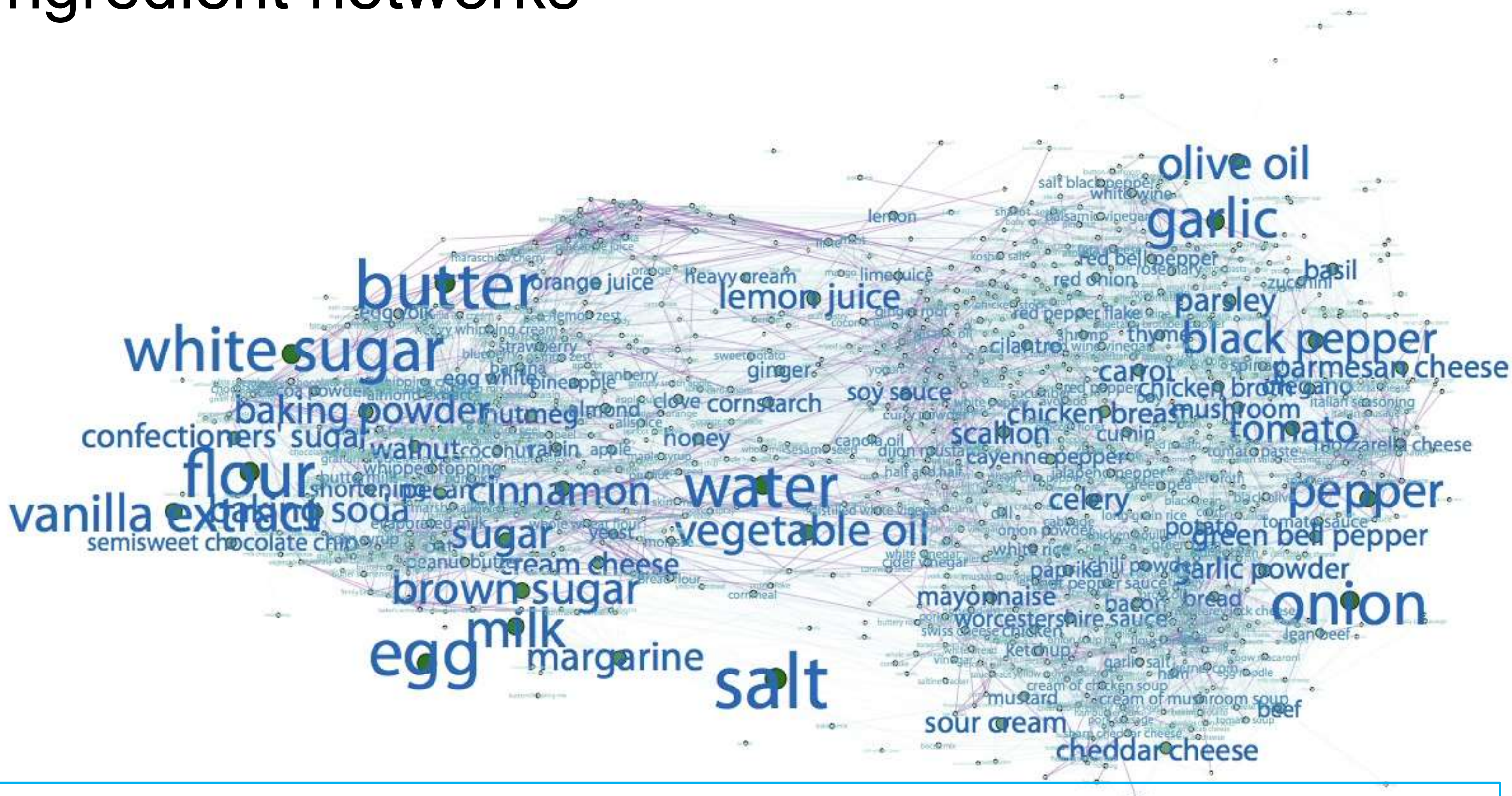
- **Data Naturally Represented as Graph**
  - Map of scientific collaboration



http://olihb.com/2011/01/23/map-of-scientific-collaboration-between-researchers/

Motivation

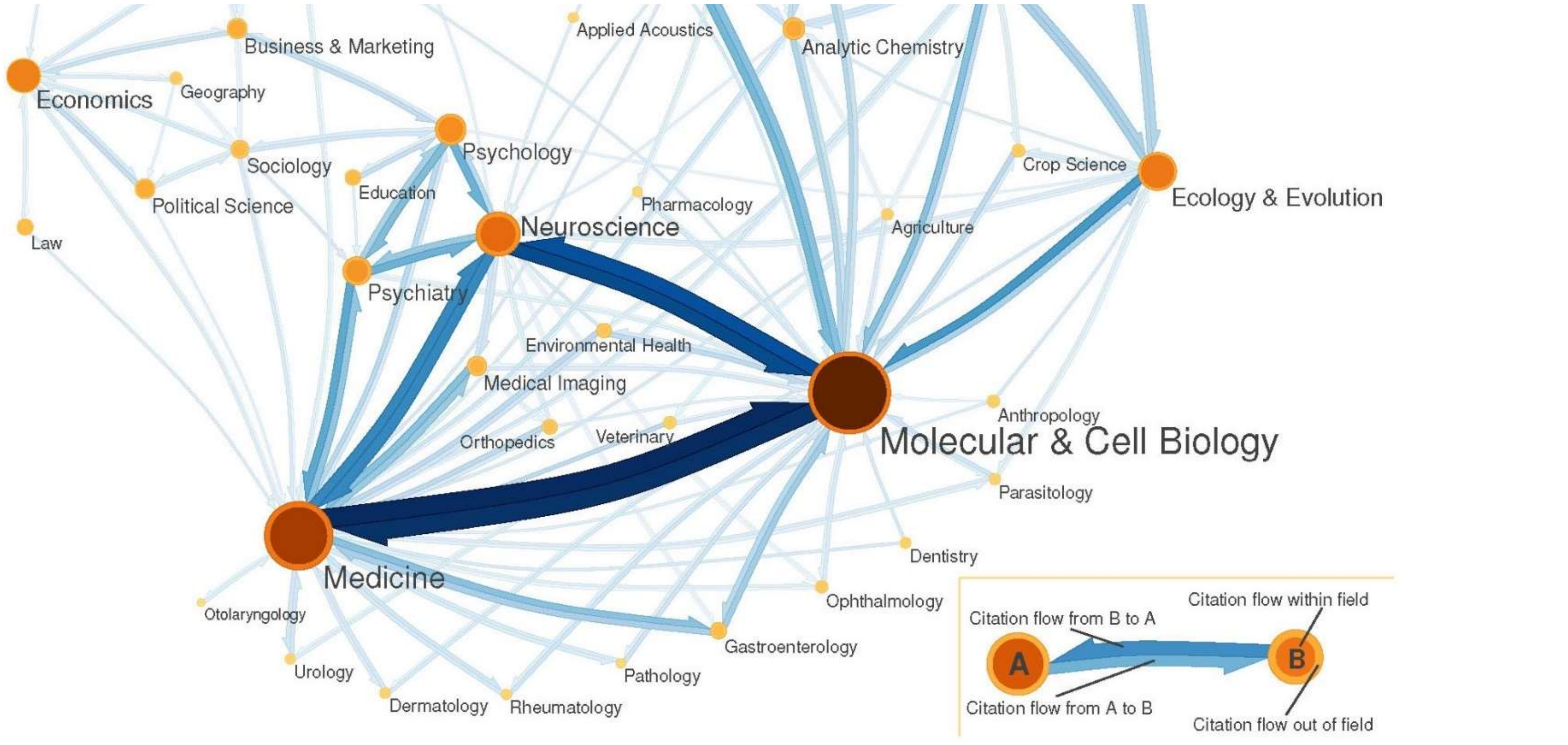
- **Data Naturally Represented as Graph**
  - Ingredient networks



Teng, Chun-Yuen, Yu-Ru Lin, and Lada A. Adamic.
"Recipe recommendation using ingredient networks."
Proceedings of the 4th Annual ACM Web Science Conference. ACM, 2012.

