Recommendation and ranking

Mark Jelasity

Recommendation

- We have a set of users and a set of items (movies, books, etc)
- Given a set of ratings by some users on some items,
  - approximate the unknown ratings (ie the unknown elements of user-item matrix),
  - or maybe just find a set of unrated items for a user that are predicted to be rated high by that user
- Machine learning problem
  - Training data: known ratings

Collaborative filtering

- A recommender system that makes use of data from many users (collaboration) to predict the taste of each user
- Memory based
  - Similarity measures between, for example, users (correlation or cosine similarity, etc)
  - For example, find the k nearest neighbors (k-nn), and use the ratings by those users to calculate any missing rating (weighted average, majority, etc)
  - One can use item similarity too, etc

Collaborative filtering

- Memory based methods, pros
  - Simple, good enough, can build social links too, incremental
- Memory based methods, cons
  - Sparse data is a problem (how to calculate similarity?)
  - Users are in reality interested in a mixture of topics, and very few users are interested in exactly the same mixture, so basing everything on similarity is simplistic

Collaborative filtering

- Model based methods
  - Singular value decomposition (SVD), latent Dirichlet allocation (LDA), etc, on the user-item matrix
  - machine learning methods, such as support vector machines (SVM), neural nets, etc
- Model based methods can handle sparsity but are more complex and expensive
- In this lecture we stick to memory based methods
  - The key is to find K-nn neighbors!

P2P collaborative filtering

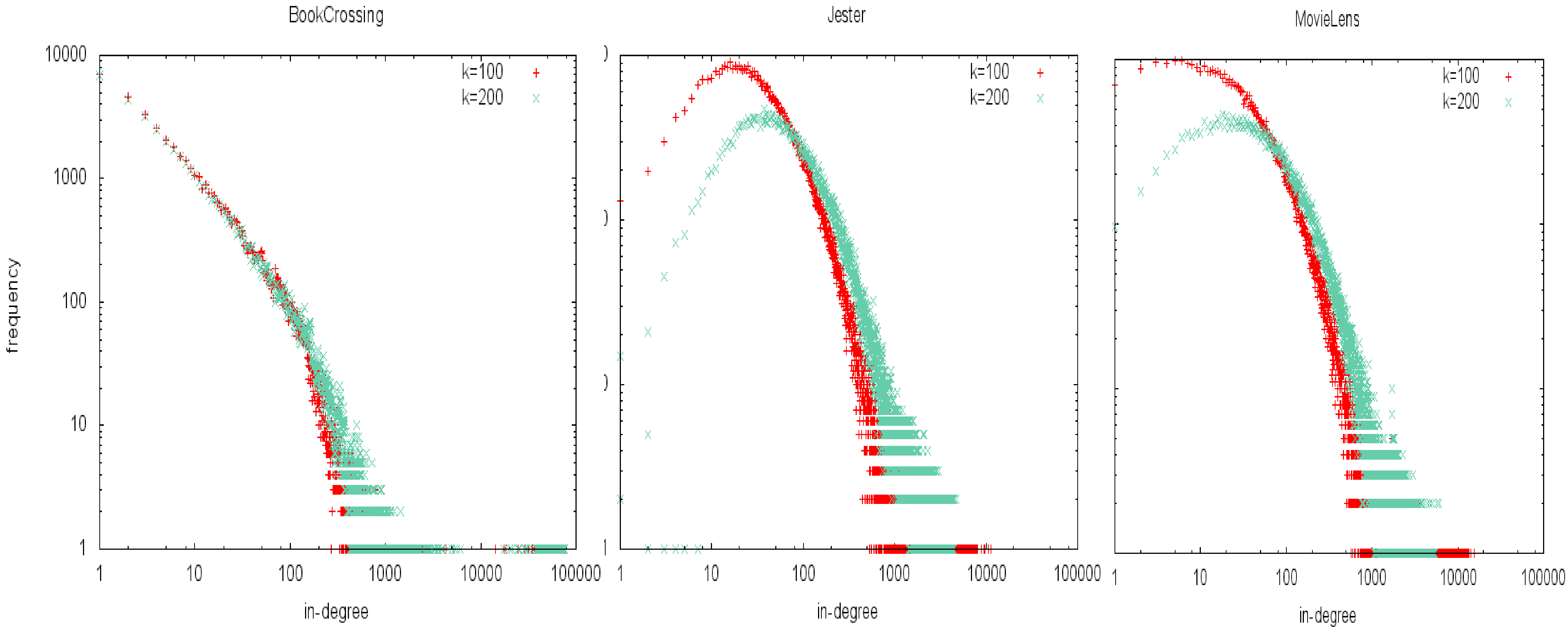
- User-based collaborative filtering in a P2P environment
  - Find “similar” users
  - Use a weighted average of these users' recommendation as a prediction
- Not the best method, but it's very simple and naturally P2P
- Practical P2P aspects
  - Efficiency
  - Convergence
  - Load balancing
  - Parallel versions of existing and novel algorithms

Properties of data sets

- Sparsity and #items are very different
- Minimal number of item evaluations is very different

	MovieLens	Jester	BookCrossing
# users	71,567	73,421	77,806
# items	10,681	100	185,974
size of train	9,301,274	3,695,834	397,011
sparsity	1.2168%	50.3376%	0.0027%
size of eval	698,780	440,526	36,660
eval/train	7.5127%	11.9195%	9.2340%
# items $\geq$	20	15	1
rate set	$1, \ldots, 5$	$-10, \ldots, 10$	$1, \ldots, 10$
MAE(med)	0.93948	4.52645	2.43277

