Mining Data that Changes

17 July 2014

max planck institut
informatik

Data is Not Static

- Data is not static
  - New transactions, new friends, stop following somebody in Twitter, ...
- But most data mining algorithms assume static data
  - Even a minor change requires a full-blown re-computation

Types of Changing Data

1. New observations are added
    - New items are bought, new movies are rated
    - The existing data doesn’t change
2. Only part of the data is seen at once
3. Old observations are altered
    - Changes in friendship relations

Types of Changing-Data Algorithms

- **On-line** algorithms get new data during their execution
  - Good answer at any given point
  - Usually old data is not altered
- **Streaming** algorithms can only see a part of the data at once
  - Single-pass (or limited number of passes), limited memory
- **Dynamic** algorithms’ data is changed constantly
  - More, less, or altered

Measures of Goodness

- **Competitive ratio** is the ratio of the (non-static) answer to the **optimal** off-line answer
  - Problem can be NP-hard in off-line
  - What’s the cost of uncertainty
- **Insertion** and **deletion times** measure the time it takes to update a solution
- **Space complexity** tells how much space the algorithm needs

Concept Drift

- Over time, users' opinions and preferences change
  - This is called **concept drift**
- Mining algorithms need to counter it
  - Typically data observed earlier weights less when computing the fit

On-Line vs. Streaming

On-line

- Must give good answers at all times
- Can go back to already-seen data
- Assumes all data fits to memory

Streaming

- Can wait until the end of the stream
- Cannot go back to already-seen data
- Assumes data is too big to fit to memory

On-Line vs. Dynamic

On-line	Dynamic
• Already-seen data doesn’t change	• Data is changed all the time
• More focused on competitive ratio	• More focused on efficient addition and deletion
• Cannot change already-made decisions	• Can revert already-made decisions

Example: Matrix Factorization

- On-line matrix factorization: new rows/columns are added and the factorization needs to be updated accordingly
- Streaming matrix factorization: factors need to be build by seeing only a small fraction of the matrix at a time
- Dynamic matrix factorization: matrix's values are changed (or added/removed) and the factorization needs to be updated accordingly

On-Line Examples

- Operating systems’ cache algorithms
- Ski rental problem
- Updating matrix factorizations with new rows
  - I.e. LSI/pLSI with new documents

Streaming Examples

- How many distinct elements we’ve seen?
- What are the most frequent items we’ve seen?
- Keep up the cluster centroids over a stream

Dynamic Examples

- After insertion and deletion of edges of a graph, maintain its parameters:
  - Connectivity, diameter, max. degree, shortest paths, ...
- Maintain clustering with insertions and deletion

Streaming

Sliding Windows

- Streaming algorithms work either per element or with **sliding windows**
  - Window = last k items seen
    - Window size = memory consumption
  - "What is X in the current window?"

Example Algorithm: The 0th Moment

- **Problem:** How many distinct elements are in the stream?
  - Too many that we could store them all, must estimate
- Idea: store a value that lets us estimate the number of distinct elements
  - Store many of the values for improved estimate

The Flajolet–Martin Algorithm

- Hash element *a* with hash function *h* and let *R* be the number of trailing zeros in *h*(*a*)
  - Assume *h* has large-enough range (e.g. 64 bits)
  - The estimate for # of distinct elements is 2^R
- Clearly space-efficient
  - Need to store only one integer, *R*

Flajolet, P., & Nigel Martin, G. (1985). Probabilistic counting algorithms for data base applications. Journal of Computer and System Sciences, 31(2), 182–209. doi: 10.1016/0022-0000(85)90041-8

Does Flajolet–Martin Work?

- Assume the stream elements come u.a.r.
- Let $trail(h(a))$ be the number of trailing 0s
- $\Pr[trail(h(a)) \geq r] = 2^{-r}$
- If stream has m distinct elements, Pr["For all distinct elements, $trail(h(a)) \leq r$"] $= (1 - 2^{-r})^{m}$
  - Approximately $\exp(-m2^{-r})$ for large-enough r
  - Hence: Pr["We have seen a s.t. $trail(h(a)) \geq r$"]
    - approaches 1 if $m \gg 2^{r}$ and approaches 0 if $m \ll 2^{r}$