Motivation

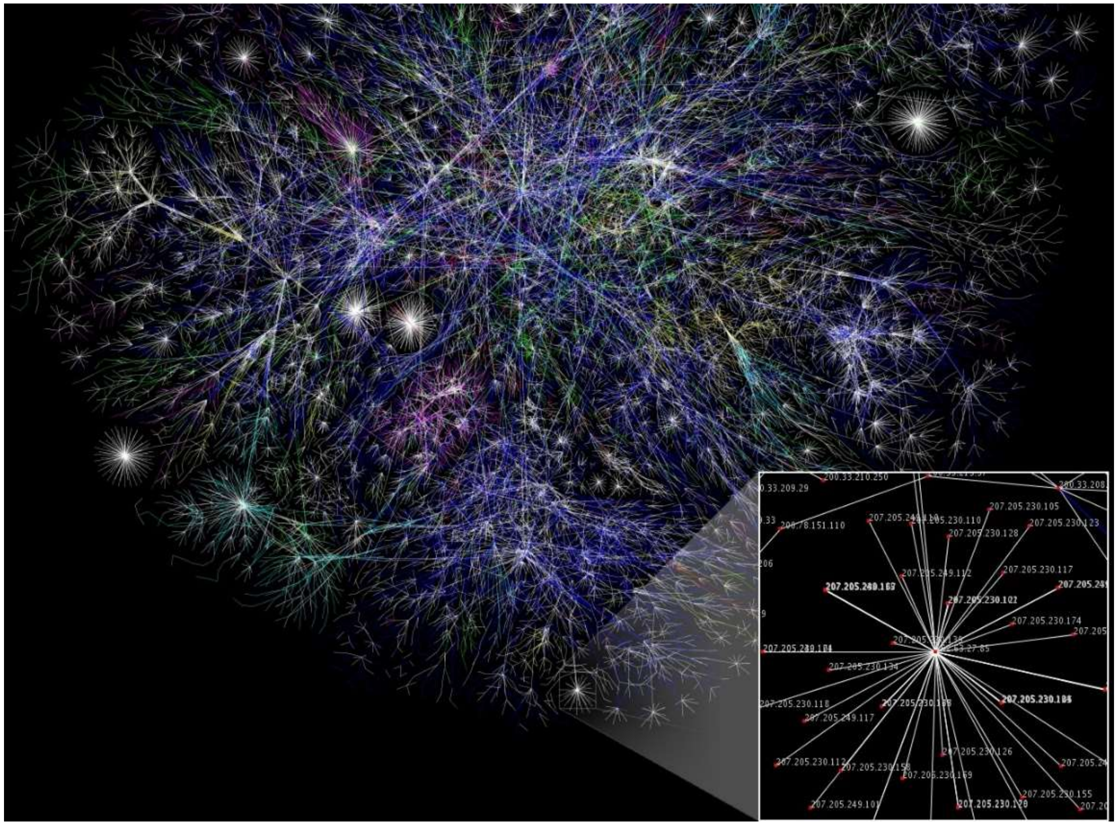
- **Data Naturally Represented as Graph**
  - The science citation map



West, Jevin D. *Eigenfactor: ranking and mapping scientific knowledge*. Diss. University of Washington, 2010.

Motivation

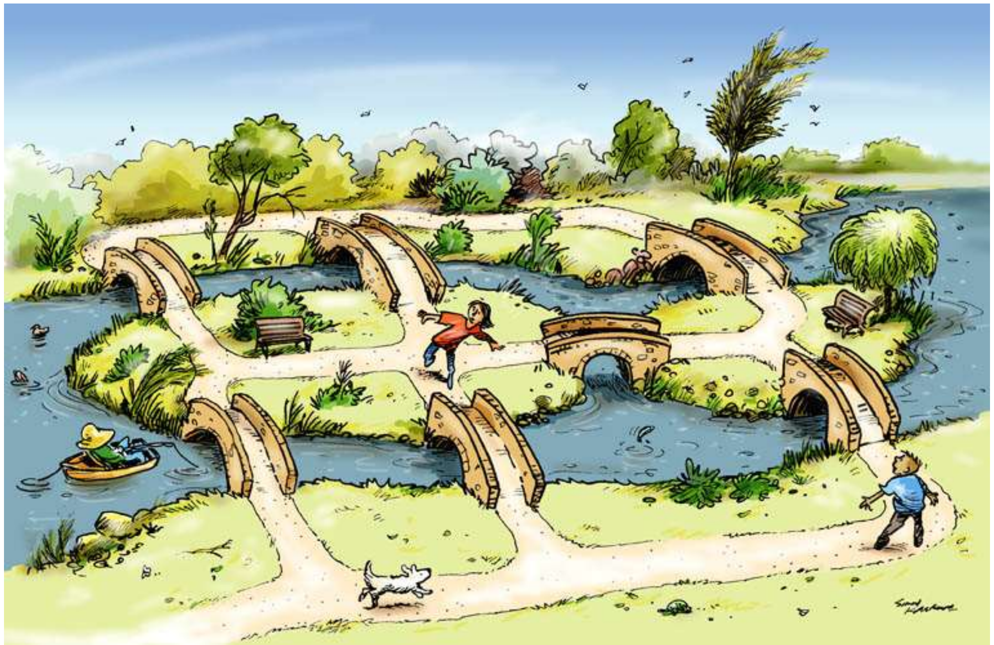
- **Data Naturally Represented as Graph**
  - The map of Internet in 2005 (Wikipedia)



Partial map of the Internet based on the January 15, 2005 data found on opte.org. Each line is drawn between two nodes, representing two IP addresses.

Motivation

- **Data Naturally Represented as Graph**
  - Topological networks
    - Power supply network, water supply network, transportation…
    - Seven bridges of Konigsberg (Euler's path) puzzle.



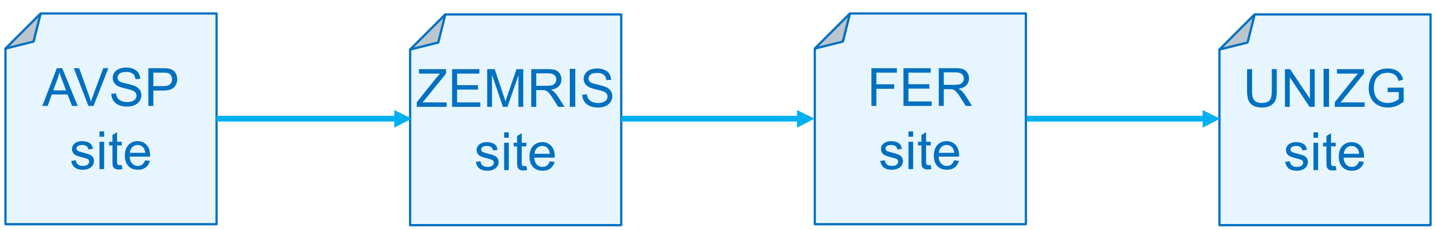
Is Euler's path is possible?

The answer is NO.

It depends on the degrees of nodes. The graph should have exactly zero or two nodes with an odd degree.

Link Analysis

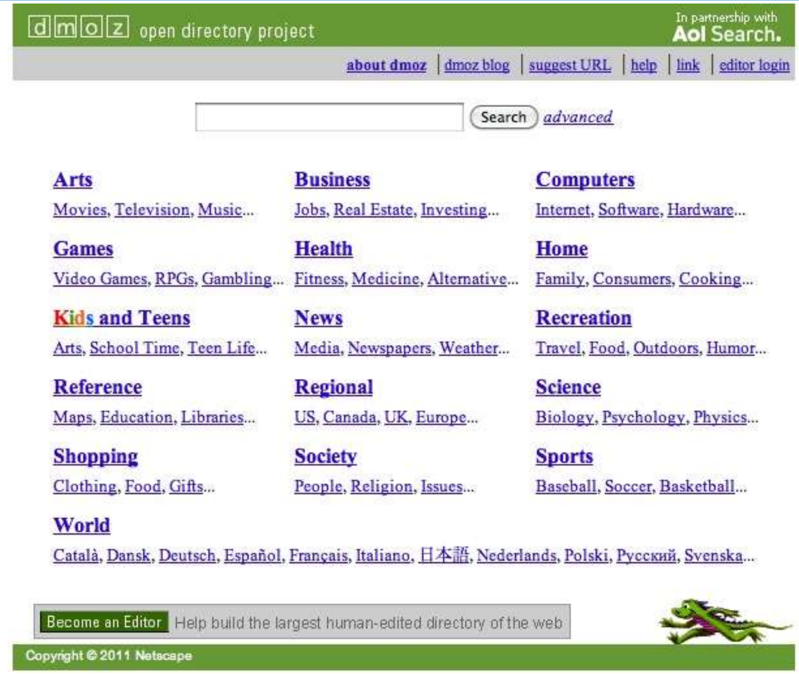
- **Imagine the Web as a Graph**
  - Directed graph
    - Nodes: Websites
    - Edges: Link among websites – Hyperlinks



Copyright © 2016 CCL: Analysis of massive data sets

Link Analysis

- **How to organize the Web?**
  - First try: Human organized
    - Web directories
      - Yahoo, DMOZ, LookSmart
    - Put each new website in its category
    - Issue: Web is huge, the approach does not scale!!
  - Second try: Web Search
    - Information Retrieval
      - Find relevant docs in a small and trusted subset
      - Newspapers articles and similar trusted data corpus
    - Issue: Again, web is huge, plenty of untrusted documents, random things, web spam, etc.



