PageRank and Adsorption

MapReduce Framework

Plan for today

- PageRank
  - Different formulations: Iterative, Matrix
  - Complications: Sinks and hogs
  - Implementation in MapReduce
- Adsorption
  - Label propagation
  - Implementation in MapReduce

NEXT

Cork Complex Systems Lab

Background

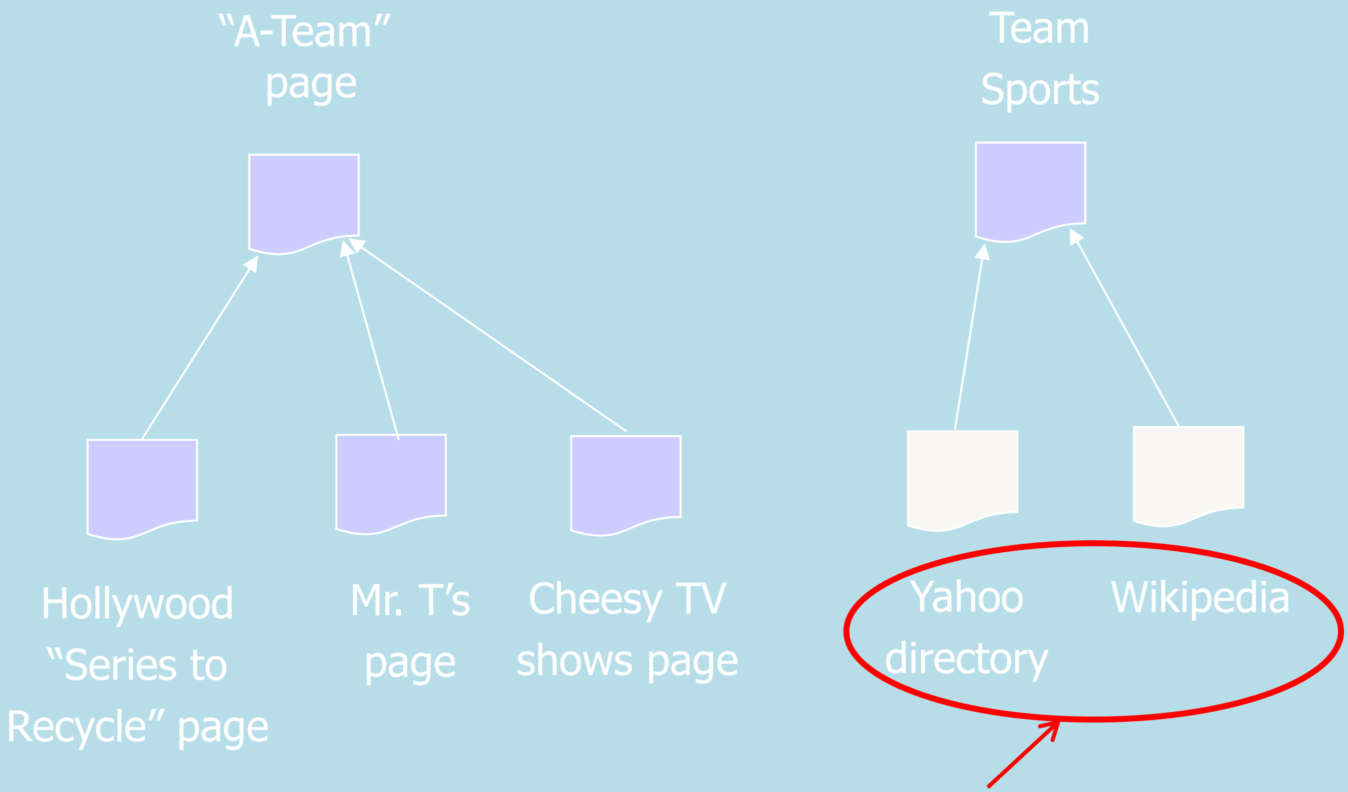
- History:
  - Proposed by Sergey Brin and Lawrence Page (Google's Bosses) in 1998 at Stanford.
  - Algorithm of the first generation of Google Search Engine.
  - "The Anatomy of a Large-Scale Hypertextual Web Search Engine".

- Target:
  - Measure the importance of Web page based on the link structure alone.
  - Assign each node a numerical score between 0 and 1: PageRank.
  - Rank Web pages based on PageRank values.

Why link analysis?

- Suppose a search engine processes a query for "team sports"
  - Problem: Millions of pages contain these words!
  - Which ones should we return first?
- Idea: Hyperlinks encode a considerable amount of human judgment
  - What does it mean when a web page links another page?
  - Intra-domain links: Often created primarily for navigation
  - Inter-domain links: Confer some measure of authority
- So, can we simply boost the rank of pages with lots of inbound links?

Problem: Popularity $\neq$ relevance!



Shouldn't links from Yahoo and Wikipedia "count more"?

Other applications

- This question occurs in several other areas:
  - How do we measure the "impact" of a researcher? (#papers? #citations?)
  - Who are the most "influential" individuals in a social network?
    (#friends?)
  - Which programmers are writing the "best" code? (#uses?)
  - ...