In-degree distribution of kNN graphs



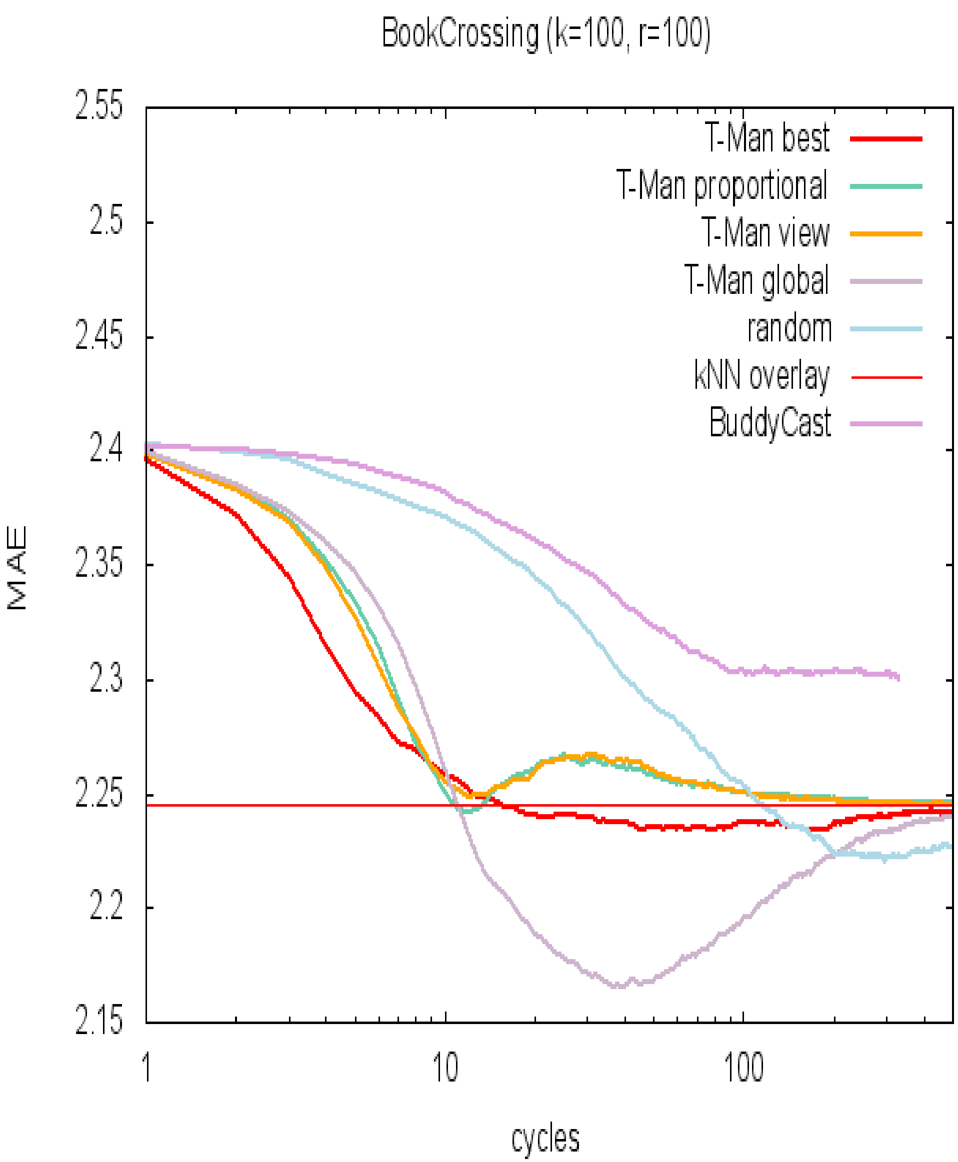
Algorithms

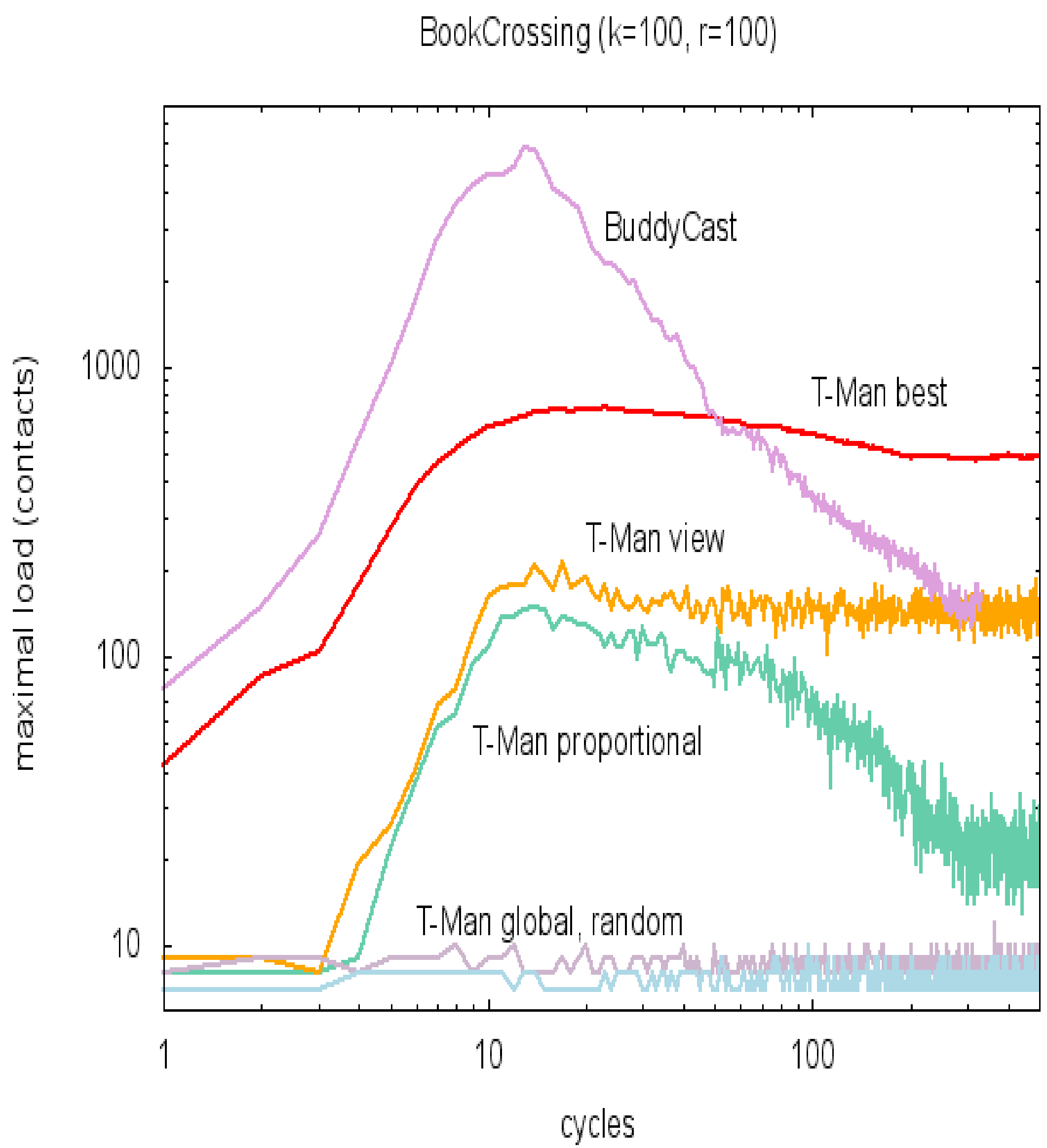
- BuddyCast
  - We use the taste buddy list for recommendations
  - Block list has limit of 100 for feasible simulation...
- Random samples
  - Periodically get r random users
  - Add these to the current list of known users
  - Pick the k most similar users from the known users and throw away the rest
  - This converges to kNN graph, although slowly

Algorithms

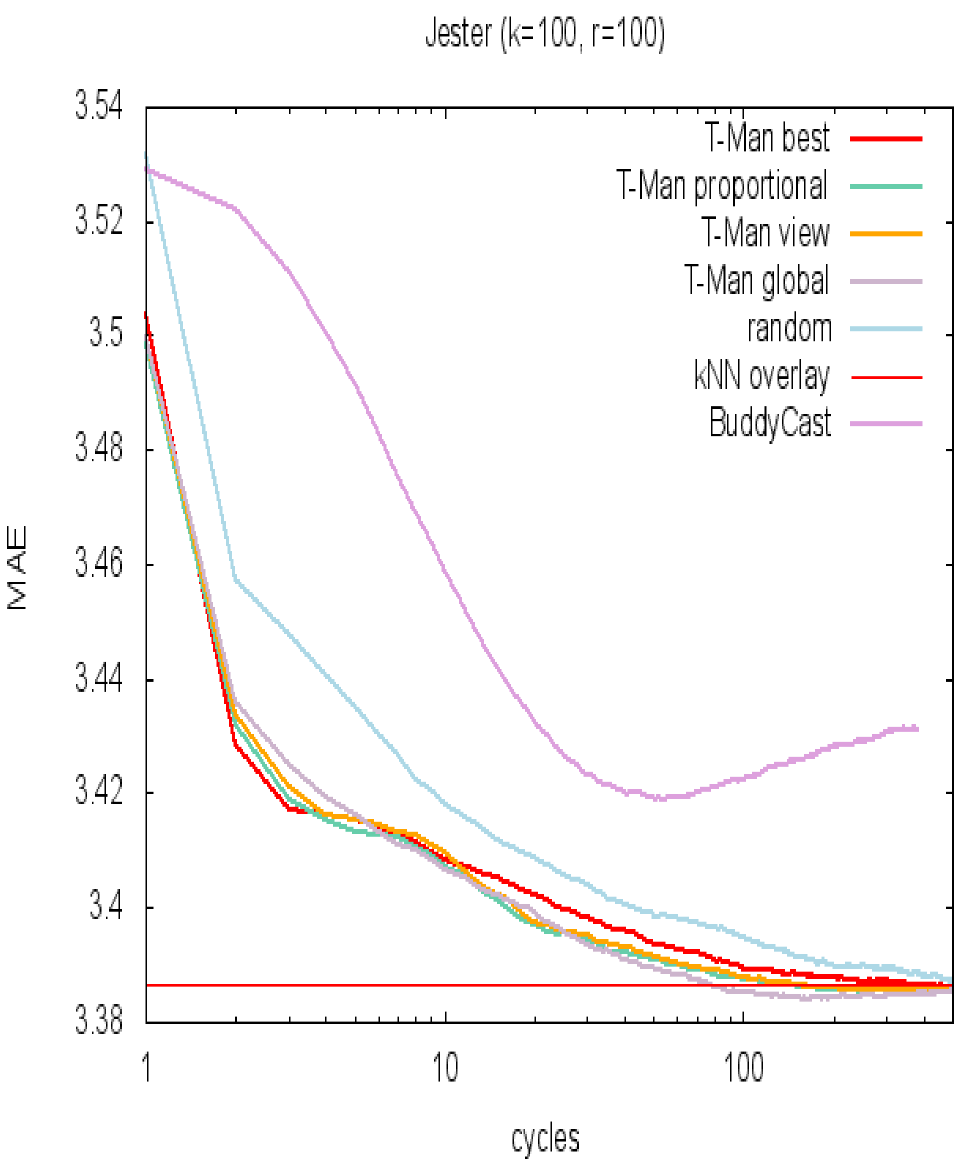
- T-Man view exchange
  - Select a peer to exchange the k known users with
  - Merge the two views and keep the closest k users
- Peer selection methods we looked at
  - **Global**: pick a random peer from the network
  - **View**: pick a random peer from the view
  - **Best**: pick the closest peer
  - **Proportional**: from view, with probability inversely proportional to the load experienced by the peer