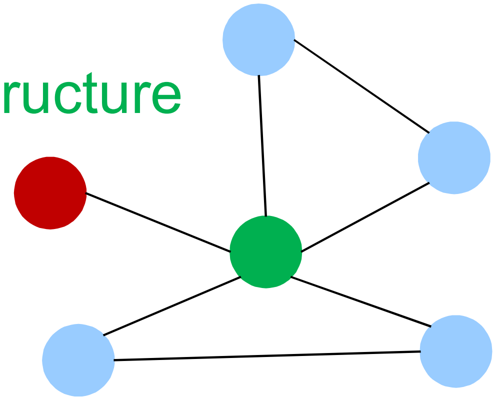
Link Analysis

- **Two main challenges of Web Search**
  - There are plenty of information sources on the web
    Who do we trust?
    - Idea: Use structure of the web graphs
    - Trusted pages are likely to point to each other!
  - What is the best (appropriate) answer to query "media"
    - There is an ambiguity in this query, no single right answer
    - Idea: Again, utilize structure of the web
    - The pages that actually know about media might all be pointing to many media sites

Link Analysis

- **Ranking Nodes on the Graph**
  - All web pages are not equally important
    - www.johndoe.com
    - www.fer.hr
  - The structure of the web graph tells something
    - There is a large diversity when considering node connectivity
    - The idea: Lets rank pages by the link structure

The green node is more important than the red node



Link Analysis

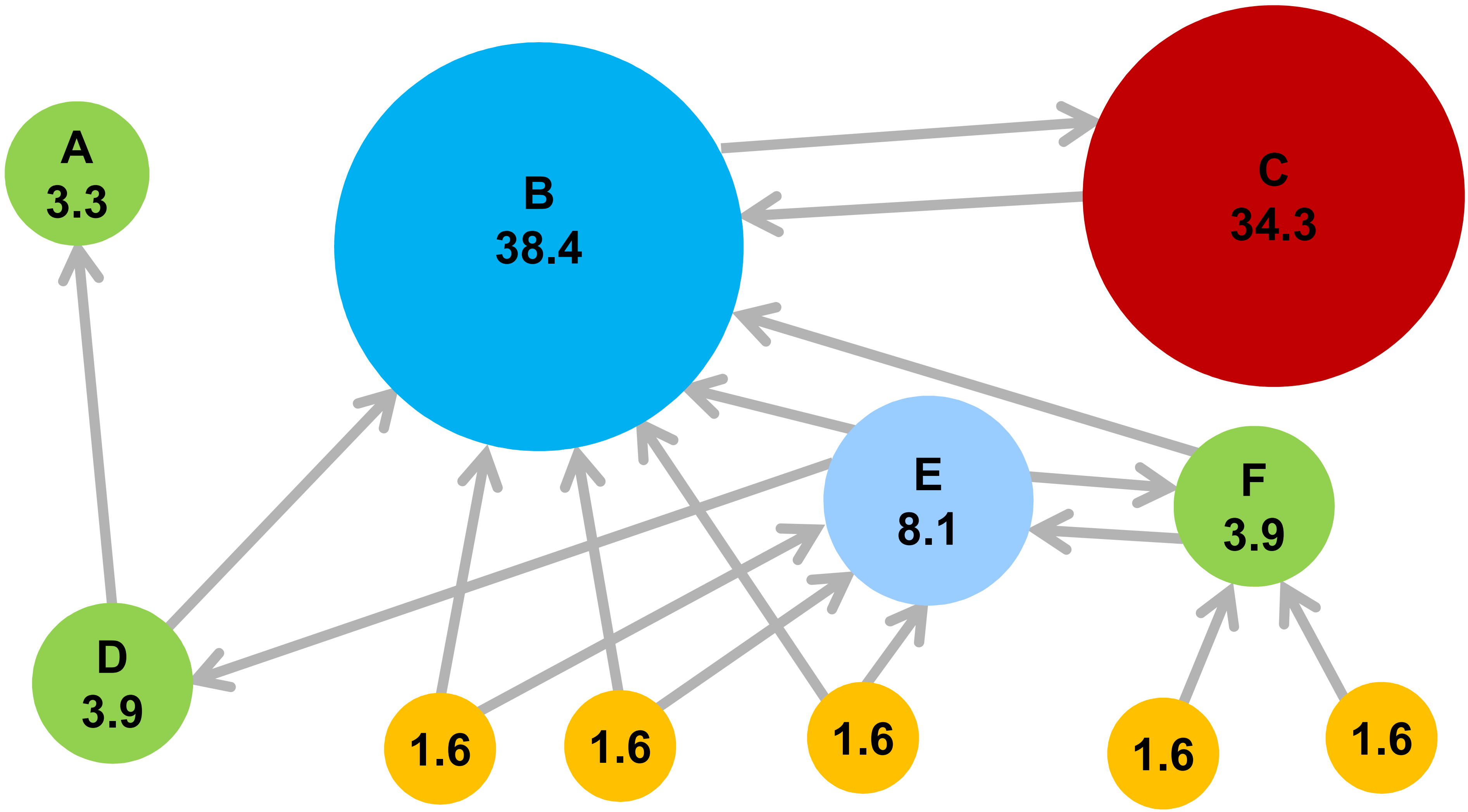
- **Link Analysis Algorithms**
  - Computing the importance of the nodes in the graph
  - Various solutions to the Link Analysis problem
    - Page Rank
    - Topic-Specific (Personalized) Page Rank
    - Web Spam Detection Algorithms

Flow Formulation

- **Count Node's Links**
  - Idea: Utilize graph structure
    Count node links as votes
    - Incoming links vs Outgoing links
  - Imagine incoming links as votes
    - www.fer.hr has 24622 incoming links
    - www.johndoe.com has 0 incoming links
  - However, all incoming links are not equally important
    - Links originating from important pages are worth more
    - The nature of the problem is recursive

Flow Formulation

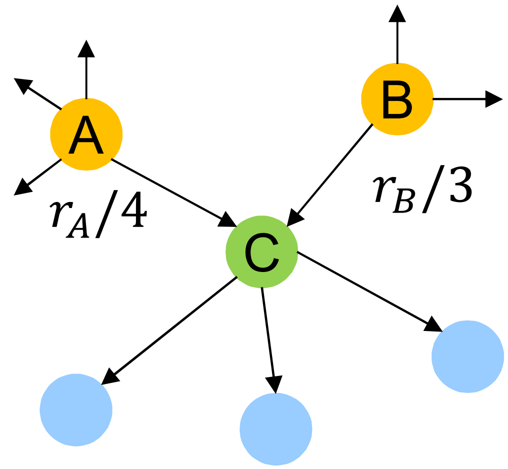
- Graph Example: PageRank Scores



Flow Formulation

- **Page Rank: Recursive Formulation**
  - Idea: Incoming link's value is proportional to the importance of its source node
  - If node j with importance r_j has n outgoing links
    - Each links gets r_j/n votes
  - Node j's importance is equal to the sum of the votes of its incoming links

$$r_C = \frac{r_A}{4} + \frac{r_B}{3}$$

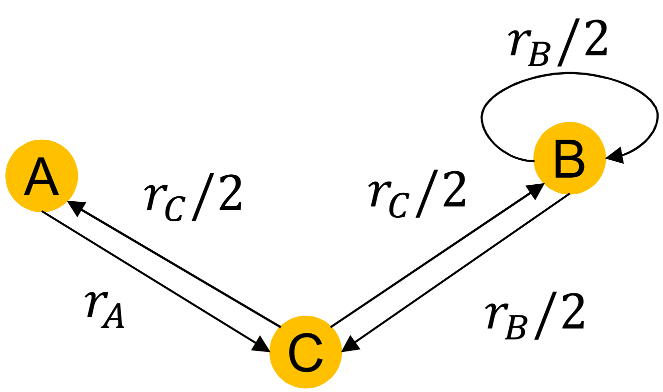


Flow Formulation

- **Page Rank: The "Flow" Model**
  - A link (vote) from an important page is worth more
  - A node is important if it is linked by other important nodes
  - Rank definition r_j for node j:

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

d_i is the out-degree of the node i



"Flow" equations:

$$r_A = \frac{r_C}{2}$$

$$r_B = \frac{r_B}{2} + \frac{r_C}{2}$$

$$r_C = r_A + \frac{r_B}{2}$$

Flow Formulation

- **Page Rank: The "Flow" Model**
  - 3 equations, 3 unknowns, no constants
    - There is no unique solution
  - However, there is an additional constraint
    - We force the uniqueness
    - The sum of all node ranks sums to 1
  - Unique solution $r_A = \frac{1}{5}, r_B = \frac{2}{5}, r_C = \frac{2}{5}$
  - Gauss elimination method works for small graphs
    - $O(N^3)$, it does not scale already for $N = 10^4$
    - We need some other solution