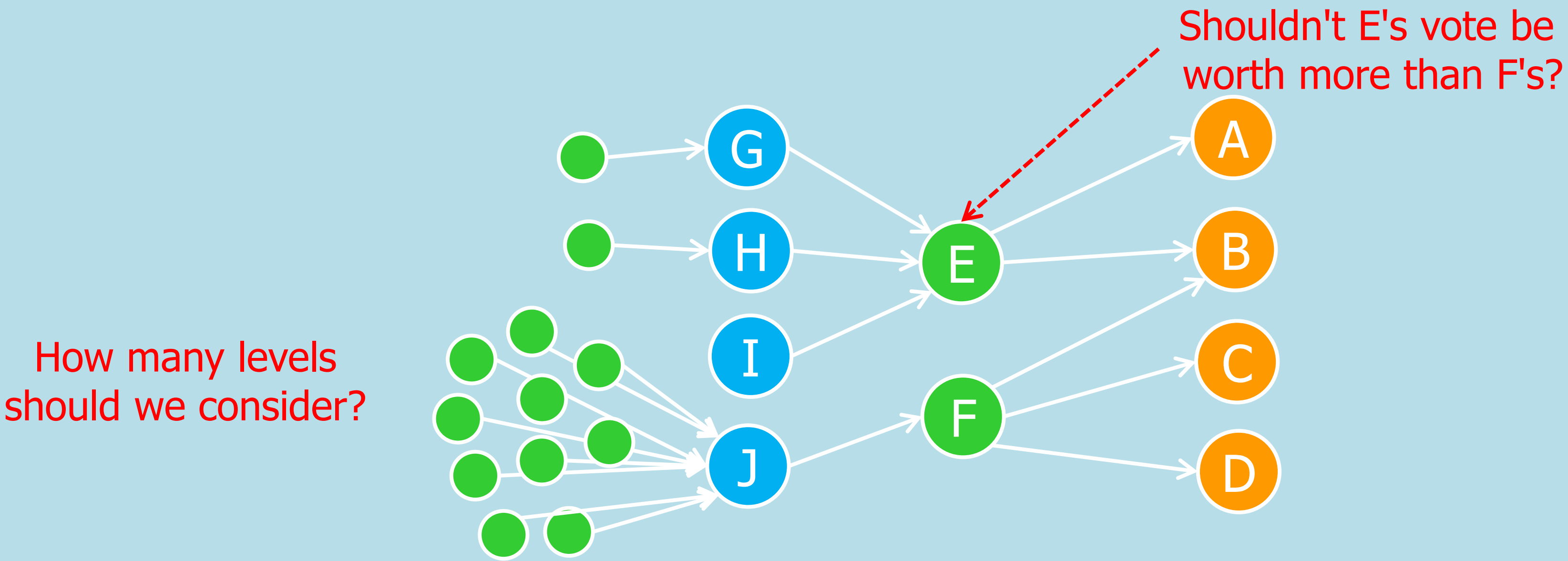
Largest Matrix Computation in the World

- Computing PageRank done via matrix multiplication
  - matrix has 3 billion rows and columns.
- The matrix is sparse
  - average number of outlinks is between 7 and 8.
- Researchers still trying to speed-up the computation
- PageRank convergence
  - Setting d = 0.15 (teleportation probability or decay factor for loops) or above requires at most 100 iterations to convergence.

Link Spamming to Improve PageRank

- Spam is the act of trying unfairly to gain a high ranking on a search engine for a web page without improving the user experience.
- *Link farms* - join the farm by copying a hub page which links to all members.
- *Selling links* from sites with high PageRank.

PageRank: Intuition



- Imagine a contest for The Web's Best Page
  - Initially, each page has one vote
  - Each page votes for all the pages it has a link to
  - To ensure fairness, pages voting for more than one page must split their vote equally between them
  - Voting proceeds in rounds; in each round, each page has the number of votes it received in the previous round
  - In practice, it's a little more complicated - but not much!

PageRank

- Each page i is given a rank x_i
- Goal: Assign the x_i such that the rank of each page is governed by the ranks of the pages linking to it:

$$x_i = \sum_{j \in B_i} \frac{1}{N_j} x_j$$

Rank of page i

Every page j that links to i

Number of links out from page j

Rank of page j

How do we compute the rank values?

Random Surfer Model

- PageRank has an intuitive basis in random walks on graphs
- Imagine a random surfer, who starts on a random page and, in each step,
  - with probability d, clicks on a random link on the page
  - with probability 1-d, jumps to a random page (bored?)

Reason explained later

- The PageRank of a page can be interpreted as the fraction of steps the surfer spends on the corresponding page
  - Transition matrix can be interpreted as a Markov Chain

Iterative PageRank (simplified)

Initialize all ranks to be equal, e.g.:

$$x_i^{(0)} = \frac{1}{n}$$

Iterate until convergence

$$x_i^{(k+1)} = \sum_{j \in B_i} \frac{1}{N_j} x_j^{(k)}$$

No need to decide how many levels to consider!

Example: Step 0

Initialize all ranks
to be equal

$$x_i^{(0)} = \frac{1}{n}$$

Example: Step 1

Propagate weights across out-edges

$$x_i^{(k+1)} = \sum_{j \in B_i} \frac{1}{N_j} x_j^{(k)}$$

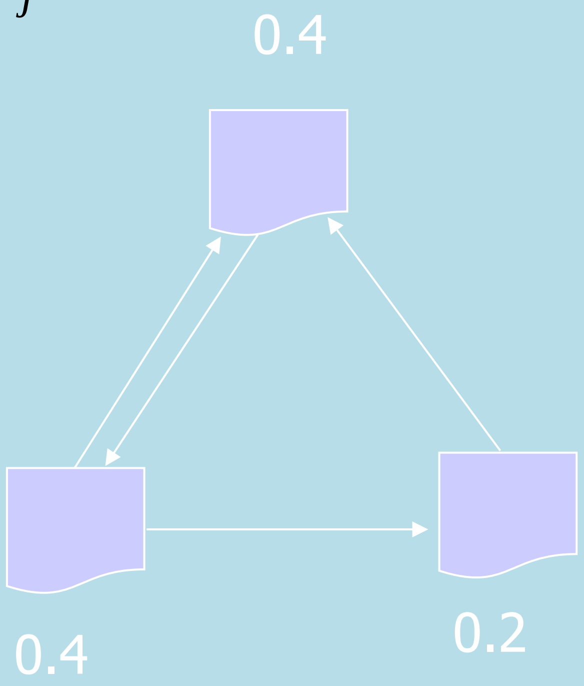
Example: Step 2

Compute weights based on in-edges

$$x_i^{(1)} = \sum_{j \in B_i} \frac{1}{N_j} x_j^{(0)}$$

Example: Convergence

$$x_i^{(k+1)} = \sum_{j \in B_i} \frac{1}{N_j} x_j^{(k)}$$



Naïve PageRank Algorithm Restated

- Let
  - N(p) = number outgoing links from page p
  - B(p) = number of back-links to page p

$$PageRank(p) = \sum_{b \in B(p)} \frac{1}{N(b)} PageRank(b)$$

  - Each page b distributes its importance to all of the pages it points to (so we scale by 1/N(b))
  - Page p's importance is increased by the importance of its back set

In Linear Algebra formulation

- Create an m x m matrix M to capture links:
  - $M(i, j) = 1 / n_j$ if page i is pointed to by page j and page j has n_j outgoing links
    $= 0$ otherwise
  - Initialize all PageRanks to 1, multiply by M repeatedly until all values converge:

$$\begin{bmatrix} PageRank(p_1') \\ PageRank(p_2') \\ \dots \\ PageRank(p_m') \end{bmatrix} = M \begin{bmatrix} PageRank(p_1) \\ PageRank(p_2) \\ \dots \\ PageRank(p_m) \end{bmatrix}$$

  - Computes principal eigenvector via power iteration

Cork Complex Systems Lab

A brief example

$$\begin{vmatrix} g' \\ y' \\ a' \end{vmatrix} = \begin{vmatrix} 0 & 0.5 & 0.5 \\ 0 & 0 & 0.5 \\ 1 & 0.5 & 0 \end{vmatrix} * \begin{vmatrix} g \\ y \\ a \end{vmatrix}$$

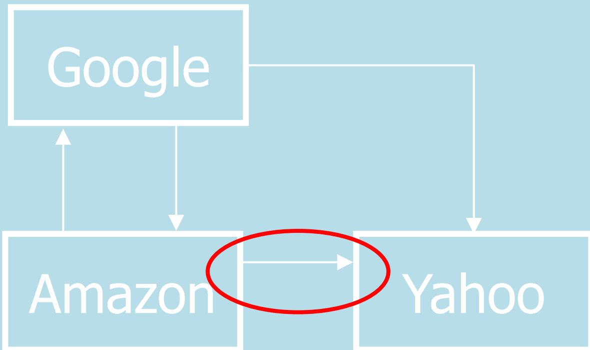
Running for multiple iterations:

$$\begin{vmatrix} g \\ y \\ a \end{vmatrix} = \begin{vmatrix} 1 \\ 1 \\ 1 \end{vmatrix}, \begin{vmatrix} 1 \\ 0.5 \\ 1.5 \end{vmatrix}, \begin{vmatrix} 1 \\ 0.75 \\ 1.25 \end{vmatrix}, \ldots \begin{vmatrix} 1 \\ 0.67 \\ 1.33 \end{vmatrix}$$