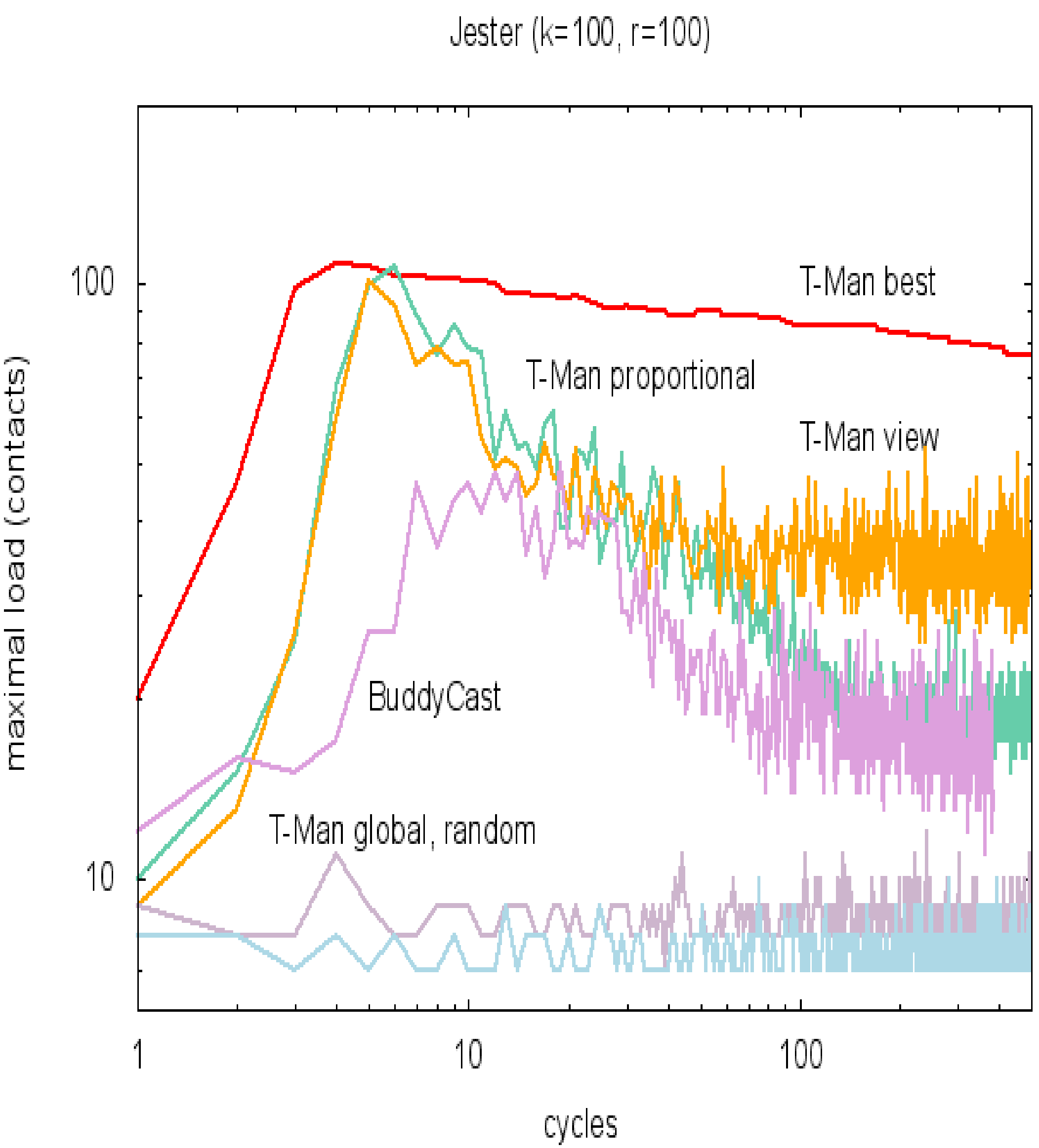
BookCrossing database



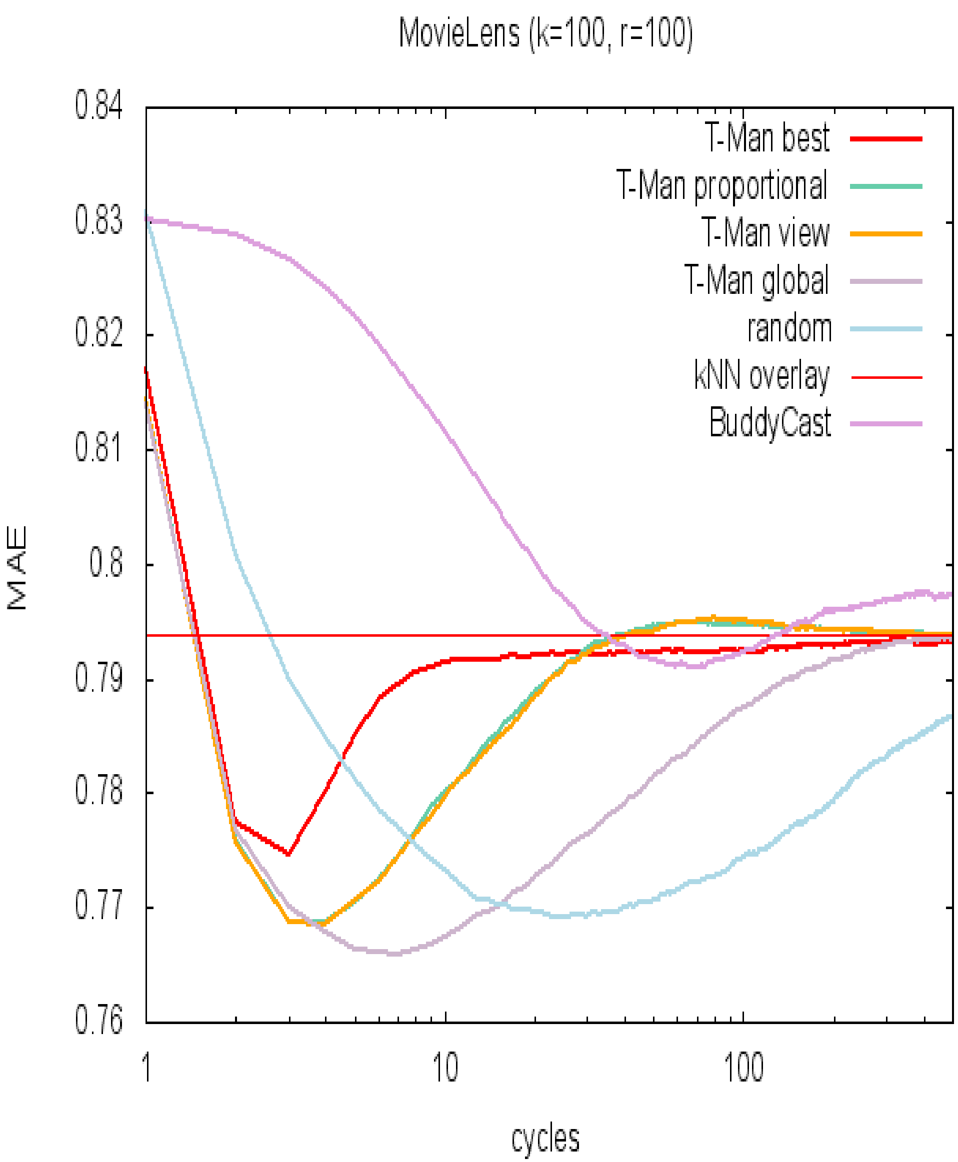


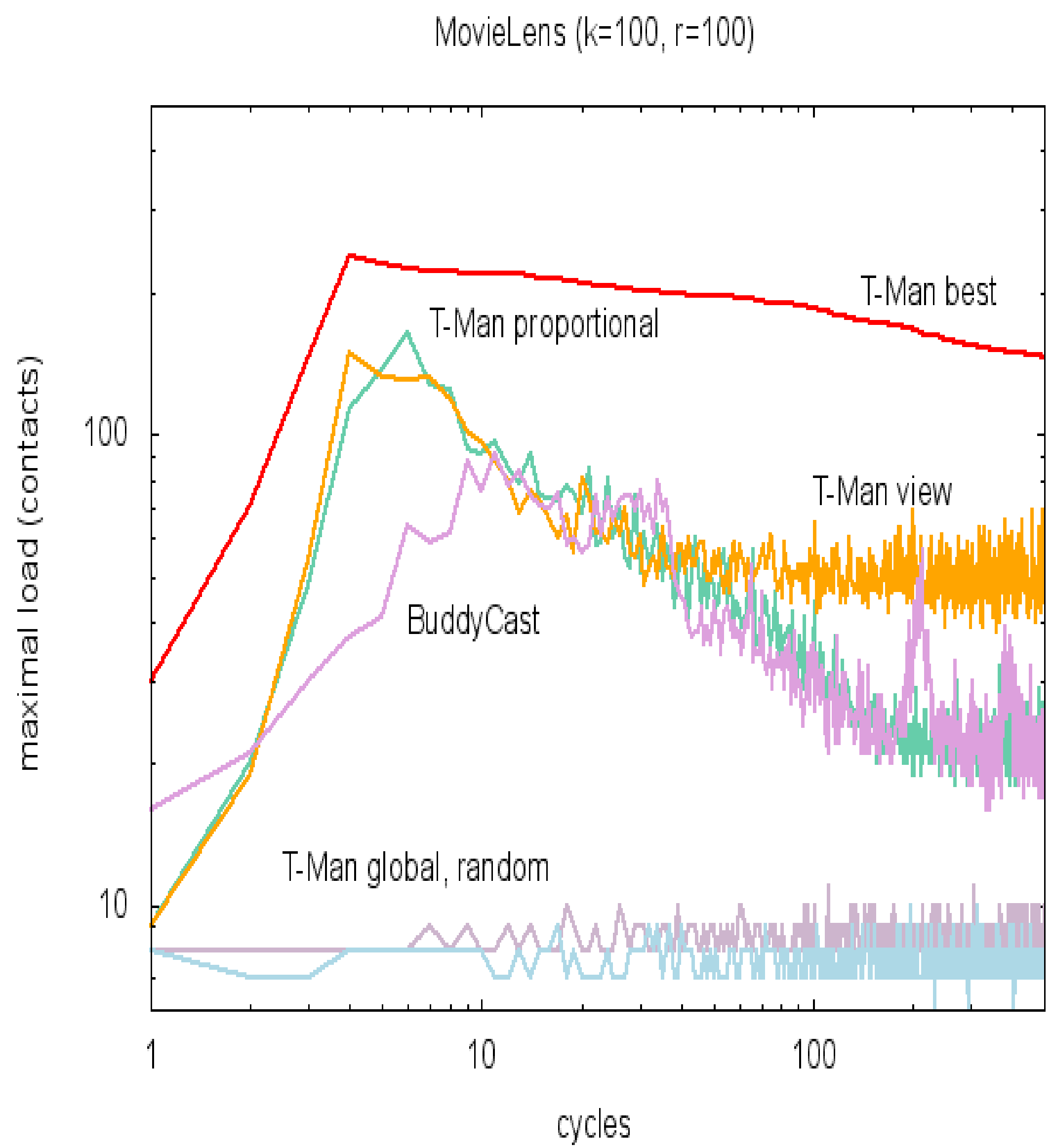
Jester database





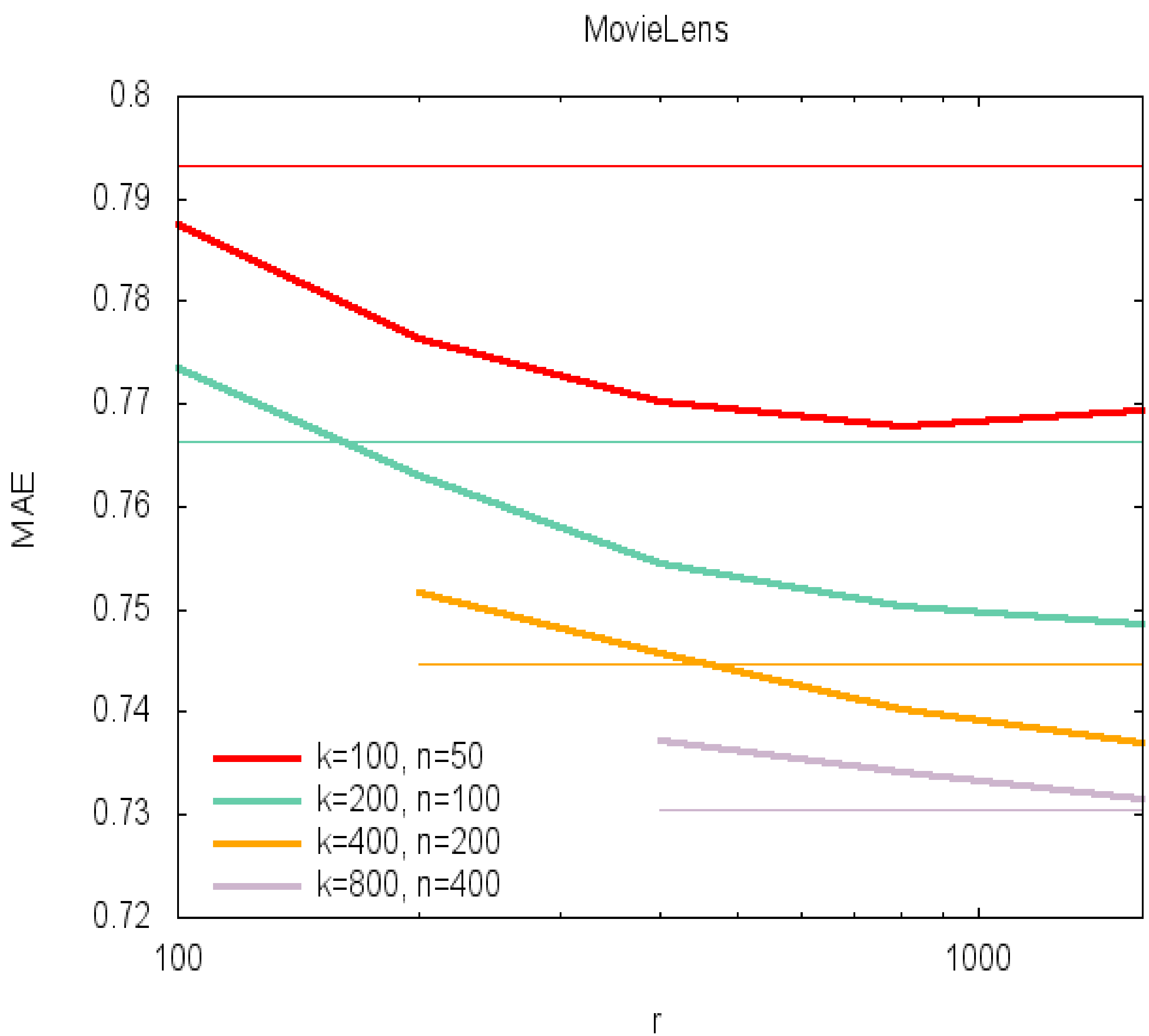
MovieLens database





Not so close is sometimes better

- If k is too small, then it is better to add neighbors that are a bit further away (too similar is not good)



Conclusions

- KNN similarity graphs can have long tails and therefore can induce unbalanced load
- Fully random communication combined with view exchange based convergence seems to be best (T-Man + global view selection)
- Sometimes it is better to use random samples too instead of only the top-k neighborhood.
- It is an open problem to develop P2P versions of other classes of recommender algorithms

Ranking and recommendation

- Recommendation is personalized by definition
- Ranking is global
  - Can be thought of as “default” ratings, aggregated over large populations of users
- Google is moving towards becoming a recommender service these days!
- *Are there globally valid ratings at all in some domain?*

Ranking algorithms

- We look at “user item matrices” again, but this time they are binary
  - “Users” and “items” are the same (eg web pages)
  - Rating is binary (page *a* links to page *b*, or not)
  - In fact, this defines directed graphs (maybe weighted, but often not), eg the WWW.
- Accordingly, ranking algorithms are expressed as graph algorithms. (The reason is that often graphs are all what we have, eg social graphs, web graph etc.)