"Flow" equations:

$$r_A = \frac{r_C}{2}$$
$$r_B = \frac{r_B}{2} + \frac{r_C}{2}$$
$$r_C = r_A + \frac{r_B}{2}$$

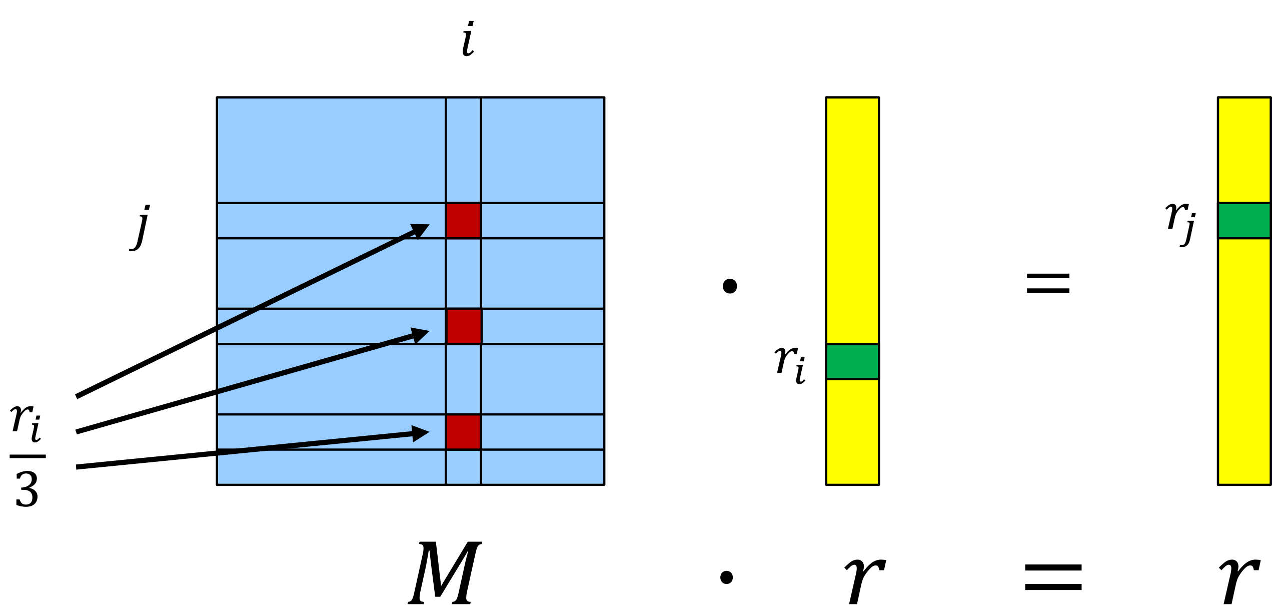
Matrix Formulation

- **Write Flow Formulation in Matrix M**
  - Stochastic Adjacency Matrix M
  - If $i \rightarrow j$, then $M_{ji} = 1/d_i$, else $M_{ji} = 0$
  - M is a column stochastic matrix
    - Columns of the matrix sum to 1
  - Rank vector r: vector with an entry per page
    - r_i is the importance score of node i
    - $\sum_i r_i = 1$
  - The flow equation can be written as: $$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$$

$$r = M \cdot r$$

Matrix Formulation

- **Example**
  - Flow equation $$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$
  - Matrix formulation $$M \cdot r = r$$
    - Page i links to 3 pages, including j



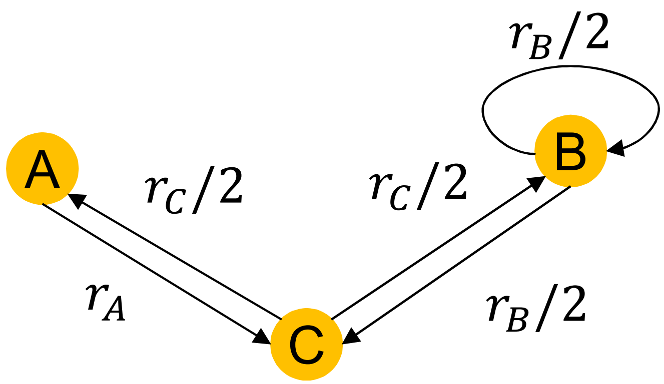
Copyright © 2016 CCL: Analysis of massive data sets

Matrix Formulation

- **Flow Equation:** $M \cdot r = r$
- **Eigenvector:** $A \cdot v = \lambda \cdot v$
  - v is an eigenvector of matrix A, with the corresponding eigenvalue λ
  - Hence, the rank vector r is an eigenvector of the stochastic matrix M
    - Its corresponding eigenvalue is 1
    - The largest (dominant) eigenvalue of M is 1
      - M is column stochastic (each column sums to 1, non-zero entries)
      - r is unit length (non-zero entries sum to 1)
- **Power Iteration Method**
  - Algorithm for finding the dominant eigenvector

Matrix Formulation

- **Back to our Example…**



"Flow" equations:

$$r_A = \frac{r_C}{2}$$
$$r_B = \frac{r_B}{2} + \frac{r_C}{2}$$
$$r_C = r_A + \frac{r_B}{2}$$

$$\underbrace{\begin{array}{c|ccc} & A & B & C \\ \hline A & 0 & 0 & 1/2 \\ B & 0 & 1/2 & 1/2 \\ C & 1 & 1/2 & 0 \end{array}}_{M}$$

$$\underbrace{\begin{bmatrix} r_A \\ r_B \\ r_C \end{bmatrix}}_{r} = \underbrace{\begin{bmatrix} 0 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \\ 1 & 1/2 & 0 \end{bmatrix}}_{M} \cdot \underbrace{\begin{bmatrix} r_A \\ r_B \\ r_C \end{bmatrix}}_{r}$$

Copyright © 2016 CCL: Analysis of massive data sets

Power Iteration Method

- **Input: Graph with N nodes**
- **Power Iteration: iterative algorithm**
  - There are N nodes
  - Initialize: $r^{(0)} = \left[\frac{1}{N}, \ldots, \frac{1}{N}\right]^T$
  - Iteratively compute: $r^{(t+1)} = M \cdot r^{(t)}$
  - Until: $|\, r^{(t+1)} - r^{(t)} |_1 > \epsilon$
    - $|x|_1 = \sum_{1 \le i \le N} |x_i|$, L_1 norm
    - Some other vector norm can be used, for instance Euclidean

Power Iteration Method

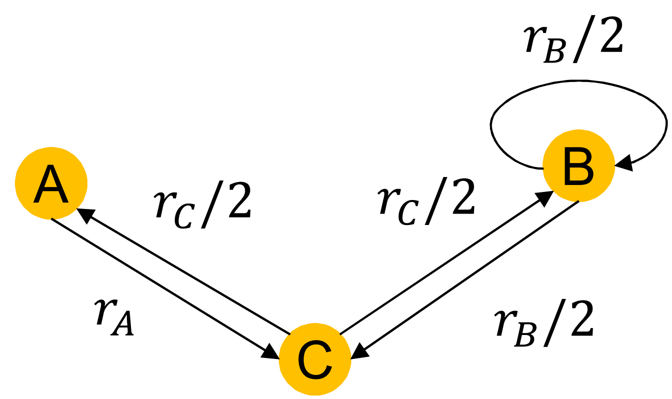
Power Iteration Example

- Algorithm

for $j = 1$ **to** N
$$r_j := 1/N$$

do
$$r_j' := \sum_{i \to j} \frac{r_i}{d_i}$$

while $(r_j' \neq r_j)$



"Flow" equations:

$$r_A = \frac{r_C}{2}$$
$$r_B = \frac{r_B}{2} + \frac{r_C}{2}$$
$$r_C = r_A + \frac{r_B}{2}$$

- Results:

$$\begin{bmatrix} r_A \\ r_B \\ r_C \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \quad \begin{bmatrix} 1/6 \\ 2/6 \\ 3/6 \end{bmatrix} \quad \begin{bmatrix} 3/12 \\ 5/12 \\ 4/12 \end{bmatrix} \quad \ldots \quad \begin{bmatrix} 3/15 \\ 6/15 \\ 6/15 \end{bmatrix}$$

Iteration 0 1 2 …

Power Iteration Method

- **Power Iteration Proof**
  - Method for finding the dominant eigenvector
    - $r^{(1)} = M \cdot r^{(0)}$
    - $r^{(2)} = M \cdot r^{(1)} = M \cdot \left(M \cdot r^{(0)}\right) = M^2 \cdot r^{(0)}$
    - $r^{(k)} = M \cdot r^{(k-1)} = M \cdot \left(M^{k-1} \cdot r^{(0)}\right) = M^k \cdot r^{(0)}$
  - Claim:
    - $M \cdot r^{(0)}, M^2 \cdot r^{(0)}, \ldots, M^k \cdot r^{(0)}$ converges to the dominant eigenvector
    - We assume:
      - M has eigenvalues such that $\lambda_1 > \lambda_2 > \ldots > \lambda_n$
      - M has n linearly independent eigenvectors $x_1, x_2, \ldots, x_n$, such that $A\mathrm{x}_i = \lambda_i x_i$