Total rank sums to number of pages

Oops #1

– PageRank sinks



$$\begin{vmatrix} g' \\ y' \\ a' \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0.5 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0 \end{vmatrix} * \begin{vmatrix} g \\ y \\ a \end{vmatrix}$$

'dead end' - PageRank is lost after each round

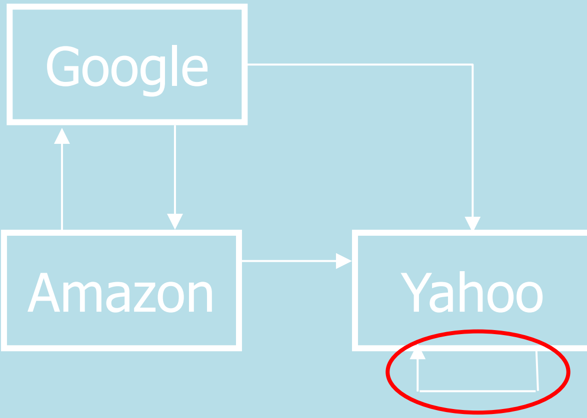
Running for multiple iterations:

$$\begin{vmatrix} g \\ y \\ a \end{vmatrix} = \begin{vmatrix} 1 \\ 1 \\ 1 \end{vmatrix}, \begin{vmatrix} 0.5 \\ 1 \\ 0.5 \end{vmatrix}, \begin{vmatrix} 0.25 \\ 0.5 \\ 0.25 \end{vmatrix}, \ldots, \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix}$$

Cork Complex Systems Lab

Oops #2

– PageRank hogs



PageRank cannot flow out and accumulates

$$\begin{vmatrix} g' \\ y' \\ a' \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0 & 0 \end{vmatrix} * \begin{vmatrix} g \\ y \\ a \end{vmatrix}$$

Running for multiple iterations:

$$\begin{vmatrix} g \\ y \\ a \end{vmatrix} = \begin{vmatrix} 1 \\ 1 \\ 1 \end{vmatrix}, \begin{vmatrix} 0.5 \\ 2 \\ 0.5 \end{vmatrix}, \begin{vmatrix} 0.25 \\ 2.5 \\ 0.25 \end{vmatrix}, \ldots, \begin{vmatrix} 0 \\ 3 \\ 0 \end{vmatrix}$$

Stopping the Hog

$$\begin{vmatrix} g' \\ y' \\ a' \end{vmatrix} = 0.85 \begin{vmatrix} 0 & 0 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0 & 0 \end{vmatrix} * \begin{vmatrix} g \\ y \\ a \end{vmatrix} + \begin{vmatrix} 0.15 \\ 0.15 \\ 0.15 \end{vmatrix}$$

Running for multiple iterations:

$$\begin{vmatrix} g \\ y \\ a \end{vmatrix} = \begin{vmatrix} 0.57 \\ 1.85 \\ 0.57 \end{vmatrix}, \begin{vmatrix} 0.39 \\ 2.21 \\ 0.39 \end{vmatrix}, \begin{vmatrix} 0.32 \\ 2.36 \\ 0.32 \end{vmatrix}, \dots, \begin{vmatrix} 0.26 \\ 2.48 \\ 0.26 \end{vmatrix}$$

… though does this seem right?

Improved PageRank

- Remove out-degree 0 nodes (or consider them to refer back to referrer)
- Add decay factor d to deal with sinks

$$PageRank(p) = (1-d) + d\sum_{b \in B_p} \frac{1}{N(b)} PageRank(b)$$

- Typical value: d=0.85
- Intuition in the idea of the “random surfer”:
  - Surfer occasionally stops following link sequence and jumps to new random page, with probability 1 - *d*

Markov Chains

- Markov Chain:
  - A Markov chain is a discrete-time stochastic process consisting of N states, each Web page corresponds to a state.
  - A Markov chain is characterized by an N*N transition probability matrix P.
- Transition Probability Matrix:
  - Each entry is in the interval [0,1].
  - $\forall i, j, P_{ij} \in [0,1]$ P_{ij} is the probability that the state at the next time-step is j, conditioned on the current state being i.
  - Each entry P_{ij} is known as a transition probabilit and depends only on the current state i. Markov property.

$$\forall i, \sum_{j=1}^{N} Pij = 1$$

Cork Complex Systems Lab

Markov Chains

- Transition Probability Matrix:
  - A matrix with non-negative entries that satisfies $\forall i, \sum_{j=1}^{N} Pij = 1$
  - is known as a stochastic matrix.
  - Has a principal left eigenvector corresponding to its largest eigenvalue, which is 1.
- Derive the Transition Probability Matrix P:
  - Build the adjacency matrix A of the web graph.
    - There is a hyperlink from page i to page j, Aij = 1, otherwise Aij =0.
  - Divide each 1 in A by the number of 1s in its row.
  - Multiply the resulting matrix by 1- α.
  - Add α/N to every entry of the resulting matrix, to obtain P.

Markov Chains

- Ergodic Markov Chain :
  - Conditions:
    - Irreducibility
      - A sequence of transitions of nonzero probability from any state to any state.
    - Aperiodicity
      - States are not partitioned into sets such that all state transitions occur cyclically from one set to another.
  - Property:
    - There is a unique steady-state probability vector π that is the principal left eigenvector of P.
    - $\eta(i,t)$ is the number of visits to state i in t steps.
    - $\pi(i)>0$ is the steady-state probability for state i.

$$\lim_{t->\infty} \frac{\eta(i,t)}{t} = \pi(i)$$

PageRank Computation

- Target
  - Solve the steady-state probability vector π, which is the PageRank of the corresponding Web page.
  - $\pi P = \lambda \pi$, λ is 1 for stochastic matrix.
- Method
  - Power iteration.
  - Given an initial probability distribution vector x0
  - $x_0 * P = x_1$, $x_1 * P = x_2$ … Until the probability distribution converges. (Variation in the computed values are below some predetermined threshold.)