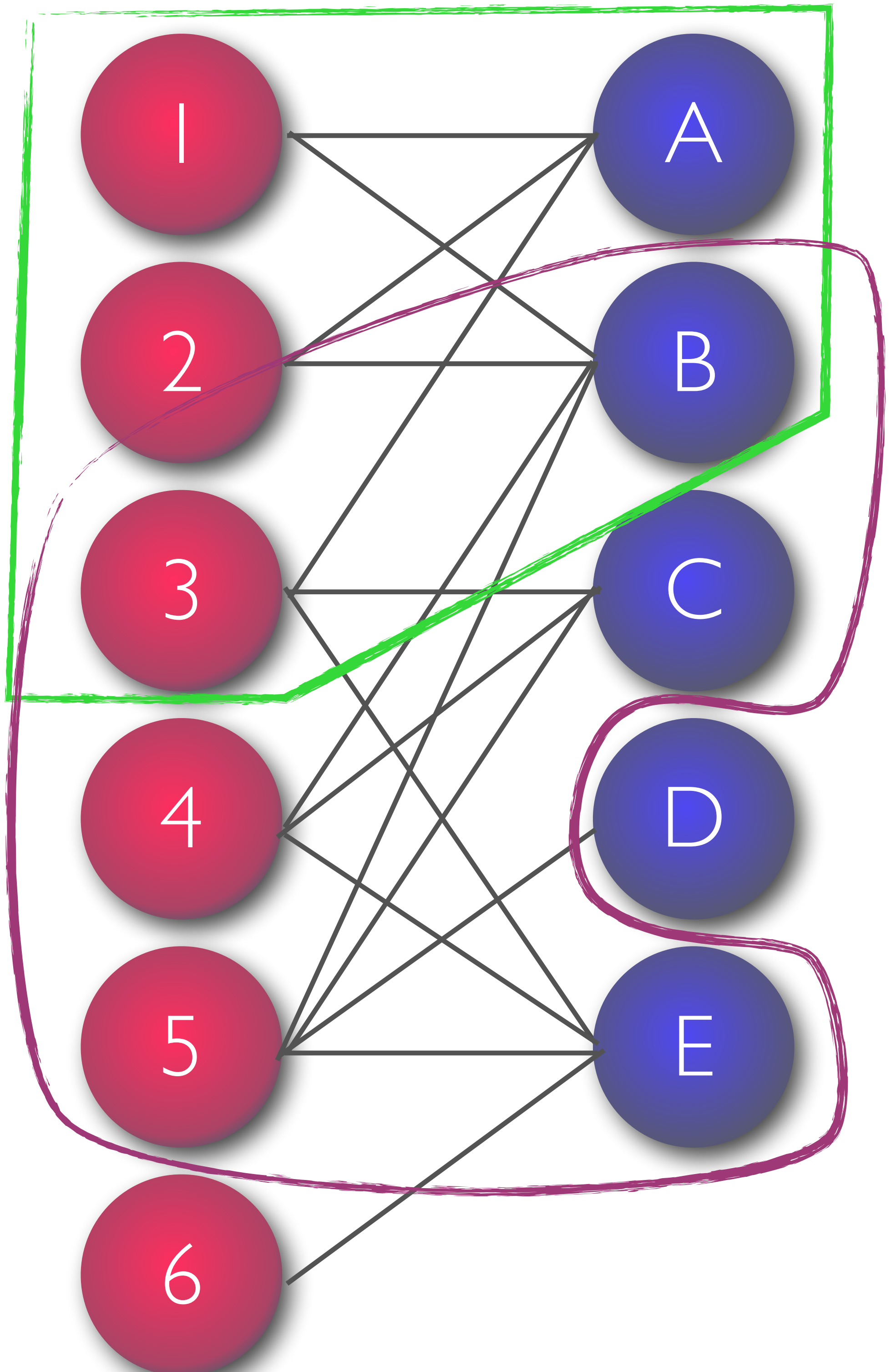
Many Hash Functions

- Take average?
  - A single r that's too high at least doubles the estimate
    $\Rightarrow$ the expected value is infinite
- Take median?
  - Doesn't suffer from outliers
  - But it's always a power of two
    $\Rightarrow$ adding hash functions won't get us closer than that
- Solution: group hash functions in small groups, take their average and the median of the averages
  - Group size preferably $\approx \log m$