Power Iteration Method

- **Power Iteration Proof**
  - Proof:
    - Since x_i are linearly independent, we can write:
    - $r^{(0)} = c_1 x_1 + c_2 x_2 + \dots + c_n x_n$
    - $M r^{(0)} = M(c_1 x_1 + c_2 x_2 + \dots + c_n x_n)$
    - $M r^{(0)} = c_1 (M x_1) + c_2 (M x_2) + \dots + c_n (M x_n)$
    - $M r^{(0)} = c_1 (\lambda x_1) + c_2 (\lambda x_2) + \dots + c_n (\lambda x_n)$
    - Repeating algorithm iteration produces:
    - $M^{(k)} r^{(0)} = c_1 (\lambda^{(k)} x_1) + c_2 \left(\lambda^{(k)} x_2\right) + \dots + c_n \left(\lambda^{(k)} x_n\right)$

Power Iteration Method

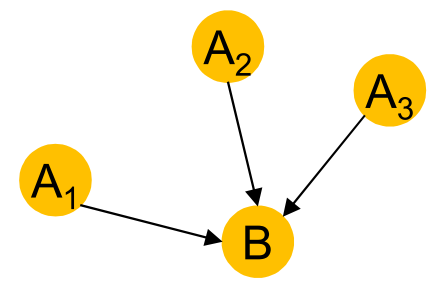
Power Iteration Proof

- Proof (continued):
  - $M^{(k)}r^{(0)} = c_1(\lambda^{(k)}x_1) + c_2(\lambda^{(k)}x_2) + \ldots + c_n(\lambda^{(k)}x_n)$
  - $M^{(k)}r^{(0)} = \lambda_1^{(k)}[c_1x_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^{(k)} x_2 + \ldots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^{(k)} x_n]$
  - Having $\lambda_1 > \lambda_i$ and $k \rightarrow \infty$, each $\left(\frac{\lambda_i}{\lambda_1}\right)^{(k)} = 0$, for all $i = 2, \ldots, n$
- Hence:
  - $M^{(k)}r^{(0)} \approx c_1(\lambda_1^{(k)}x_1)$
    - If c_1 is 0 ($r^{(0)}$ does not have a component in the direction of x_1), the method will not converge

Power Iteration Method

- **Random Walk Interpretation**
  - Imagine a random walker that explores a graph
    - At some time t, the walker is at some node A
    - At time $t+1$, the walker randomly follows some outgoing link from node A
    - The walker gets to some node B linked from node A
    - The random walk process repeats indefinitely

$$r_B = \sum_{A \to B} \frac{r_A}{d_A}$$

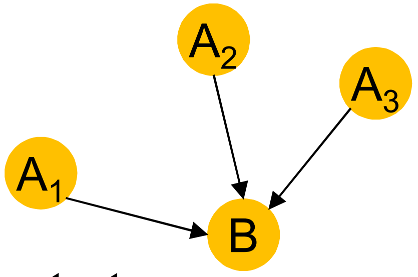


  - We define:
    - $p(t)$ as the vector whose i^{th} coordinates corresponds to the probability that the walker is at node i at time t
    - $p(t)$ is a probability distribution over nodes

Copyright © 2016 CCL: Analysis of massive data sets

Power Iteration Method

- **Random Walk Interpretation**
  - Having a graph structure and $p(t)$, where is the walker at time $p(t+1)$?
    - Follows the links uniformly at random
    - $p(t\ +\ 1) = M \cdot p(t)$



  - If at some point the walker reaches a state,
    - $p(t\ +\ 1) = M \cdot p(t)\ =\ p(t)$,
      then $p(t)$ is a stationary distribution of random walk
  - Rank vector r satisfies $r\ =\ M \cdot r$
    - Hence, r is stationary distribution for the random walk

Power Iteration Method

- **Theory of Random Walks**
  - Markov processes
    - Processes that are memoryless
    - One can make predictions for the future based solely on its present state
  - If the initial graph satisfies certain conditions, the stationary distribution is unique and it will be always reached
  - Thus, regardless what the initial distribution at time $t = 0$ is, the stationary distribution will be achieved

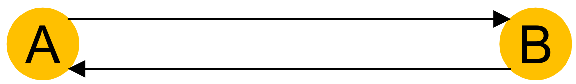
Copyright © 2016 CCL: Analysis of massive data sets

Google Formulation

- **So far:**
  - Flow equation: $$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$$
  - Matrix formulation: $$r = M \cdot r$$
  - Two questions:
    1) Does it converge?
    2) Does the solution makes sense?

Google Formulation

- Does it converge?



$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

  - Solution

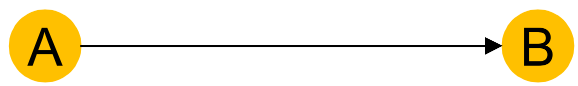
▪	$r_A =$	1	0	1	…
▪	$r_B =$	0	1	0	…
	Iteration	0	1	2	…

It will never converge!

Copyright © 2016 CCL: Analysis of massive data sets

Google Formulation

- **Does the solution makes sense?**



$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

  - Solution

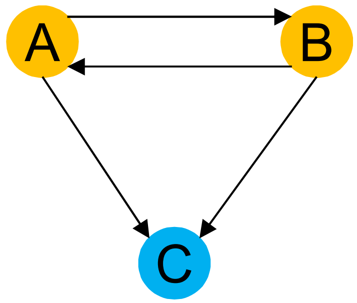
$r_A =$	1	0	0	…
$r_B =$	0	1	0	…
Iteration	0	1	2	…

It will converge to a meaningless result!

Copyright © 2016 CCL: Analysis of massive data sets

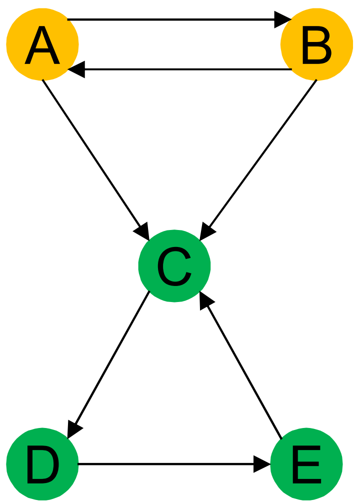
Google Formulation

- **PageRank issues**
  - (1) Some nodes are dead ends
    without out links
    - Random walker does not have a way to go
    - The importance "leaks out" from such nodes



Dead end

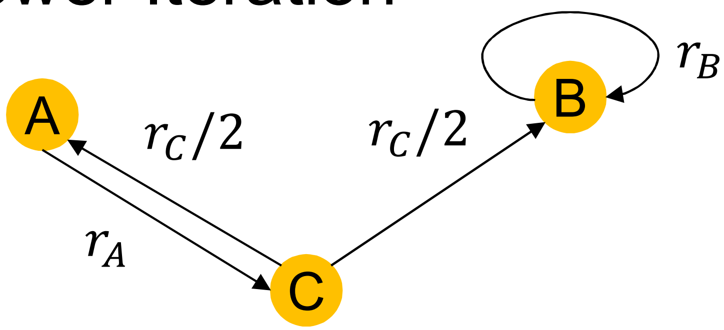
  - (2) There exist spider traps:
    (all links are within the group)
    - Random walker can not go out of the trap
    - At some point spider traps absorb
      all the importance



Spider trap

Google Formulation

- **Issue (2): Spider traps**
  - Power Iteration



$$
\begin{array}{c|ccc}
 & A & B & C \\
\hline
A & 0 & 0 & 1/2 \\
B & 0 & 1 & 1/2 \\
C & 1 & 0 & 0
\end{array}
$$

$$\underbrace{\qquad\qquad}_{M}$$

  - Results:

$$\begin{bmatrix} r_A \\ r_B \\ r_C \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \quad \begin{bmatrix} 1/6 \\ 3/6 \\ 2/6 \end{bmatrix} \quad \begin{bmatrix} 1/6 \\ 4/6 \\ 1/6 \end{bmatrix} \quad \cdots \quad \boxed{\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}}$$