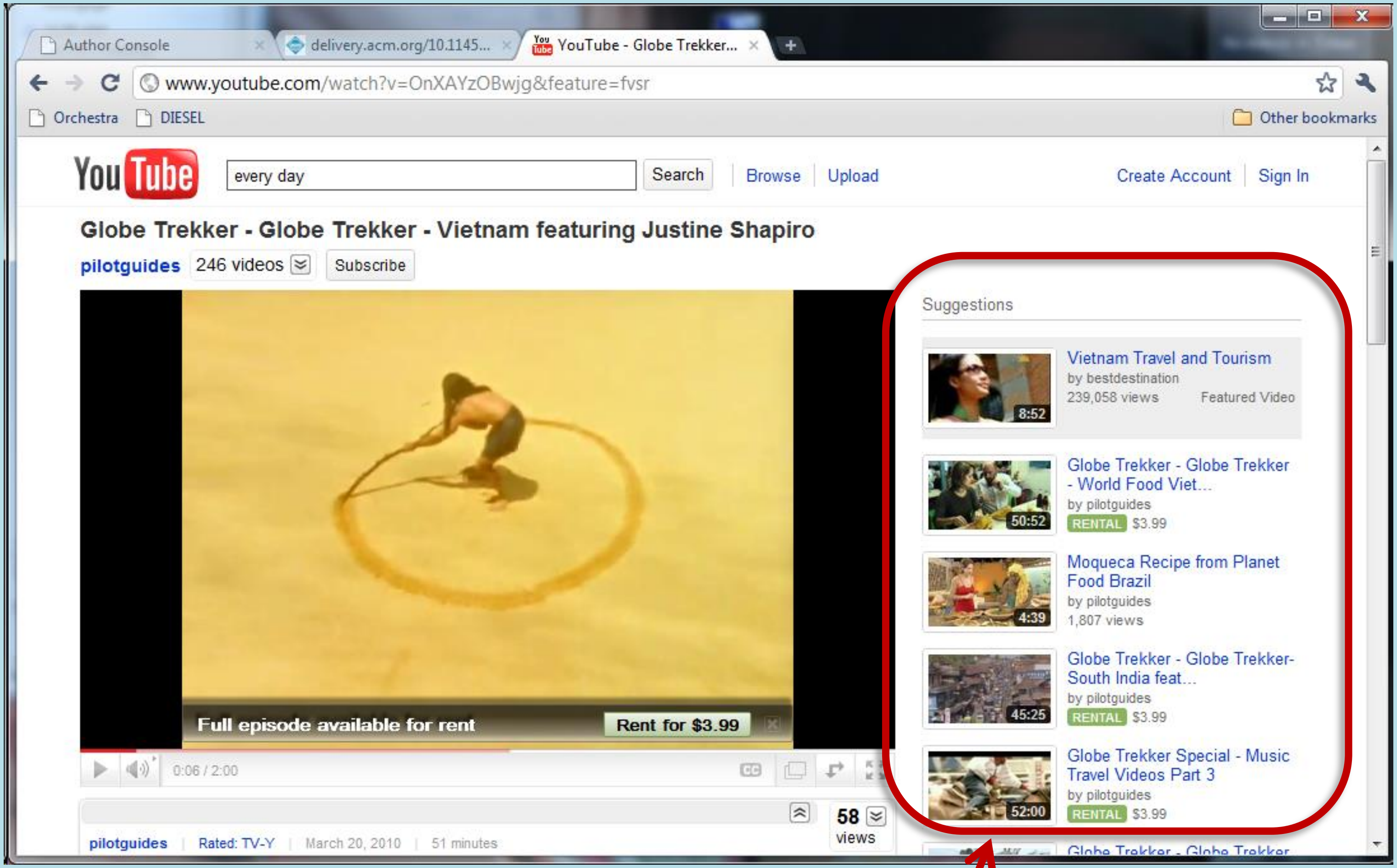
PageRank on MapReduce

- Inputs
  - Of the form: page → (currentWeightOfPage, {adjacency list})
- Map
  - Page p "propagates" $1/N_p$ of its d * weight(p) to the destinations of its out-edges (think like a vertex!)
  - Output adjacency list
- Reduce
  - Page p sums the incoming weights and adds (1-d), to get its weight'(p)
- Iterate until convergence
  - Common practice: run some fixed number of times, e.g., 25x
  - Alternatively: Test after each iteration with a second MapReduce job, to determine the maximum change between old and new weights

Plan for today

- PageRank
  - Different formulations: Iterative, Matrix
  - Complications: Sinks and hogs
  - Implementation in MapReduce
- Adsorption NEXT
  - Label propagation
  - Implementation in MapReduce

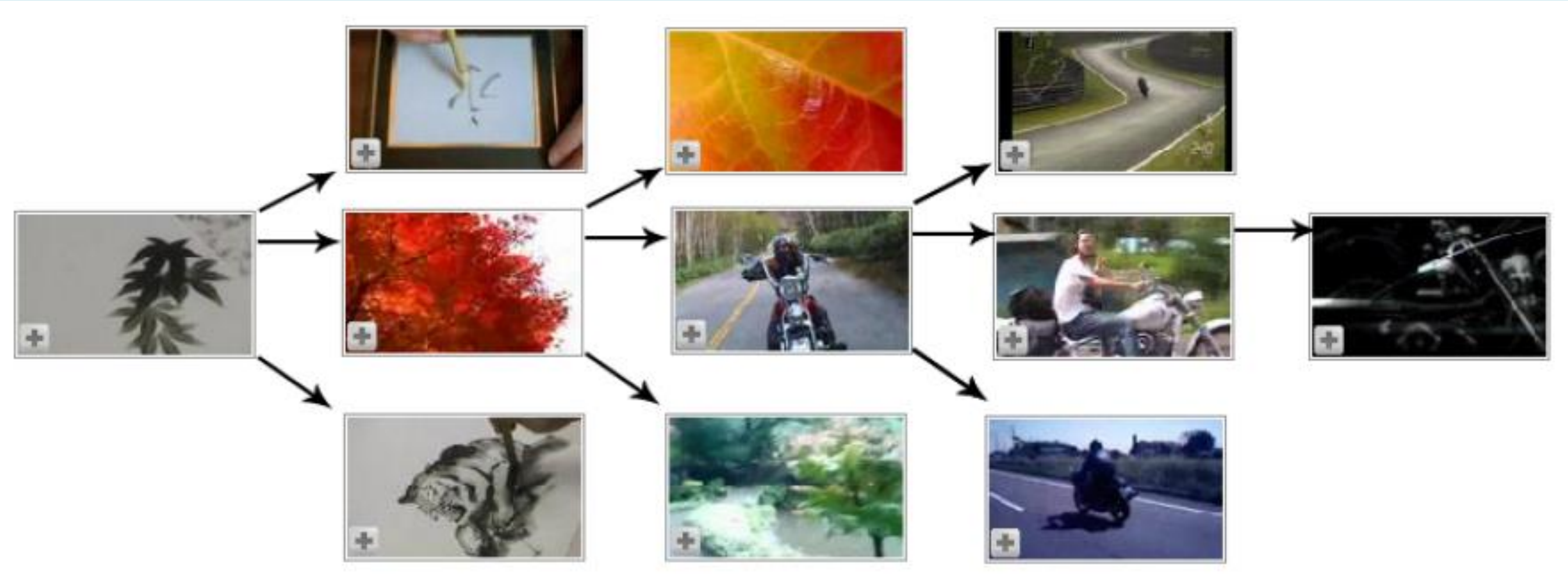
YouTube Suggestions



What can we leverage to make such recommendations?