Ranking algorithms

- Again, there are a large variety of these
  - Centrality indices (degree, betweenness, etc)
  - Eigenvector-based rankings (eg, PageRank)
  - Model based ranking
    - **learning to rank based on available large training databases collected and rated by hand**
- We stick to eigenvector-based methods in this lecture
  - Elegant, powerful, efficient, wide applicability

Applications of eigenvectors

- eigenvector centrality (sociology)
  - my importance is depends on the importance of those I know
- PageRank (Google web search)
  - my usefulness (rank) depends on the usefulness of the pages I'm connected to

$$x_i \propto \sum_{j=1}^{n} a_{ij} x_j$$

Applications of eigenvectors

- EigenTrust (trust building in p2p networks)
  - I have high(low) reputation if I have high reputation for peers that have high(low) reputation
- spectral graph layout
  - my ideal position depends on the ideal position of my neighbors

$$x_i \propto \sum_{j=1}^{n} a_{ij} x_j$$

PageRank

- The mathematical form is eigenvector calculation: $Ax=\lambda x$
- For PageRank, the A matrix is given by the raw normalized, “desinkified” adjecency matrix B and some adjustments: for a $0<d<1$ we want

$$x_i = d\left[\sum_{j=1}^{n} b_{ji} x_j\right] + \frac{(1-d)}{n}$$

Personalized PageRank

- Instead of a uniform probability random surfer, we personalize the random surfing step, where r_i is the probability of jumping to page i
- In fact this is a recommender algorithm!

$$x_i = d\left[\sum_{j=1}^{n} b_{ji} x_j\right] + (1-d)\,r_i$$

Power iteration

- simplest method to get the dominant eigenvector
  - iterate matrix multiplication with almost any initial vector, and normalize in the meantime
  - stop when the angle of the vector has converged
- if λ=1 (Markovian processes, random walk) then normalization is not needed
- For a suitable dominant eigenvector *v* and large *m*:

$$x^{(m+1)} = Ax^{(m)} = A^{m+1}x^{(0)} \approx \lambda^{m+1}v$$

HITS algorithm

- Two rankings: authority (*x*) and hubness (*y*)
  - Good hubs point to good authorities
  - Good authorities are pointed to by good hubs
- Let *A* be the adjecency matrix: we need the dominant eigenvectors of $A^T A$ and AA^T

$$x_i \propto \sum_{j=1}^{n} a_{ji} y_j$$

$$y_i \propto \sum_{j=1}^{n} a_{ij} x_j$$

$$x \propto A^T y \propto A^T A x$$

$$y \propto A x \propto A A^T x$$

HITS algorithm

- How about peer review: reviewer quality (x) and paper quality (y)
- a_{ij}: rating of reviewer i of paper j
  - Good reviewers rate good papers high
  - Good papers are rated high by good reviewers
- A is user-item matrix like with recommender systems!

$$x_i \propto \sum_{j=1}^{n} a_{ji} y_j$$

$$y_i \propto \sum_{j=1}^{n} a_{ij} x_j$$

$$x \propto A^T y \propto A^T A x$$

$$y \propto A x \propto A A^T x$$

Mapping the Network to Linear Algebra

- Each network node holds one vector element
- The matrix is in the weights of links
- intuition: matrix vector multiplication can be implemented using local operations

$$x_i^{(m+1)} = \sum_{j=1}^{n} a_{ij} x_j^{(m)}$$

Asynchronous distributed iteration

- If matrix *A* is stochastic and irreducible, this algorithm is known to converge
- asynchronous power iteration (but not completely equivalent)
- We will now consider non-stochastic matrices and propose an algorithm to handle them

loop {Active Thread}
    wait(Δ)
    for each $j \in$ out-neighbors$_i$ **do**
        send $A_{ji} w_i$ to j
    $b_i \leftarrow \sum_{k \in \text{in-neighbors}_i} b_{ki}$
    $w_i \leftarrow b_i$

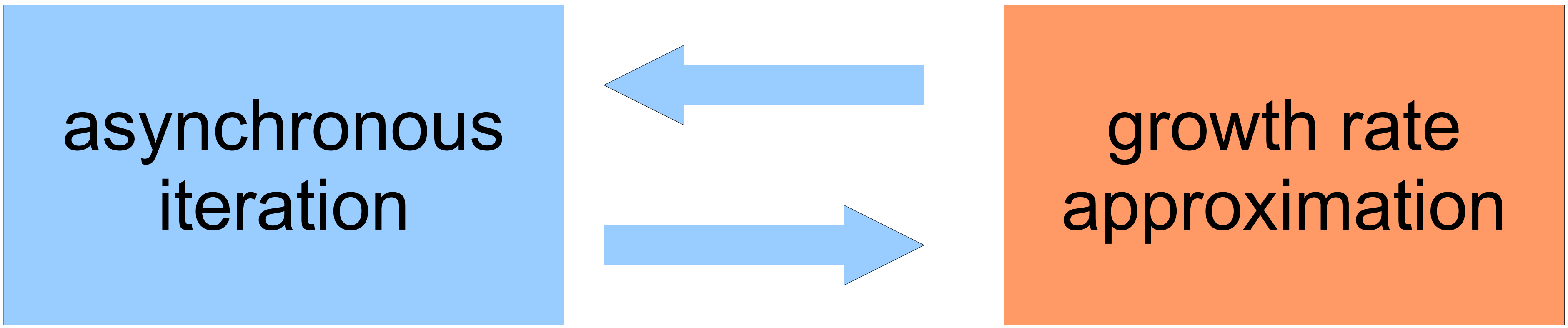
loop {Passive Thread}
    $x \leftarrow$ receive($*$)
    $k \leftarrow$ sender(x)
    $b_{ki} \leftarrow x$

normalization of non-stochastic matrices

- intuition: if $|\lambda|>1$ (or <1) then the power iteration keeps increasing (decreasing) vector length without normalization
- we need to control the length: we approximate growth rate and divide by it
  - safe because eventually little variance among the nodes: converges to λ