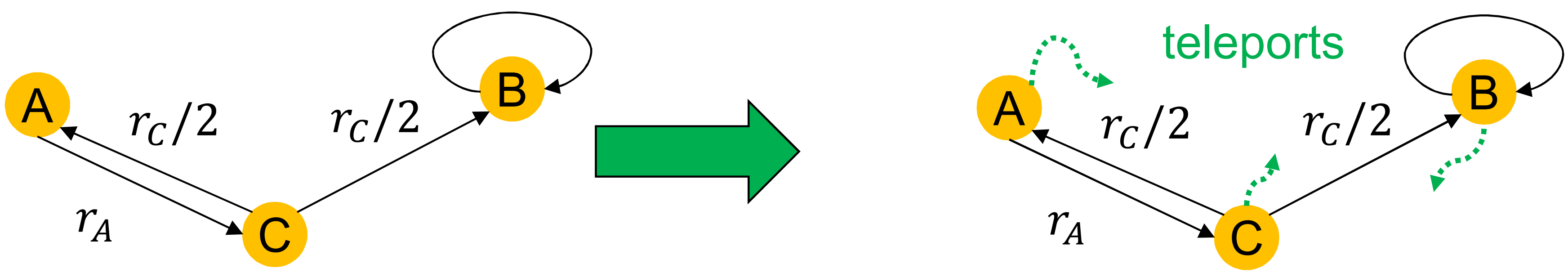
Iteration 0 1 2 …

B is a spider trap, all the importance gets trapped in node B

Copyright © 2016 CCL: Analysis of massive data sets

Google Formulation

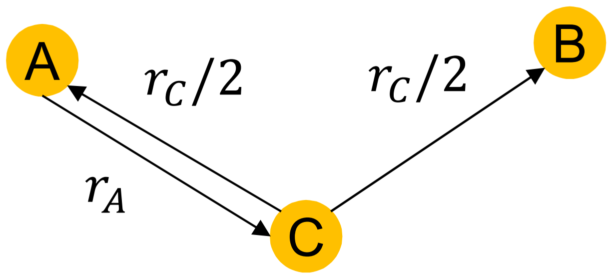
- **Issue (2): Spider traps**
- **Solution: Teleports!**
  - Google solution:
    At each time step, the random walker has two options:
    1) Follow the random link with some probability β
    2) Jump to some random node with probability $1 - \beta$
  - Common values for β
    - In the range 0.8 to 0.9
    - Random walker will teleport itself out of the spider trap



Copyright © 2016 CCL: Analysis of massive data sets

Google Formulation

- **Issue (1): Dead ends**
  - Power Iteration



$$M = \underbrace{\begin{array}{c|ccc} & A & B & C \\ \hline A & 0 & 0 & 1/2 \\ B & 0 & 0 & 1/2 \\ C & 1 & 0 & 0 \end{array}}_{M}$$

  - Results:

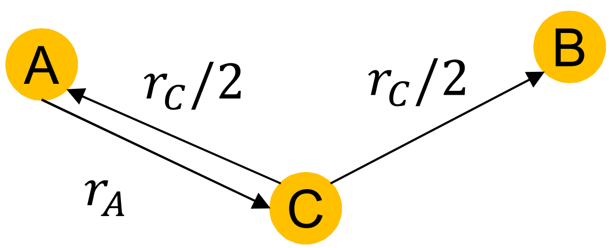
$$\begin{bmatrix} r_A \\ r_B \\ r_C \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \quad \begin{bmatrix} 1/6 \\ 1/6 \\ 2/6 \end{bmatrix} \quad \begin{bmatrix} 2/12 \\ 1/12 \\ 2/12 \end{bmatrix} \quad \cdots \quad \boxed{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}$$

Iteration 0 1 2 $\cdots$

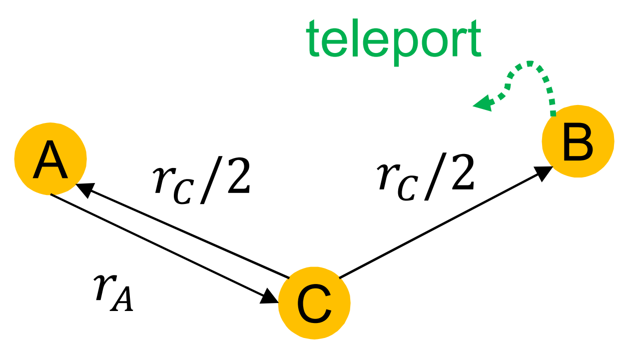
B is an dead end, all the importance "leaks out" since the matrix is not column-stochastic

Google Formulation

- **Issue (1): Dead ends**
- **Solution: Teleports!**
  - Google solution:
    Apply random teleports with the probability 1.0 from each dead end node



	A	B	C
A	0	0	1/2
B	0	0	1/2
C	1	0	0



	A	B	C
A	0	1/3	1/2
B	0	1/3	1/2
C	1	1/3	0

Google Formulation

- **Issues Specifics**
  - Spider traps
    - Spider traps are not the problem for the algorithm in a sense that it will still converge
    - The algorithm will work, but the rank scores are not what we expect
    - Solution: The walker always has the teleporting option, so she will never get stuck in a spider trap
  - Dead ends
    - Dead ends are a problem for the algorithm
    - The assumption was that the matrix is column-stochastic, so with dead ends the initial conditions are not met
    - Solution: For each dead end node the walker always follows teleports

Copyright © 2016 CCL: Analysis of massive data sets

Google Formulation

- **PageRank Equation [Brin-Page, ‘98]**

$$\mathrm{r_j} = \sum_{i \to j} \beta \frac{r_i}{d_i} + \frac{(1 - \beta)}{N}$$

  - With probability β, the walker follows link
  - With probability $1 - \beta$, the walker teleports to random page
  - This formula assumes that M has no dead ends

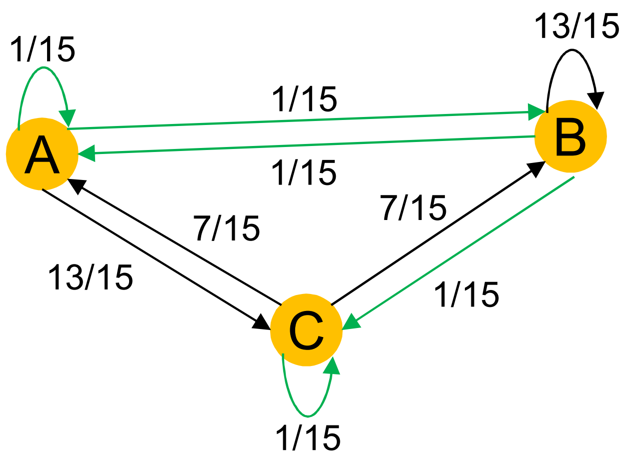
- **The Google Matrix A:**

$$A = \beta M + (1 - \beta)\left[\frac{1}{N}\right]_{N \times N}$$

- **Recursive problem: $\boldsymbol{r = A \cdot r}$**
  - The power iteration method works
    - $\beta = 0.8, 0.9$ (follow 5 steps and then jump)

Google Formulation

- **Back to our Example ($\beta = 0.8$)…**



$$0.8\underbrace{\begin{bmatrix} 0 & 0 & 1/2 \\ 0 & 1 & 1/2 \\ 1 & 0 & 0 \end{bmatrix}}_{M} + 0.2\underbrace{\begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}}_{\left[\frac{1}{N}\right]_{N\times N}}$$

$$\underbrace{\begin{array}{c} A \\ B \\ C \end{array}\begin{bmatrix} 1/15 & 1/15 & 7/15 \\ 1/15 & 13/15 & 7/15 \\ 13/15 & 1/15 & 1/15 \end{bmatrix}}_{A}$$

- **Results:**

$$\begin{bmatrix} r_A \\ r_B \\ r_C \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \quad \begin{bmatrix} 0.20 \\ 0.46 \\ 0.33 \end{bmatrix} \quad \begin{bmatrix} 0.20 \\ 0.58 \\ 0.23 \end{bmatrix} \quad \dots \quad \begin{bmatrix} 7/51 \\ 35/51 \\ 9/51 \end{bmatrix}$$

Iteration 0 1 2 …

Implementation Details

- **Computing PageRank**
  - Iterative algorithm
    - Uses matrix-vector multiplication
    - $r^{(t+1)} = A \cdot r^{(t)}$
  - It is straightforward if we have enough memory to keep A, $r^{(t)}$ and $r^{(t+1)}$
  - For example, $N = 10^9$ nodes in the graph
    - For each entry we need 4 bytes
    - For 2 vectors, we need 2 billion entries ~ 8GB, it might fit in RAM
    - Matrix A has N^2 entries – 10^{18}

Implementation Details

- **Matrix Reformulation**
  - Matrix M content
    - Consider node i, with d_i outgoing links
      - $M_{ji} = 1/|d_i|$, if $i \to j$
      - $M_{ji} = 0$ otherwise
    - Suppose we have 10 links per node, M is very sparse
  - Random teleports make the matrix A dense, but…
    - Adding a teleport link from node i to every node with the probability $(1-\beta)/N$
    - Reducing the probability of following a link from $1/|d_i|$ to $\beta/|d_i|$
    - Equivalent: Tax the importance of each node by $(1-\beta)$ and redistribute it evenly!