Cork Complex Systems Lab

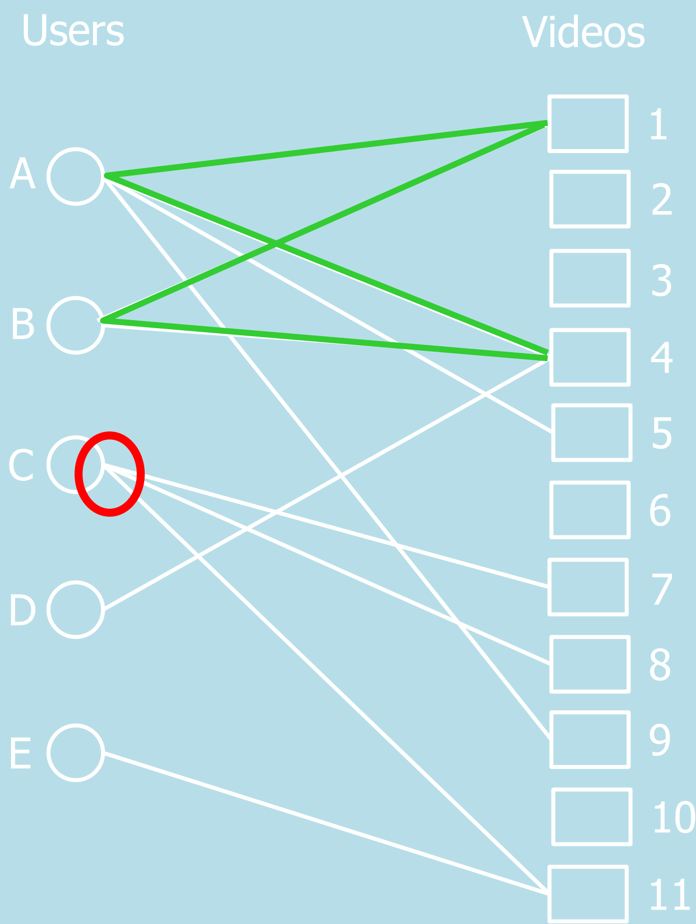
Co-views: Video-video



- Idea #1: Keep track of which videos are frequently watched together
  - If many users watched both video A and video B, then A should be recommended to users who have viewed B, and vice versa
  - If there are many such videos, how can we rank them?

Cork Complex Systems Lab

Co-Views: User-video

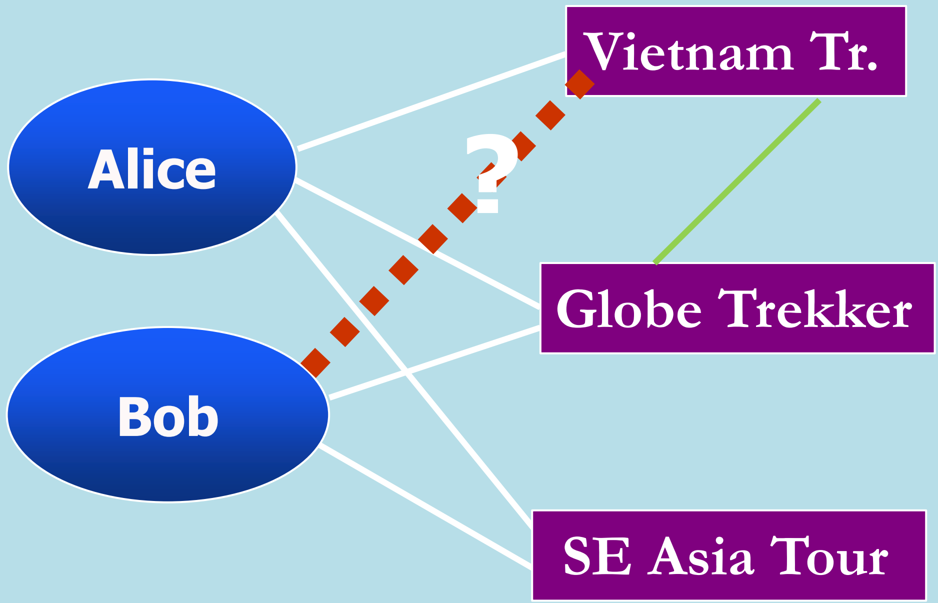


- Idea #2: Leverage similarities between users
  - If Alice and Bob have both watched videos A, B, and C, and Alice has additionally watched video D, then perhaps D will interest Bob too?
- How can we see that in the graph?
  - Short path between two videos
  - Several paths between 2 videos
  - Paths that avoid high-degree nodes in the graph (why?)

More sophisticated link analysis

- PageRank computes a stationary distribution for the random walk: the probability is independent of where you start
  - One authority score for every page
- But here we want to know how likely we are to end up at video *j* given that we started from user *i*
  - e.g., what are the odds that user i will like video j?
  - this is a probability conditioned on where you start

Video-video and user-video combined



- Our task:
  - Take the video-video co-views and the user-video co-views (potentially annotated with weights)
  - Assign to each video a score for each user, indicating the likelihood the user will want to view the video

Adsorption: Label propagation

- Adsorption: Adhesion of atoms, ions, etc. to a surface
  - The adsorption algorithm attempts to "adhere" labels and weights to various nodes, establishing a connection
- There are three equivalent formulations of the method:
  - A random walk algorithm that looks much like PageRank
  - An averaging algorithm that is easily MapReduced
    - This is the one we'll focus on
  - A linear systems formulation

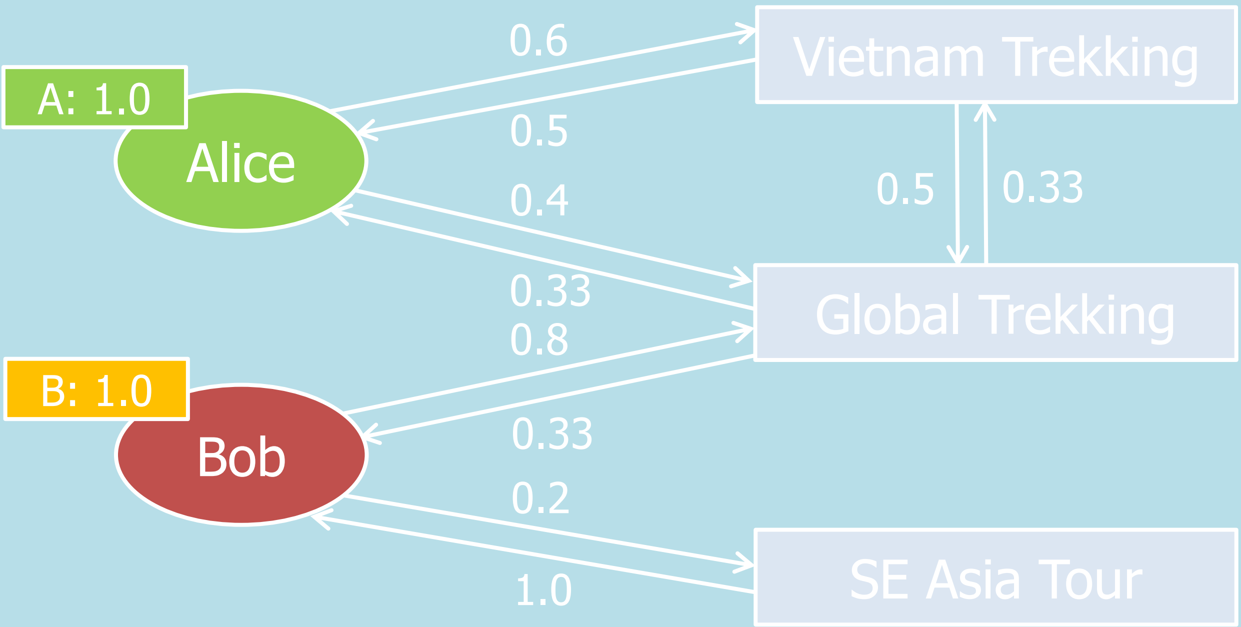
Adsorption as an iterative average

- Easily MapReducible

- Pre-processing step:
  - Create a series of labels L, one for each user or entity
  - Take the set of vertices V
  - For each label l in L:
    - Designate an “origin” node v_l (typically given node label l)
    - Annotate it with the label l and weight 1.0
  - Much like what we do in PageRank to start