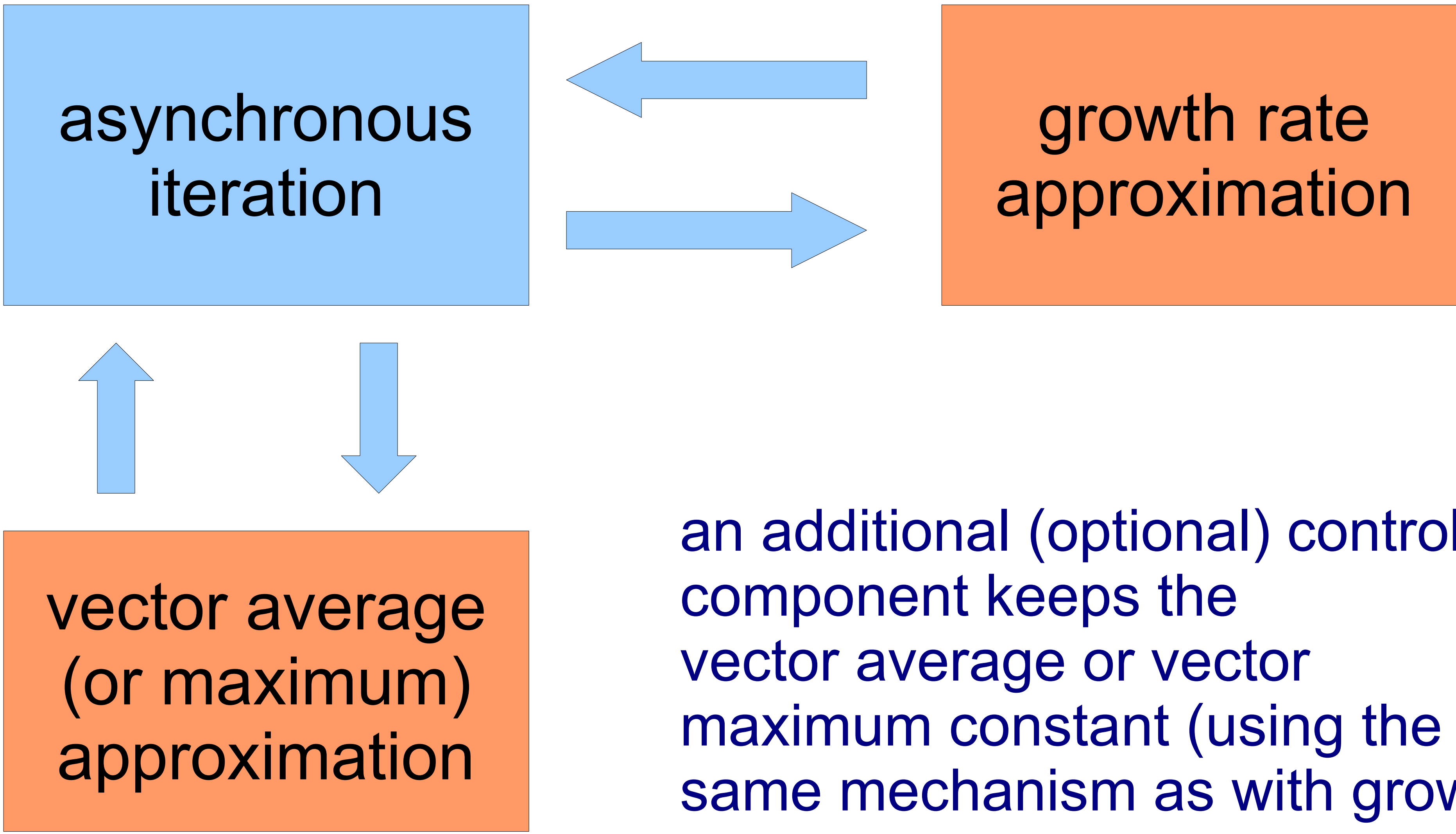
$$\|x^{(m+1)}\| = \|Ax^{(m)}\| = \|A^{m+1}x^{(0)}\| \approx \lambda^{m+1}\|v\|$$

A control component for normalization



- growth rate is approximated through a gossip-based averaging protocol that is run by all nodes beside the asynchronous iteration
  - nodes record their own growth rate and cooperate in calculating the approximate average growth rate

A control component for normalization



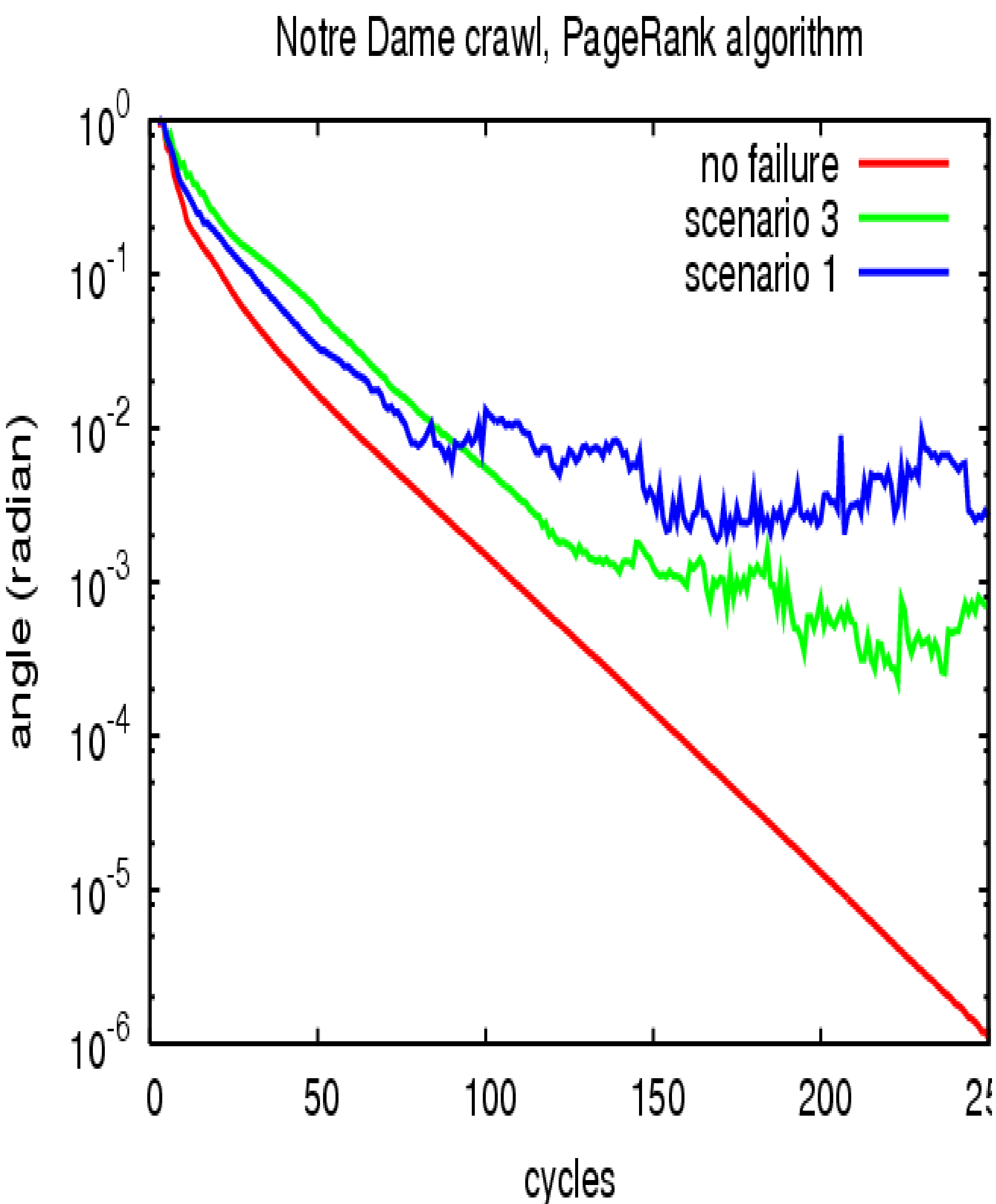
an additional (optional) control component keeps the vector average or vector maximum constant (using the same mechanism as with growth