Copyright © 2016 CCL: Analysis of massive data sets

Implementation Details

- **Matrix Reformulation**
  - $r = A \cdot r,\ A_{ji} = \beta M_{ji} + \frac{1-\beta}{N}$
  - $r_j = \sum_{i=1}^{N} A_{ji} \cdot r_i$
  - $r_j = \sum_{i=1}^{N} [\beta M_{ji} + \frac{1-\beta}{N}] \cdot r_i$
  - $r_j = \sum_{i=1}^{N} \beta M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{i=1}^{N} r_i$, note that: $\sum_{i=1}^{N} r_i = 1$
  - $r_j = \sum_{i=1}^{N} \beta M_{ji} \cdot r_i + \frac{1-\beta}{N}$
  - Hence, we get

$$r = \beta \cdot M + \left[\frac{1-\beta}{N}\right]_N$$

We assume that **M** has no dead ends

Implementation Details

- **New PageRank equation:**

$$r = \beta \cdot M + \left[\frac{1-\beta}{N}\right]_N$$

- **M is a sparse matrix!**
  - 10 links per node, ~10*N entries
  - In each iteration:
    - Compute $r^{(t+1)} = \beta \mathrm{M} \cdot \mathrm{r}^{(t)}$
    - Add a constant value $(1-\beta)/N$ to each component in $r^{(t+1)}$
    - If M has dead ends
      - $\sum_{i=1}^{N} r_i < 1$, the importance leaks out
      - We need to compute the importance that has leaked out and redistribute it evenly so r still sums to 1, $\sum_{i=1}^{N} r_i = 1$

Implementation Details

- **PageRank Algorithm**

```
Input:  Graph G (with spider traps and dead ends), β
Output: PageRank vector r
```

$r_j^{(t)} := 1/N$

do

$S := 0$ `//accumulator for non-leaked importance`

for $j := 1$ **to** N

$r'^{(t+1)}_j := 0$

$S \mathrel{+}= \sum_{i \to j} \beta \left(\frac{r_i^{(t)}}{d_i}\right)$

$r'^{(t+1)}_j \mathrel{+}= S$

for $j := 1$ **to** N `// re-insert the leaked importance`

$r'^{(t+1)}_j \mathrel{+}= (1 - S)/N$

while $\left(\sum_j \left| r_j^{(t+1)} - r_j^{(t)} \right| > \epsilon\right)$

Implementation Details

- **Data Structures**
  - Use adjacency list for graph/matrix representation
    - Encode it using nonzero entries
    - It requires space proportional to the number of links
    - Lets assume web graph
      - $N = 10^9$, 1 billion nodes
      - Space ~ $10 \cdot N = 4 \cdot 10 \cdot 10^9 =$ 40 GB
      - It will not fit in main memory, but it can fit on disk

Source node	Node Degree	Destination nodes
0	2	1, 4
1	6	0, 3, 5, 12, 98, 314
…	…	…
N	5	18, 2, 3, 8, 10

Implementation Details

- **PageRank Algorithm**
- **Hard Cases**

  - Case (1):
    - Assume M cannot fit in memory
    - Assume $r^{(t+1)}$ can fit in RAM

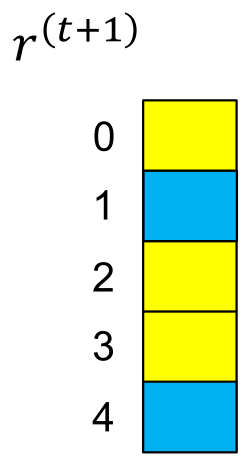
  - Case (2):
    - Assume M cannot fit in memory
    - Assume even $r^{(t+1)}$ cannot fit in RAM

Copyright © 2016 CCL: Analysis of massive data sets

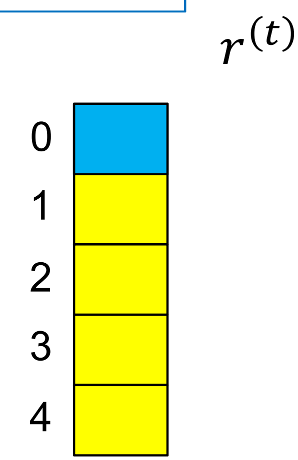
Implementation Details

- **Case (1)**
  - We assume M cannot fit in memory and $r^{(t+1)}$ can fit in RAM
  - 1 step of iteration *M and $r^{(t+1)}$ stored on disk*:

$r^{(t+1)} := (1-\beta)/N$
for each node i **with out degree** d_i
  read in memory: $i, d_i, dest_1, dest_2, \ldots, dest_{d_i}, r^{(t)}[i]$
  for $j := 1$ **to** d_i
    $r^{(t+1)}[dest_j] \mathrel{+}= \beta\, r^{(t)}[i]/d_i$



Source	Degree	Destination
0	2	1, 4
1	3	12, 98, 314
2	5	18, 2, 3, 8, 10



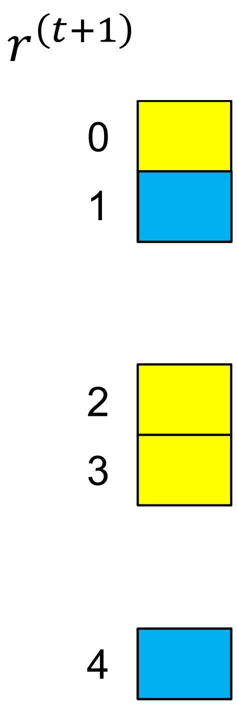
Implementation Details

- **Case (1)**
  - Computational Analysis
    - Store $r^{(t)}$ and matrix M on disk
    - In each iteration:
      - Read $r^{(t)}$
      - Write $r^{(t+1)}$
    - Hence, cost per iteration
      - $\mathbf{2|r| + |M|}$
  - What should we do in Case (2)?
    - $r^{(t+1)}$ cannot fit in RAM

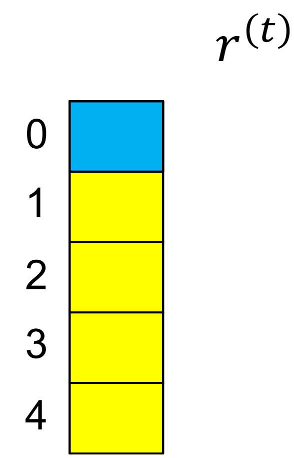
Implementation Details

Case (2)

- We assume neither M nor $r^{(t+1)}$ can fit in RAM
- Solution: we split vector $r^{(t+1)}$ into k blocks that do fit in RAM



Source	Degree	Destination
0	2	1, 4
1	3	12, 98, 314
2	5	18, 2, 3, 8, 10



Implementation Details

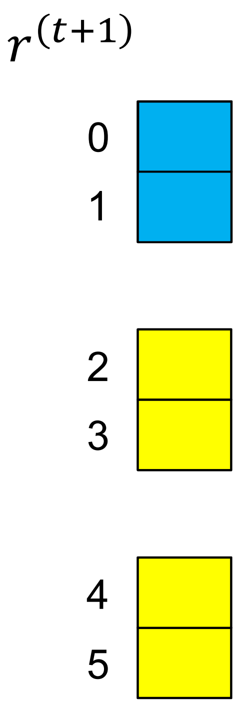
- **Case (2)**
  - Computational Analysis
    - We assume neither M nor $r^{(t+1)}$ can fit in RAM
    - Solution: we split vector $r^{(t+1)}$ into k blocks that do fit in RAM
    - Scan M and $r^{(t)}$ once for each block k_i
  - Cost per iteration
    - $\boldsymbol{k}(|\boldsymbol{M}| + |\boldsymbol{r}|)$
    - M is larger than r, $\sim(10-20)\times$ larger
    - Can we do better?

Copyright © 2016 CCL: Analysis of massive data sets

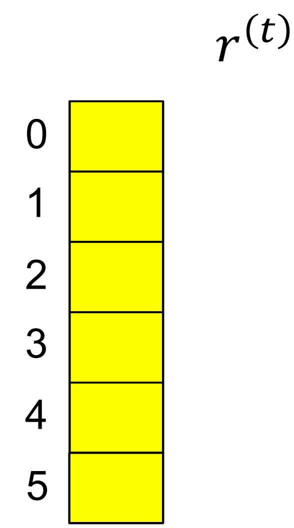
Implementation Details

Case (2) – Can we do better?