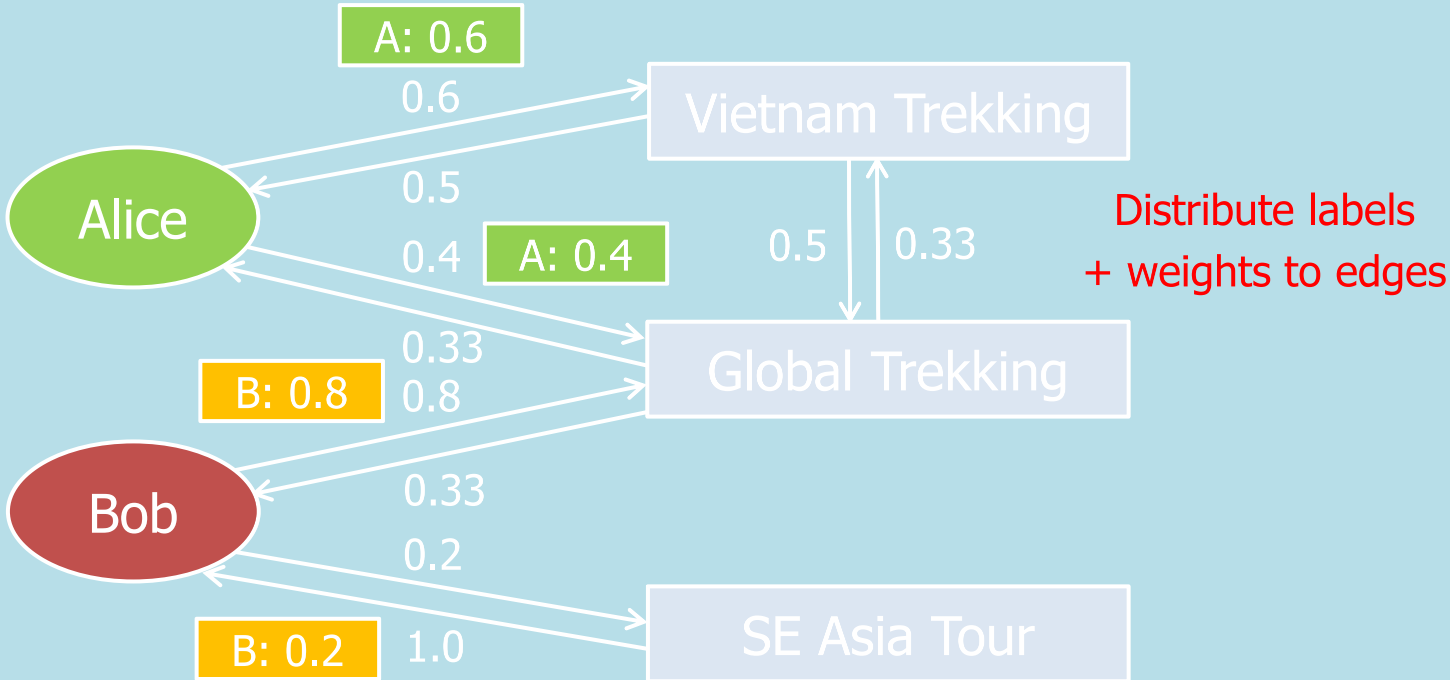
Cork Complex Systems Lab

Adsorption: Propagating labels



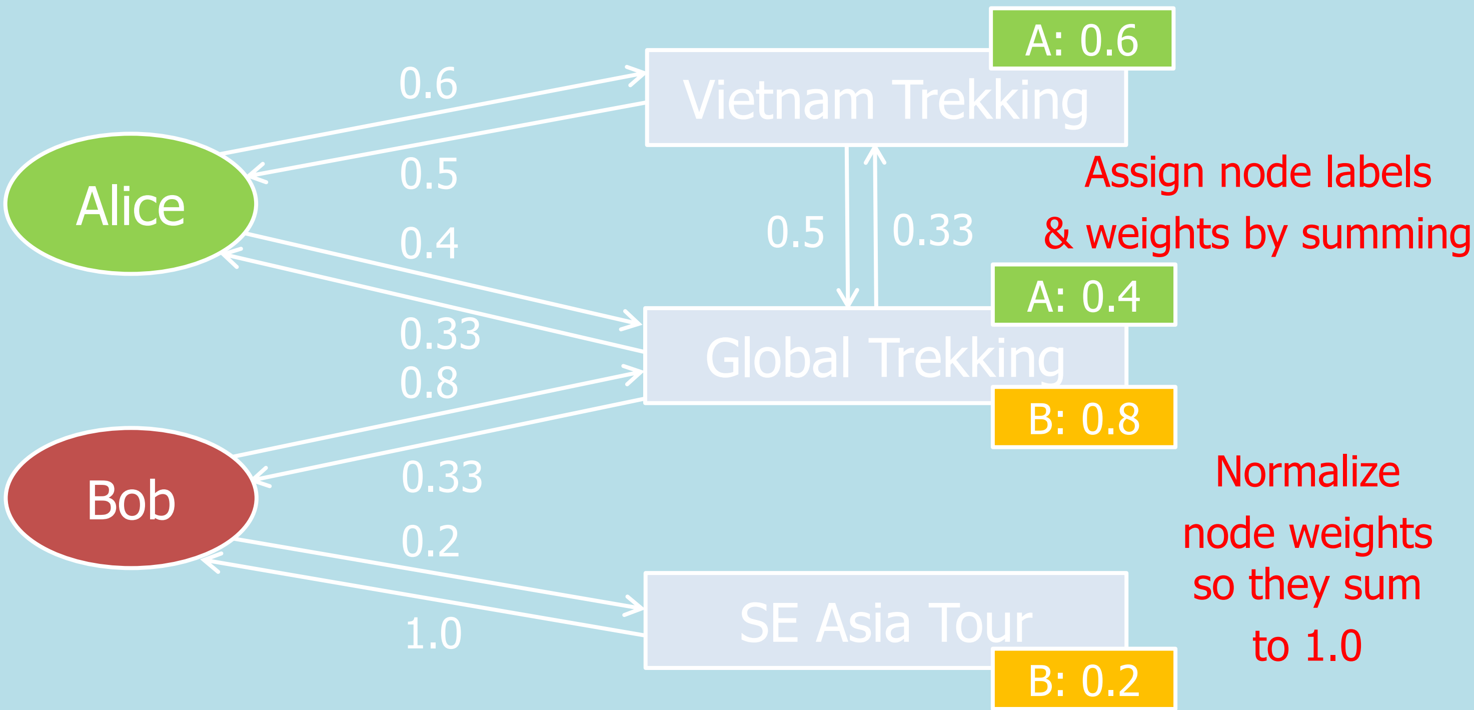
- Iterative process:
  - Compute the likelihood for each vertex v that a user x, in a random walk, will arrive at v – call this the probability of $l_x \in L$ associated with node v