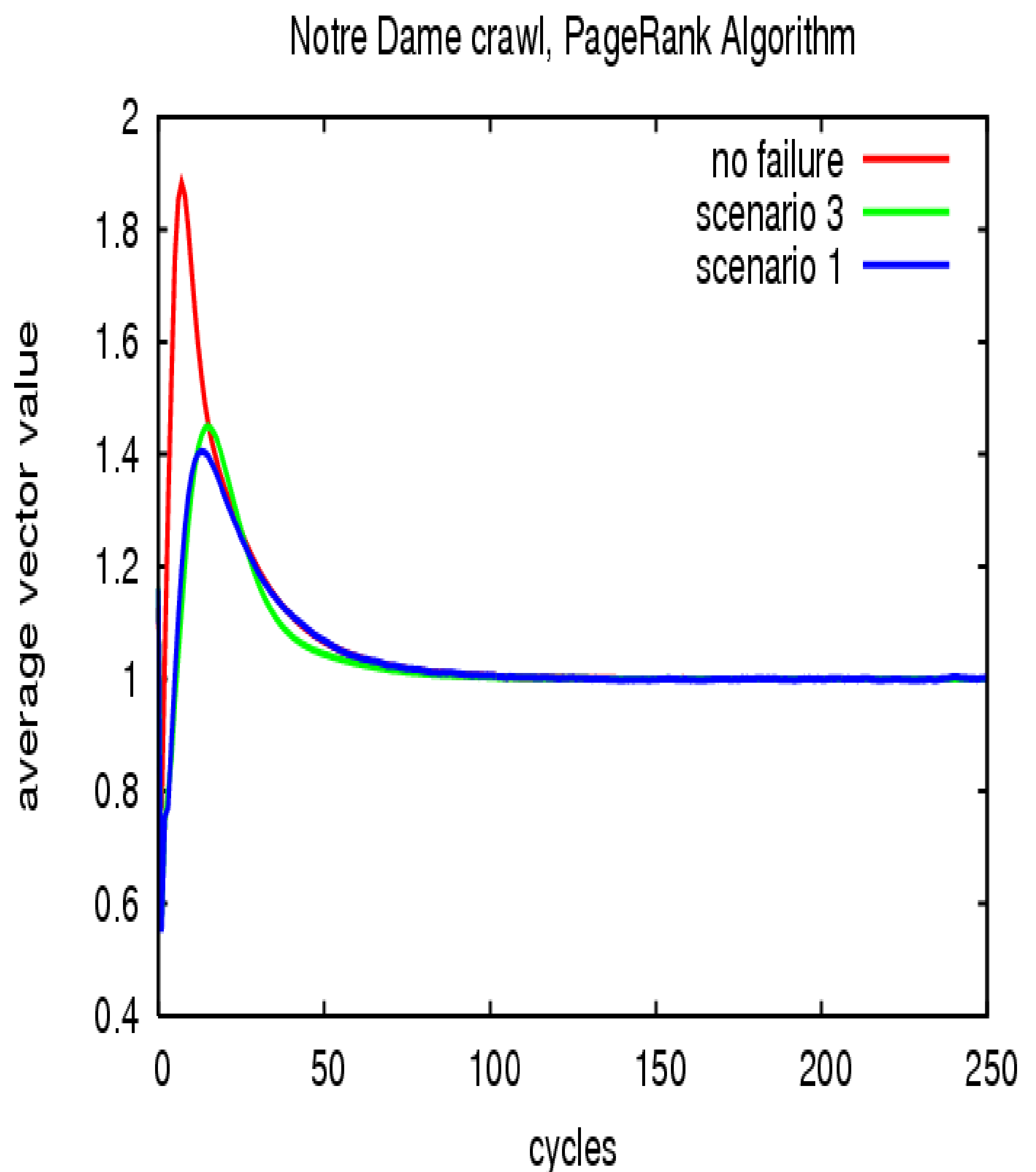
PageRank operator

- PageRank needs random surfer operator to make the graph strongly connected
- this can be implemented using the average of the vector (which we can provide)

$$x_i^{(m+1)} = d\left[\sum_{j=1}^{n} b_{ji} x_j^{(m)}\right] + (1-d)\frac{\|x^{(m)}\|_1}{n}$$

PageRank on Notre Dame crawl data





HITS algorithm

- Power iteration on AA^T and A^TA is equivalent to updating x and y in an alternatig fashion, and normalizing after a pair of updates
- Asynchronous version??
- Gossip-based normalization approach is applicable (still no proof though)

$$x_i^{(m+1)} = \sum_{j=1}^{n} a_{ji} y_j^{(m)}$$

$$y_i^{(m+1)} = \sum_{j=1}^{n} a_{ij} x_j^{(m)}$$

Some thoughts

- Distributed power iteration
  - Applicable in many cases where principal eigenvectors are needed
  - If the graph is sparse, then it is very efficient
- HITS algorithm
  - When applied for single graph, the distributed (alternating) iteration is efficient
  - When applied to a user-item matrix, we have a problem: users might have a location but items do not; not clear how to do an efficient distributed version

Social computer systems

- A large number of large-scale complex computer systems involve human input and decisions, personal data, or directly serve a social purpose
  - Social networking websites
  - Recommender systems
  - Web search
  - Forums, blogs, news
  - Wikipedia
  - BitTorrent (esp. private communities)

Privacy

- List of friends, personal data, preferences, browsing history, purchased items, physical location, etc
- Knowing others' data is good for me
- Others knowing my data is bad for me
- Privacy preserving techniques come to the rescue

Trusted services

- Centralized services need to be trusted
  - They store and process our data
  - They use secret algorithms to answer our queries
  - They might use settings and options we cannot control (eg google personalization)
  - They are often highly available, but can easily be made completely unavailable (by criminals, governments, defamation lawsuits, or software or hardware problems, etc)
  - They are now free, but they cost a lot: can advertising revenue maintain this ecosystem forever? Can net neutrality be maintained forever?

Decentralization

- Decentralized services offer the possibility (but do not guarantee!) that
  - Our data does no leave our computer
  - Our activity is not traced back to us
  - Yet the system still functions at tolerable performance levels
  - The algorithms used are all open and transparent
  - Availability and cost depends only on the availability and cost of the underlying global communication infrastructure
  - Performance degrades gracefully

Motivation

Let us design algorithms that are fully distributed, privacy preserving, and help us build the services we depend on!

Privacy preservation basics

- We are interested in the models over shared data but do not wish to share data
- Degree of distribution
  - A few large database chunks (hospitals, etc)
  - One database record per node (P2P)
- Basic approaches
  - Statistical
  - Cryptographic
  - Relay networks
  - etc

Statistical techniques

- Sequrity in statistical databases
  - Queries for only aggregate data (sum, count, etc)
  - No access to individual records
- Restricting queries
- Perturbation of entries
  - Adding noise to data
  - Swapping attributes among records
  - Replacing attribute values with samples from the same distribution
  - Sampling the query results

Cryptographic techniques

- Secure multi-party computation
  - Compute a function from private inputs
  - Perhaps simplest example: 1-2 oblivious transfer
    - **Node A has attributes x and y**
    - **Node B wants the value of either x or y, say, x.**
    - **Problem: B should get x without learning about the value of y, and A should not learn about what B wanted!**
  - Zero knowledge proofs are related
- Threshold cryptography, secret sharing
  - Collusion needed to uncover private values

Anonymous relay networks

- For example, TOR (P2P relay network)
  - Onion routing to hide the source of queries from servers
  - Supports two-way communication
- Can be used to mask the ownership of data
  - Relay data to a peer
  - Perform computations
  - Share the model

Aspects to consider

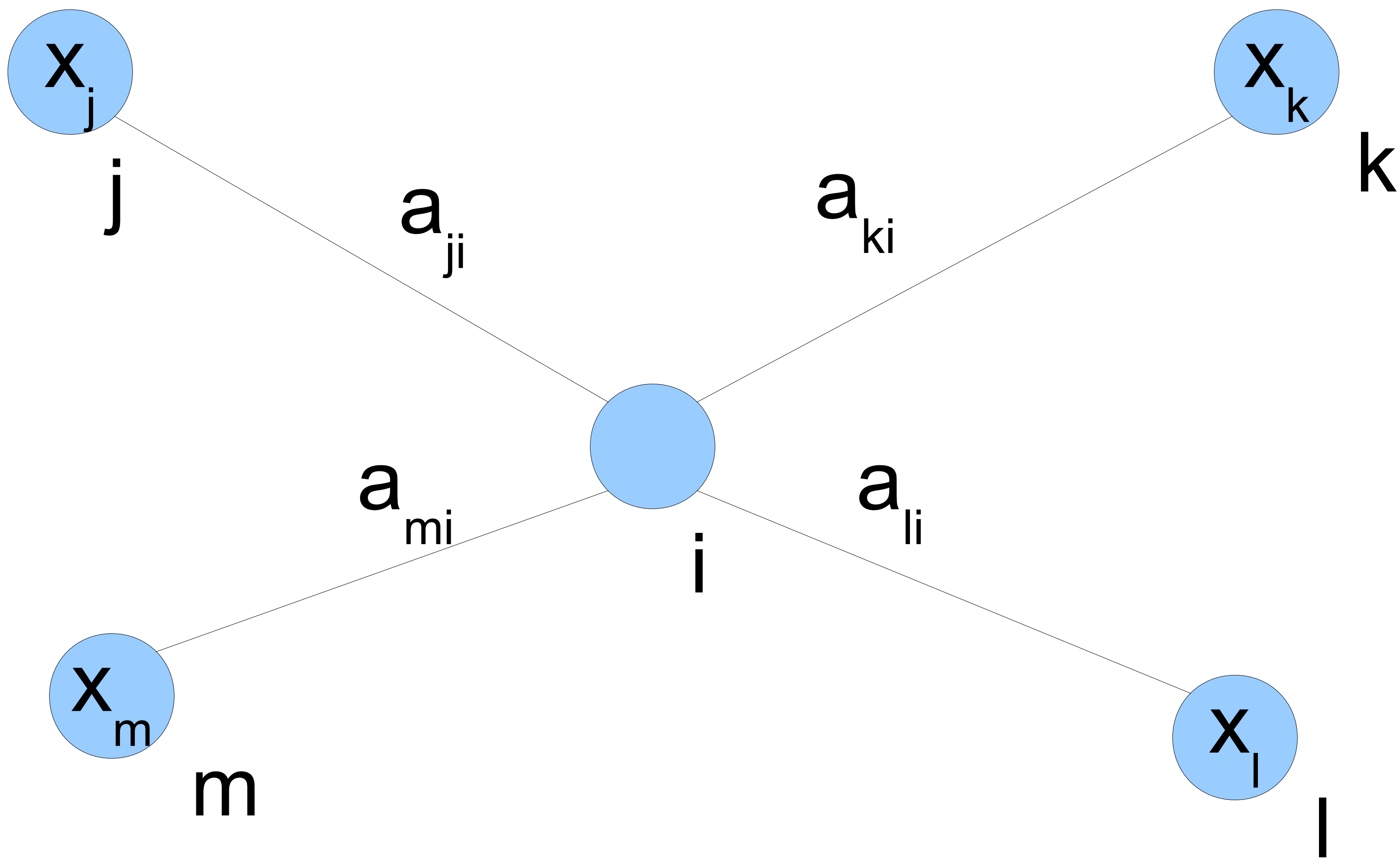
- Adversary models
  - Malicious: can inject false information, can bias the end result
  - Semi-honest: follows the protocol, but wants to steal our data
- Often secure channels are assumed (no eavesdropping)