Example Dynamic Algorithm

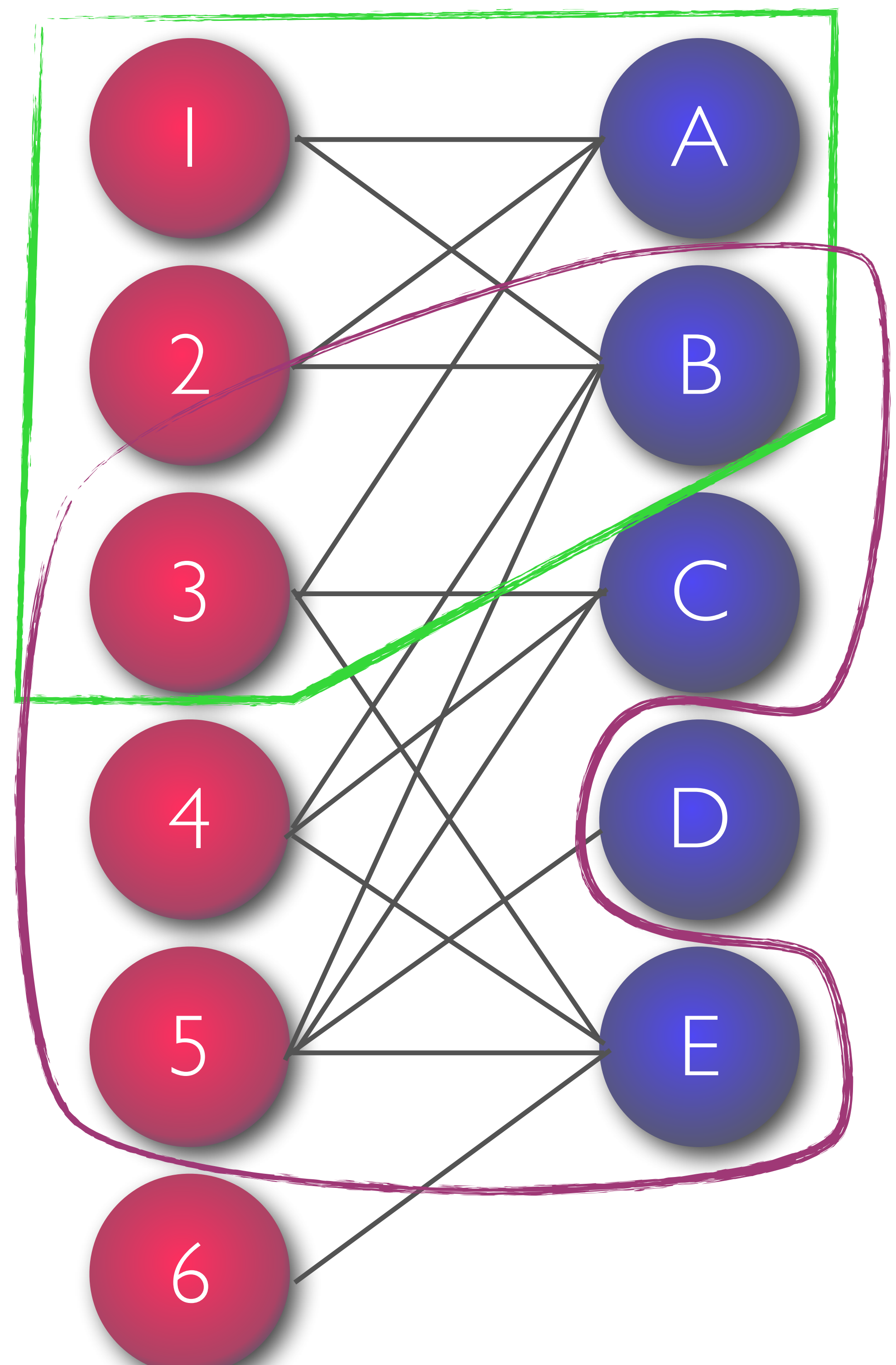
Users and Tweets

- Users follow tweets
  - A bipartite graph
- We want to know (approximate) bicliques of users who follow similar tweeters



Boolean Matrix

1	1	0	0	0
1	1	0	0	0
1	0	1	0	1
0	1	1	0	1
0	1	1	1	1
0	0	0	0	1



Boolean Matrix Factorizations

$$
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\approx
\begin{bmatrix}
1 & 0 \\
1 & 0 \\
1 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 0
\end{bmatrix}
\circ
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1
\end{bmatrix}
$$

Boolean Matrix Factorizations

$$\begin{pmatrix} & & & & & 1 & 1 & 0 & 0 & 0 \\ & & & & & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Fully Dynamic Setup

- Can handle both addition and deletion of vertices and edges
  - Deletion is harder to handle
- Can adjust the number of bicliques
  - Based on the MDL principle

Miettinen, P. (2012). Dynamic Boolean Matrix Factorizations (pp. 519–528). Presented at the 12th IEEE International Conference on Data Mining. doi:10.1109/ICDM.2012.118
Miettinen, P. (2013). Fully dynamic quasi-biclique edge covers via Boolean matrix factorizations (pp. 17–24). Presented at the 2013 Workshop on Dynamic Networks Management and Mining, ACM. doi:10.1145/2489247.2489250