Adsorption: Propagating labels



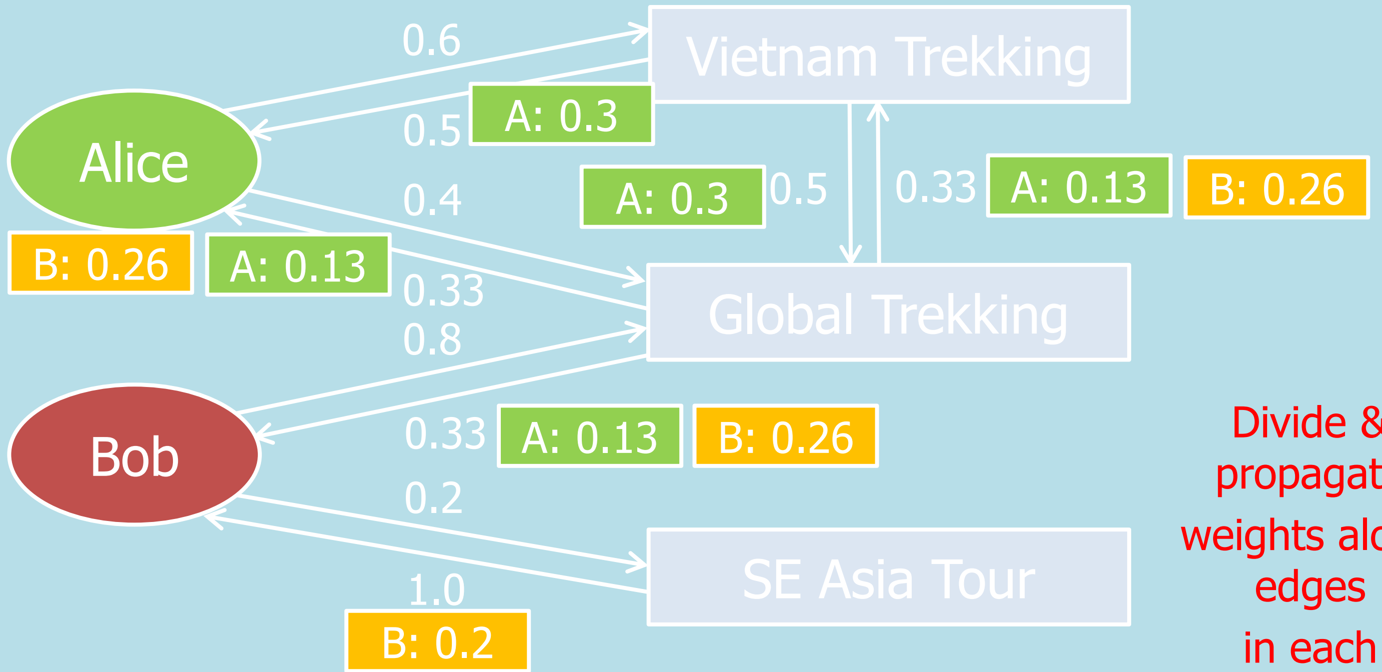
- Iterative process:
  - Compute the likelihood for each vertex v that a user x, in a random walk, will arrive at v – call this the probability of $l_x \in L$ associated with node v

Adsorption: Propagating labels



- Iterative process:
  - Compute the likelihood for each vertex v that a user x, in a random walk, will arrive at v – call this the probability of $l_x \in L$ associated with node v

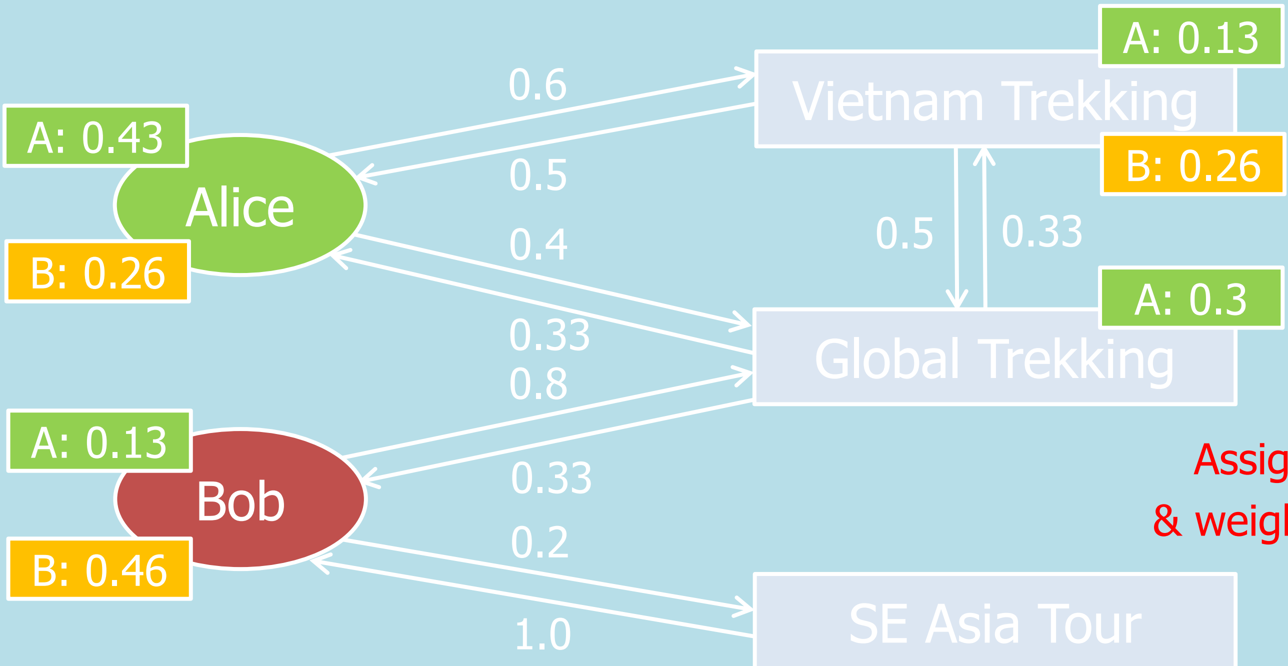
Adsorption: Propagating labels



Divide & propagate weights along edges in each direction

- Iterative process:
  - Compute the likelihood for each vertex v that a user x, in a random walk, will arrive at v – call this the probability of $l_x \in L$ associated with node v