Privacy preserving power iteration

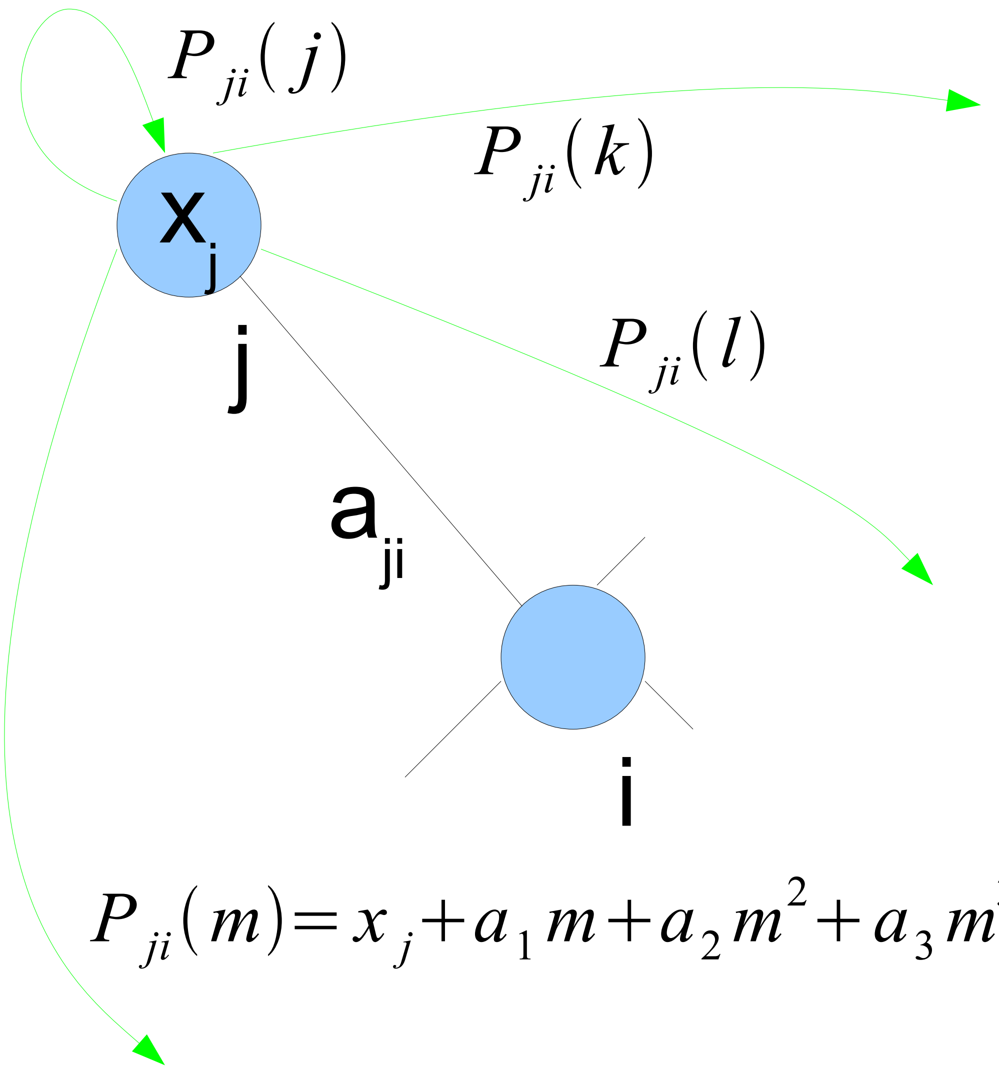
- Each network node holds one vector element
- The matrix is in the weights of links
- The basic primitive is that each node needs the sum of their neighbors' values (individual values are not needed)

$$x_i^{(m+1)} = \sum_{j=1}^{n} a_{ij} x_j^{(m)}$$

Shamir secret sharing



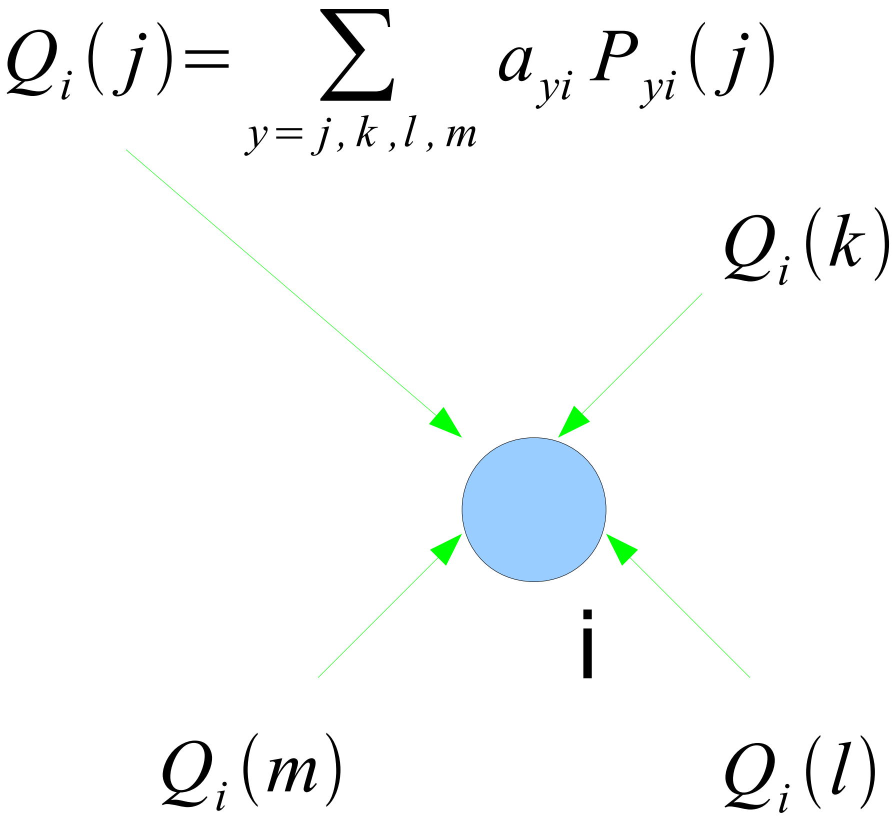
Using Shamir secret sharing



$$P_{ji}(m) = x_j + a_1 m + a_2 m^2 + a_3 m^3$$

- Every neighbor j of i generates a polinomial P_j of degree d_i-1and sends it to the neighbors l of i evaluated in l.
- The coefficients are random, except the constant

Using Shamir secret sharing



- Every neighbor j of i sends the sum of the polinomials it received to l.
- The constant coefficient, which is the weighted some we want, can be determined using the d_i points of the polinomial

Some open questions

- How about desirable features of power iteration such as
  - Asynchronicity?
  - robustness and flexibility ?
- How about the normalization component?
- How about HITS, collaborative filtering, etc?