This Ain’t Prediction

- The goal is not to predict new edges, but to adapt to the changes
  - The quality is computed on observed edges
  - Being good at predicting helps adapting, though

First Attempt

- Re-compute the factorization after every addition
- Too slow
- Too much effort given the minimal change

Example

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

$$\begin{bmatrix}1&1&0&0&0\\1&1&0&0&0\\1&0&1&0&1\\0&1&1&0&1\\0&1&1&1&1\\0&0&0&0&1\end{bmatrix} \approx \begin{bmatrix}1&0\\1&0\\1&1\\0&1\\0&1\\0&0\end{bmatrix} \quad \begin{bmatrix}1&1&0&0&0\\1&1&0&0&0\\1&1&1&0&1\\0&1&1&0&1\\0&1&1&0&1\\0&0&0&0&0\end{bmatrix}$$

Step 1: Remove

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

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

Step 2: Add

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

$$\begin{bmatrix}1&1&0&0&0\\1&1&0&1&0\\1&0&1&0&1\\0&1&1&0&1\\0&1&1&1&1\\0&0&0&0&0\end{bmatrix} \approx \begin{bmatrix}1&0\\1&0\\1&1\\0&1\\0&1\\0&0\end{bmatrix} \quad \begin{bmatrix}1&1&0&0&0\\1&1&0&0&0\\1&1&1&0&1\\0&1&1&0&1\\0&1&1&0&1\\0&0&0&0&0\end{bmatrix}$$

Step 3: Remove

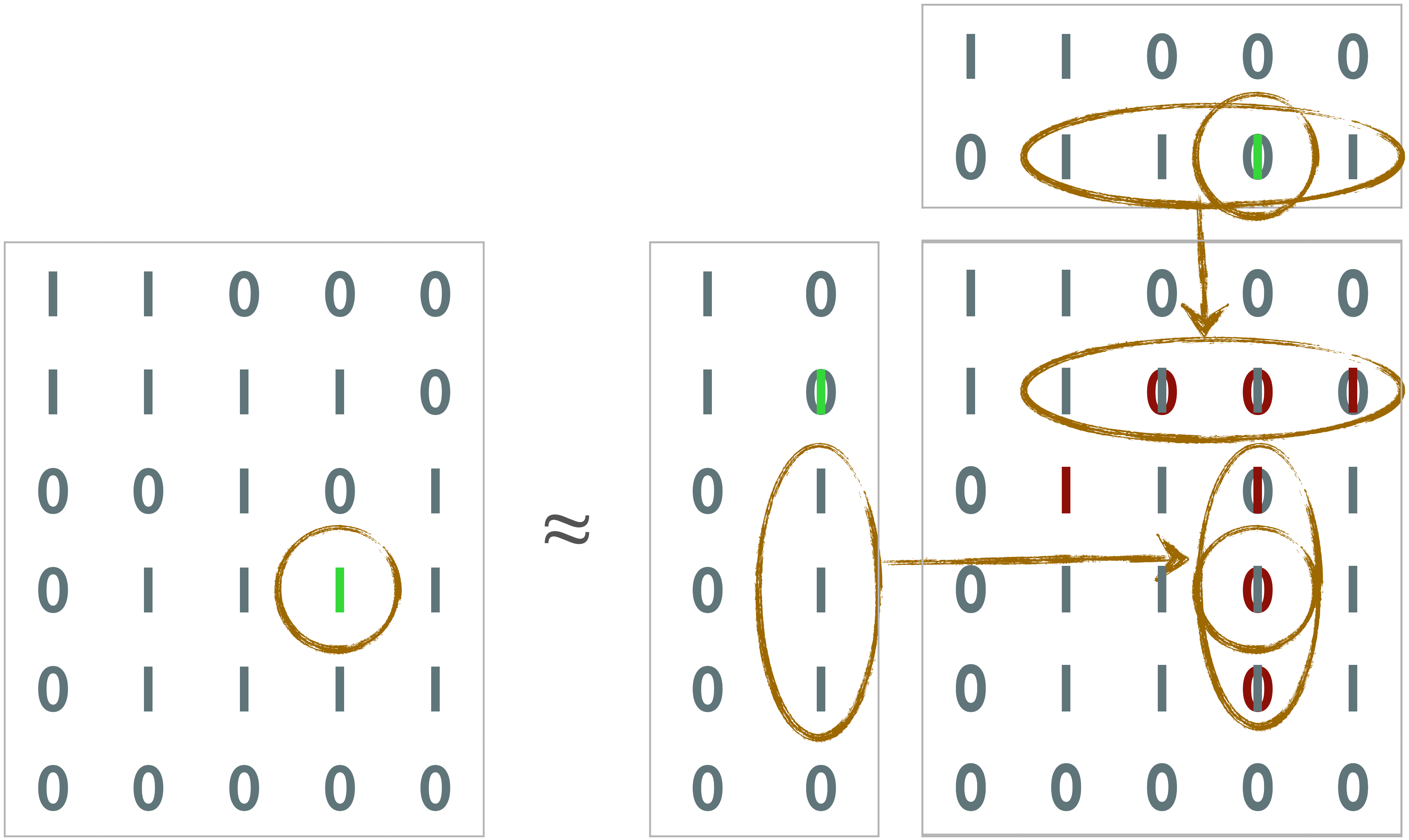
$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