- We assume neither M nor $r^{(t+1)}$ can fit in RAM
    - (A) Split vector $r^{(t+1)}$ into k blocks that do fit in RAM
    - (B) Break M into k stripes so each stripe contains only destinations that are in the corresponding block of $r^{(t+1)}$



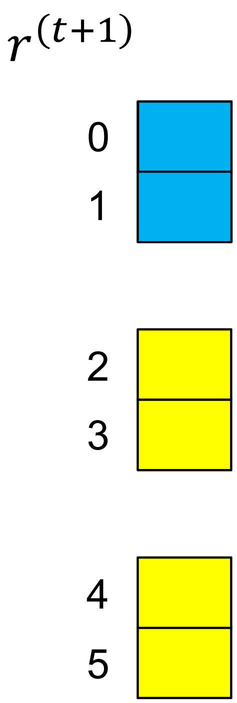
Source	Degree	Destination
0	4	0, 1, 3, 4
1	2	0, 4
2	3	1, 2, 5



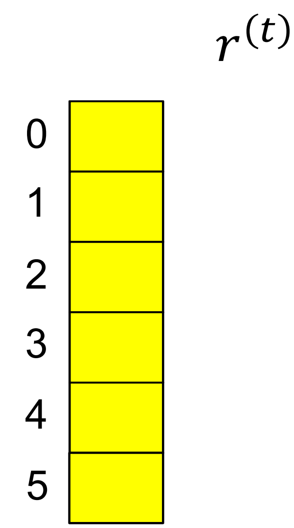
Copyright © 2016 CCL: Analysis of massive data sets

□ Case (2) – Can we do better?

- We assume neither M nor $r^{(t+1)}$ can fit in RAM
  - (A) Split vector $r^{(t+1)}$ into k blocks that do fit in RAM
  - (B) Break M into k stripes so each stripe contains only destinations that are in the corresponding block of $r^{(t+1)}$



Source	Degree	Destination
0	4	0, 1
1	2	0
2	3	1

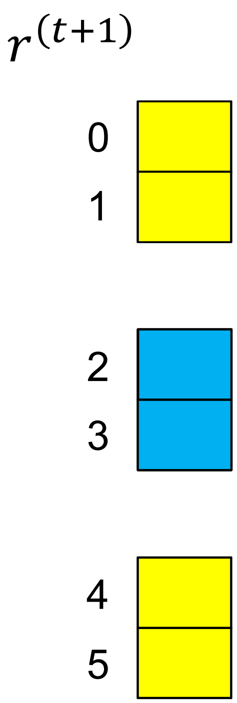


Copyright © 2016 CCL: Analysis of massive data sets

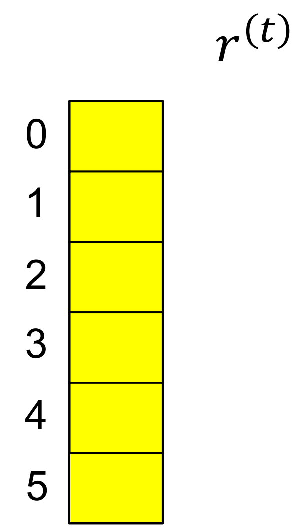
Implementation Details

□ Case (2) – Can we do better?

- We assume neither M nor $r^{(t+1)}$ can fit in RAM
  - (A) Split vector $r^{(t+1)}$ into k blocks that do fit in RAM
  - (B) Break M into k stripes so each stripe contains only destinations that are in the corresponding block of $r^{(t+1)}$

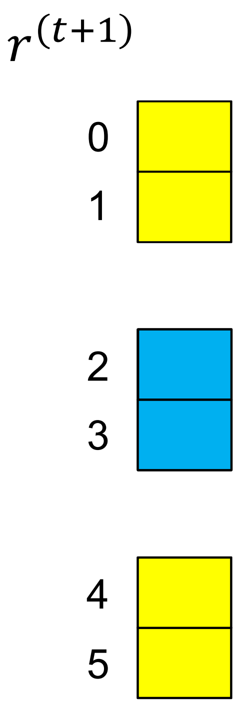


Source	Degree	Destination
0	4	0, 1, 3, 4
1	2	0, 4
2	3	1, 2, 5



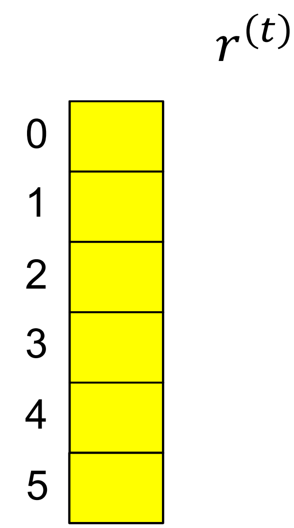
Implementation Details

- **Case (2) – Can we do better?**
  - We assume neither M nor $r^{(t+1)}$ can fit in RAM
    - (A) Split vector $r^{(t+1)}$ into k blocks that do fit in RAM
    - (B) Break M into k stripes so each stripe contains only destinations that are in the corresponding block of $r^{(t+1)}$



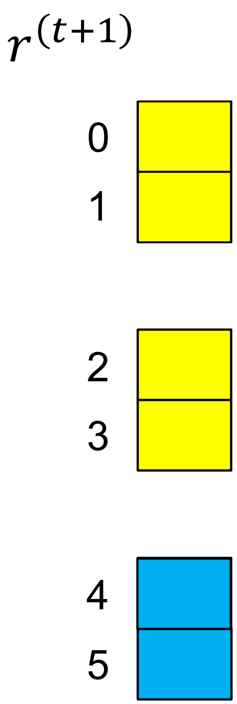
Source	Degree	Destination
0	4	0, 1
1	2	0
2	3	1

0	4	3
2	3	2

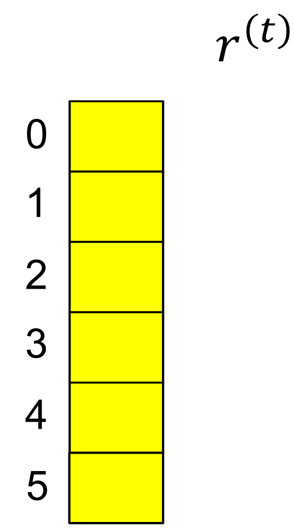


Implementation Details

- **Case (2) – Can we do better?**
  - We assume neither M nor $r^{(t+1)}$ can fit in RAM
    - (A) Split vector $r^{(t+1)}$ into k blocks that do fit in RAM
    - (B) Break M into k stripes so each stripe contains only destinations that are in the corresponding block of $r^{(t+1)}$



Source	Degree	Destination
0	4	0, 1, 3, 4
1	2	0, 4
2	3	1, 2, 5

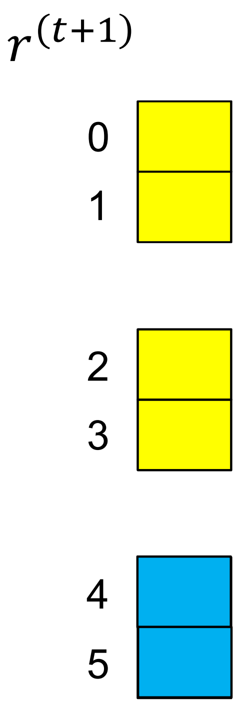


Copyright © 2016 CCL: Analysis of massive data sets

Implementation Details

Case (2) – Can we do better?

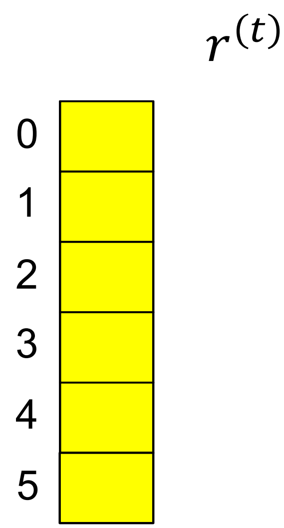
- We assume neither M nor $r^{(t+1)}$ can fit in RAM
  - (A) Split vector $r^{(t+1)}$ into k blocks that do fit in RAM
  - (B) Break M into k stripes so each stripe contains only destinations that are in the corresponding block of $r^{(t+1)}$



Source	Degree	Destination
0	4	0, 1
1	2	0
2	3	1

0	4	3
2	3	2

0	4	4
1	2	4
2	3	5



Implementation Details

- **Case (2) – Better Approach**
  - Computational Analysis
    - We assume neither M nor $r^{(t+1)}$ can fit in RAM
      - (A) Split vector $r^{(t+1)}$ into k blocks that do fit in RAM
      - (B) Break M into k stripes so each stripe contains only destinations that are in the corresponding block of $r^{(t+1)}$
  - Cost per iteration
    - Some additional overhead per stripe
    - $\boldsymbol{|M|(1+\epsilon) + k|r|}$

PageRank Issues

- **PageRank Issues**
  - It measures general popularity of the node
    - Some topics may not be so popular
    - Solution: topic-specific PageRank algorithm
  - It uses a single measure of importance
    - There are other models of importance
    - Solution: Hubs and Authorities
  - Vulnerable to Link spam
    - One could create artificial structure of nodes in order to boost the importance of particular node
    - Solution: TrustRank