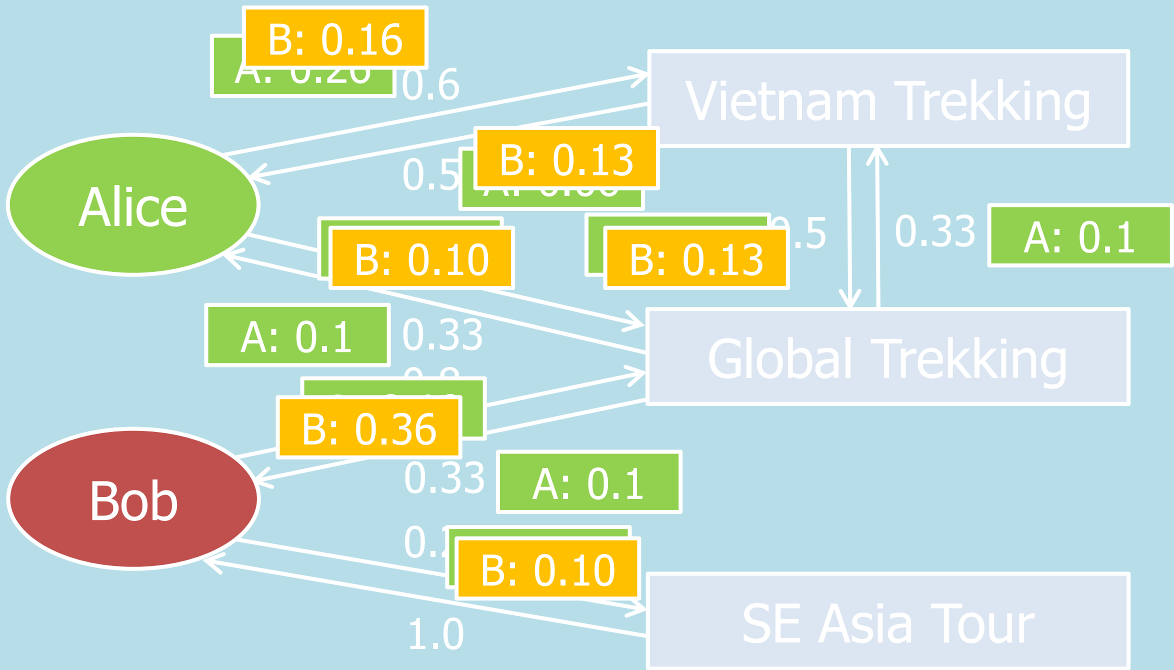
Adsorption: Propagating labels



Assign node labels
& weights by summing

- Iterative process:
  - Compute the likelihood for each vertex v that a user x, in a random walk, will arrive at v – call this the probability of $l_x \in L$ associated with node v

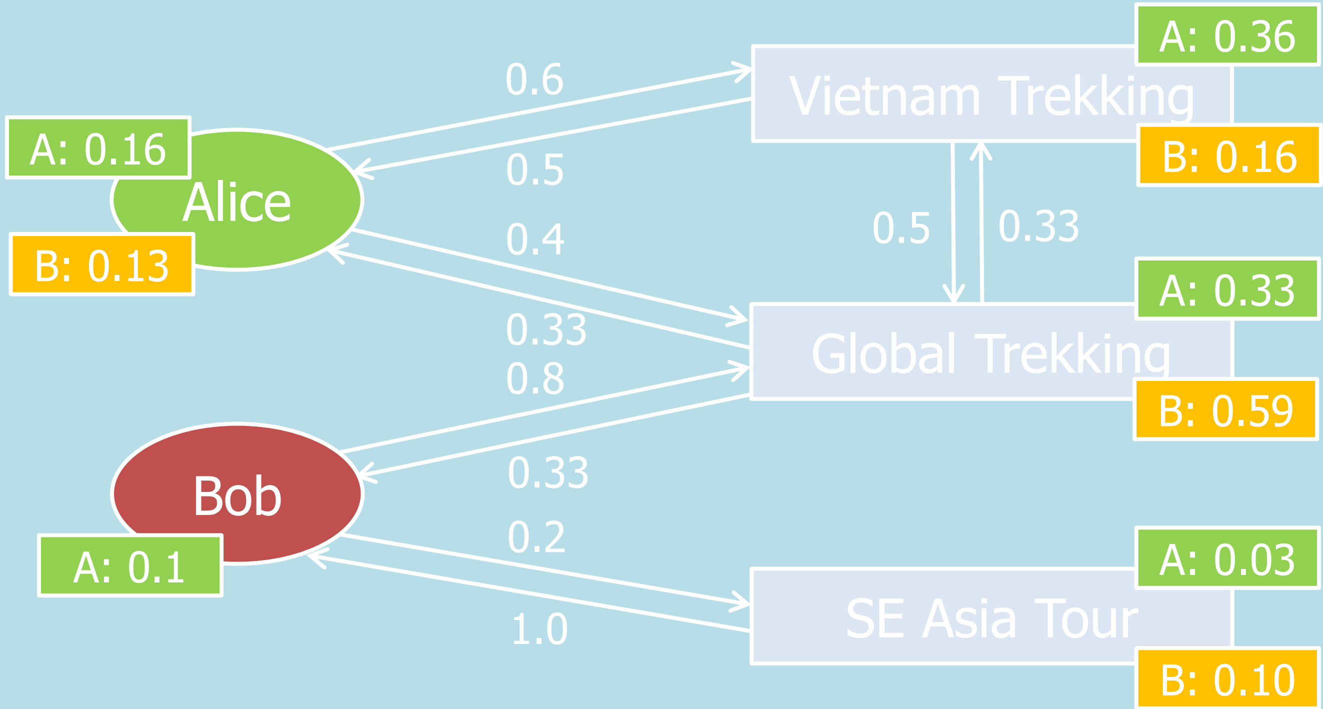
Adsorption: Propagating labels



Repeat...

- Iterative process:
  - Compute the likelihood for each vertex v that a user x, in a random walk, will arrive at v – call this the probability of $l_x \in L$ associated with node v

Adsorption: Propagating labels



… until convergence

- Iterative process:
  - Compute the likelihood for each vertex v that a user x, in a random walk, will arrive at v – call this the probability of $l_x \in L$ associated with node v

Cork Complex Systems Lab

Adsorption algorithm formulation

- Inputs: $G = (V, E, w)$ where $w : E \rightarrow \Re$;
  L: set of labels; $V_L \subseteq V$: nodes with labels

- Repeat
  foreach $v \in V$ do

  $$L_v^{new} = \sum_u w(u,v) L_u$$

  normalize L_v to have unit L_1 norm
  until convergence

- Output: Distributions $\{L_v \mid v \in V\}$

Cork Complex Systems Lab

Applications of Adsorption

- Recommendation (YouTube)

- Discovering relationships among data:
  - Classifying types of objects
  - Finding labels for columns in tables
  - Finding similar / related concepts in different tables or Web pages

Recap and Take-aways

- Whirlwind tour of common kinds of algorithms used on the Web
  - Path analysis: route planning, games, keyword search, etc.
  - Clustering and classification: mining, recommendations, spam filtering, context-sensitive search, ad placement, etc.
  - Link analysis: ranking, recommendations, ad placement

- Many such algorithms (though not all) have a reasonably straightforward, often iterative, MapReduce formulation

Cork Complex Systems Lab