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

Step 4: Add

$$\begin{matrix} & & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \approx & \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

Step 5: Add



Step 6: Remove

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

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

One Factor Too Many?

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

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

Adjusting the rank

- Use the MDL principle: Best rank is the one that lets us encode the data with least number of bits
  - Encode the data matrix using the factors and the residual (error) matrix
- Remove a factor if doing so reduces the overall encoding length
- Adding a factor is harder: need to have a new candidate factor to add

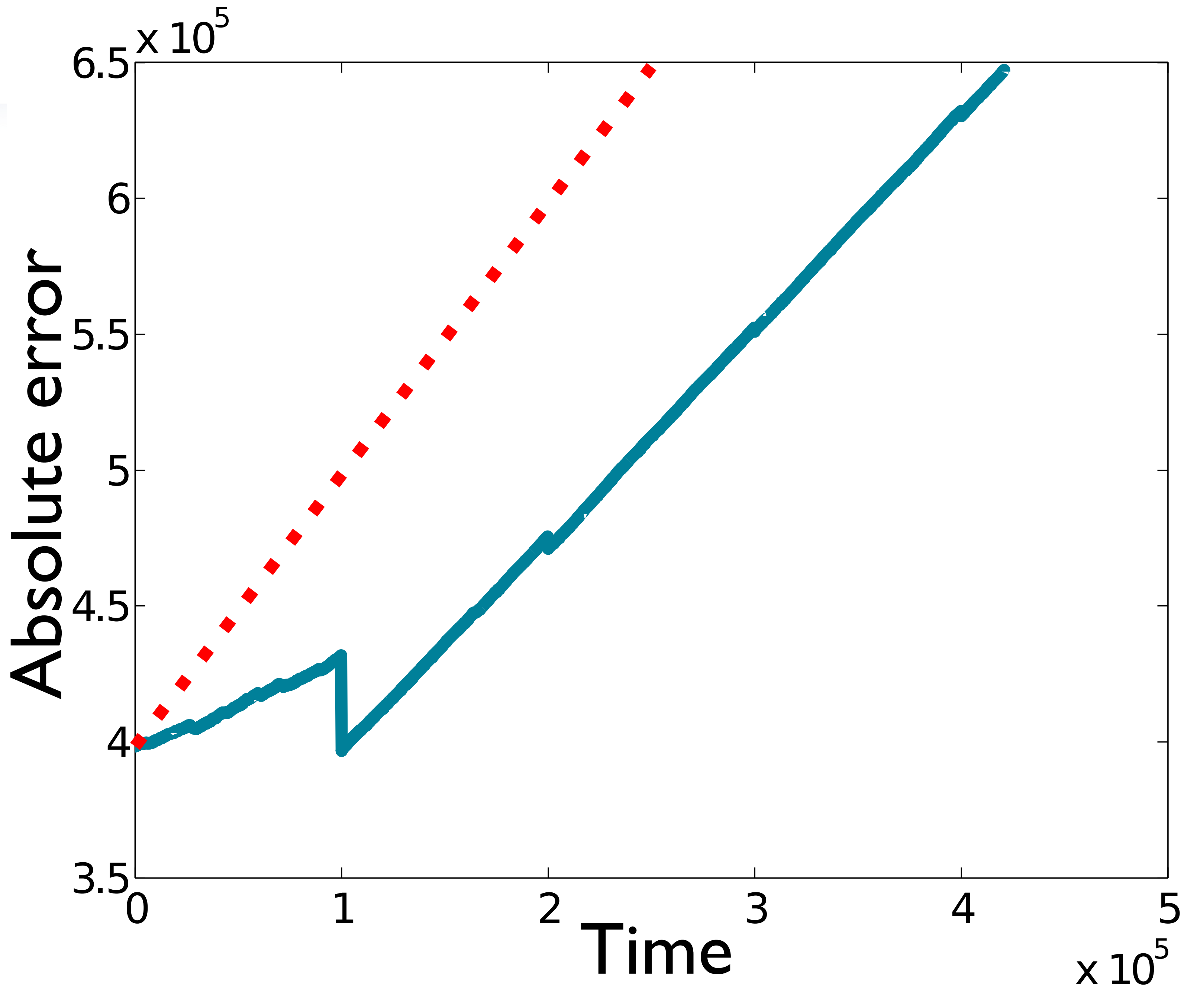
Adding a new factor

- Checking if we should remove a factor is easy
- But how to decide should we add a factor?
  - We need to decide what kind of a factor to add
- Simple heuristic: build candidates based on not-yet covered 1s and select the one with largest area

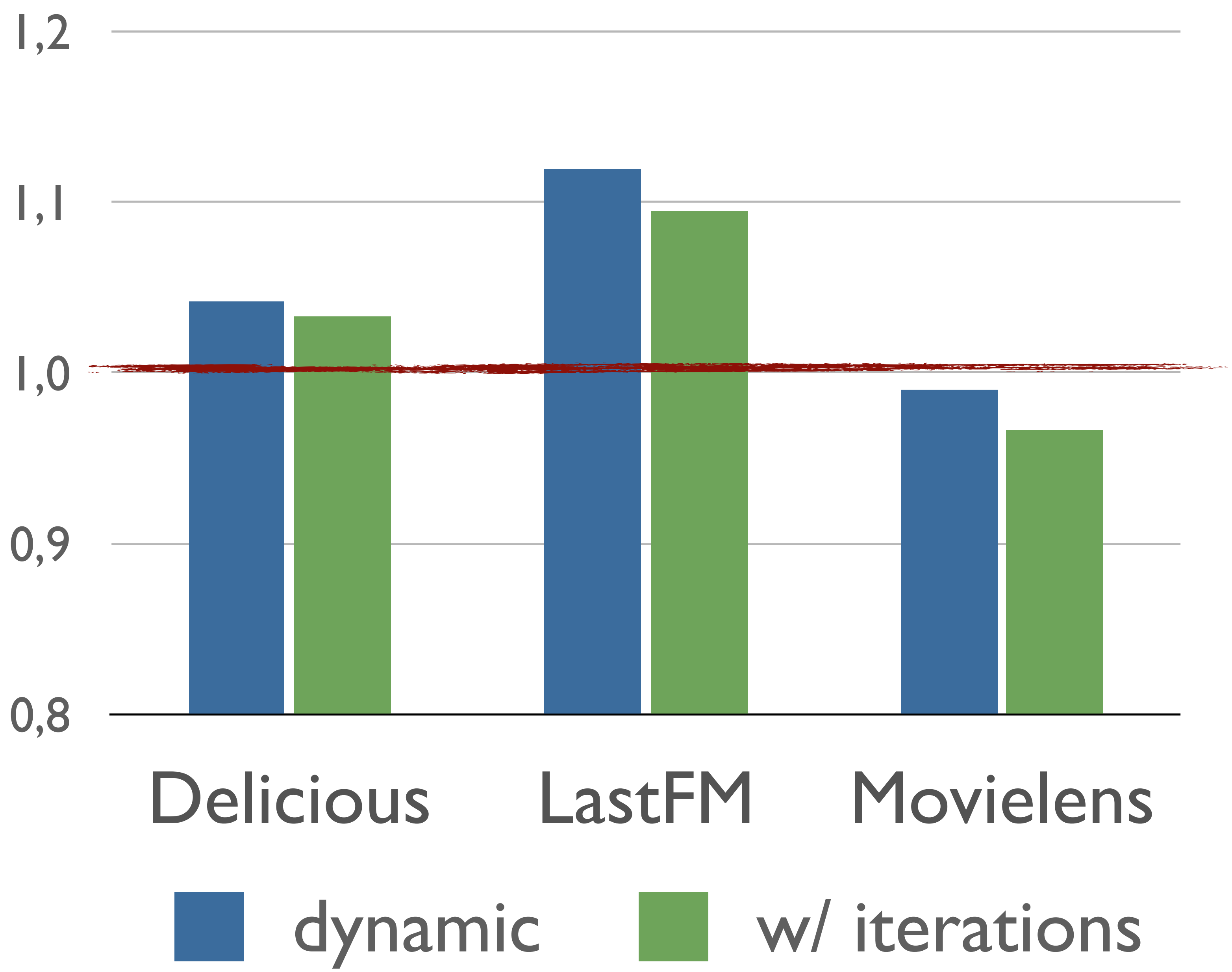
Making global updates

- The basic algorithm makes only somewhat local updates
- Fro global updates, we iteratively update ***B*** and ***C***
  - Fix ***B***, update ***C***; fix ***C***, update ***B***; etc.
  - The problem is (still) NP-hard – we use a heuristic
  - Computationally expensive

Error Over Time



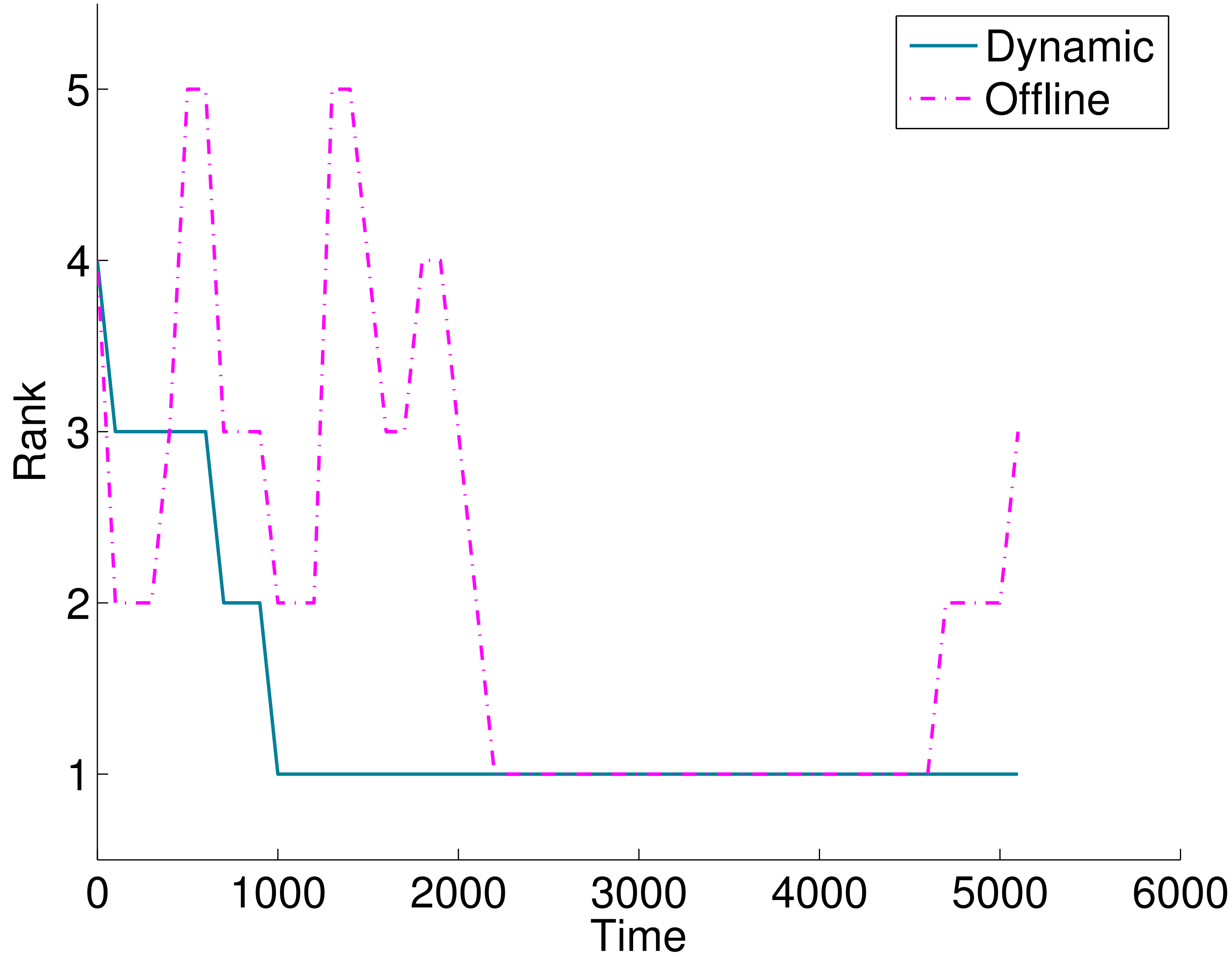
Empirical Competitiviness



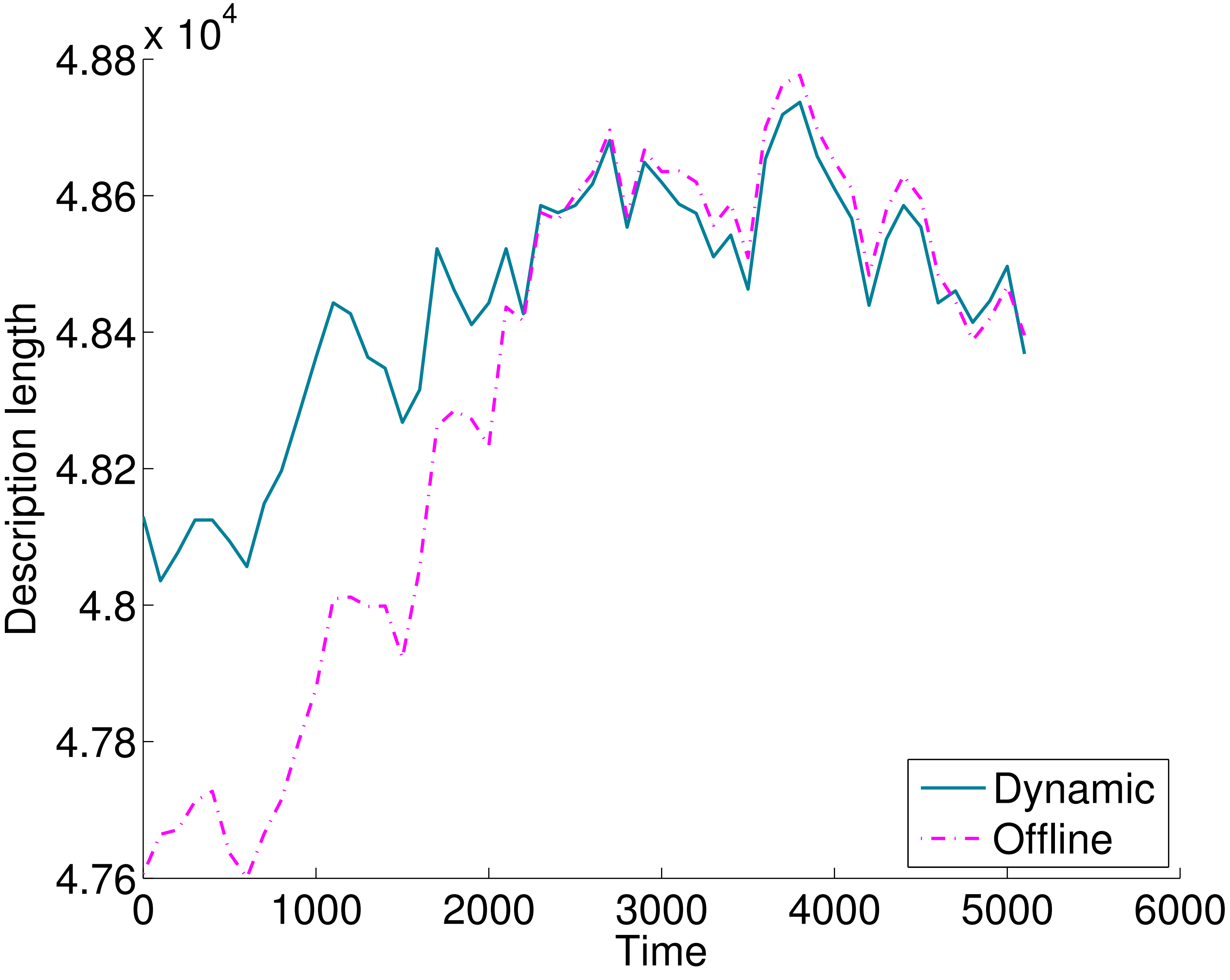
Running Times

	Delicious	LastFM	Movielens
Offline	43	200	4,21
Dynamic	4	213	4,452
w/ iterations	585	1,504	11,295

Rank Over Time



Description Length Over Time



Conclusions

- Not all data is available when you need it
  - On-line and dynamic methods try to adapt the results to the new data
- Not all data fits into memory
  - Streaming methods try to address that
- Doing data mining in dynamic or streaming environments is even harder than usual

Suggested Reading

- Rajaraman, A., Leskovec, J., & Ullman, J. D. (2013). Mining of Massive Datasets. Cambridge University Press.
  - Textbook, available on-line
- Guha, S., *et al*. (2000). Clustering data streams (pp. 359–366). In FOCS ’00.
- Sun, J., Tao, D., & Faloutsos, C. (2006). Beyond Streams and Graphs: Dynamic Tensor Analysis (pp. 374–383). In KDD ’06.