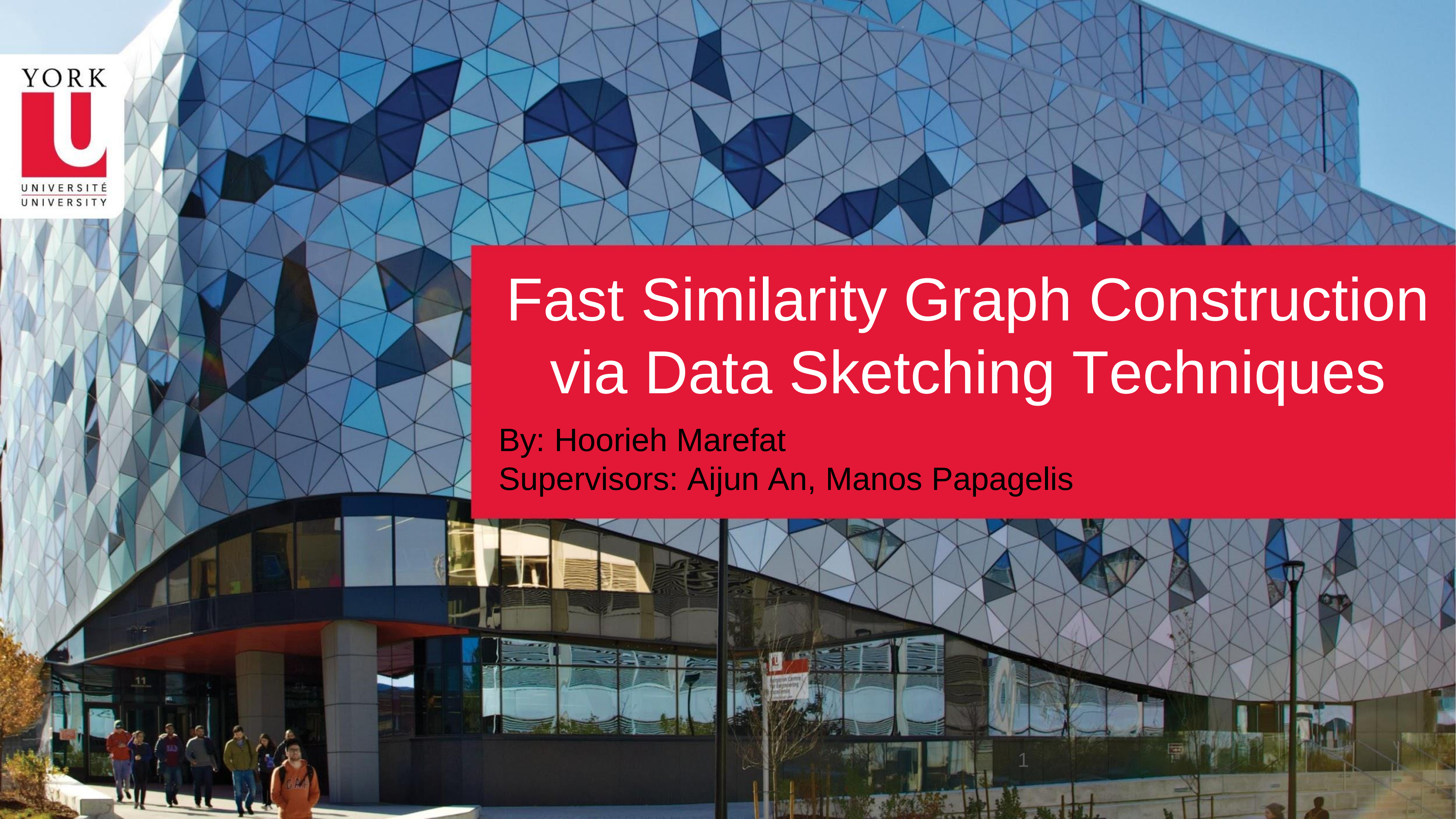


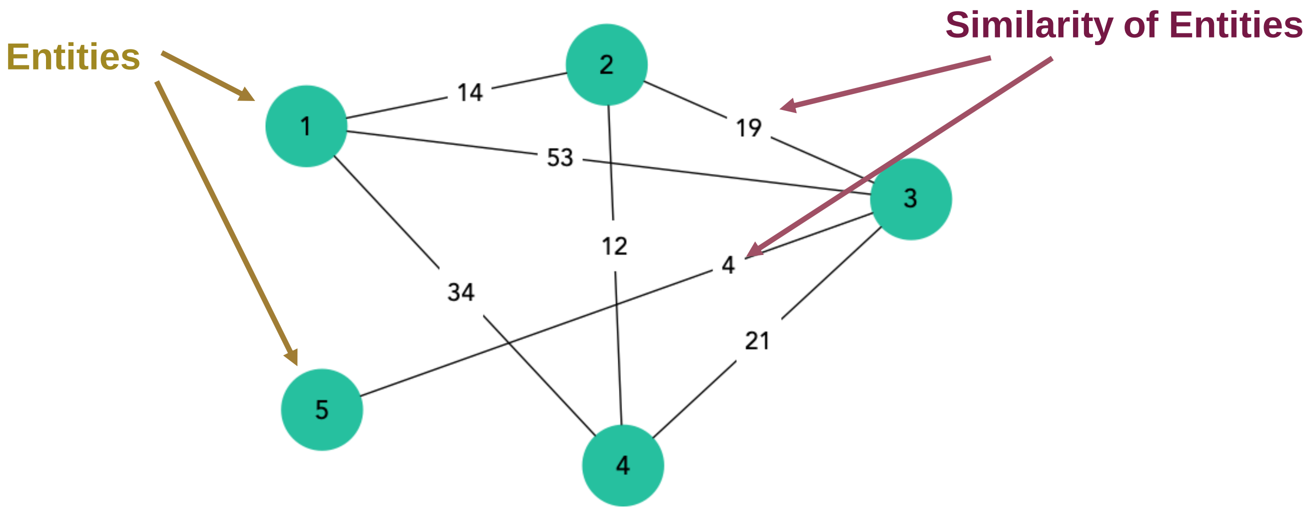
Fast Similarity Graph Construction via Data Sketching Techniques

By: Hoorieh Marefat
Supervisors: Aijun An, Manos Papagelis

Introduction

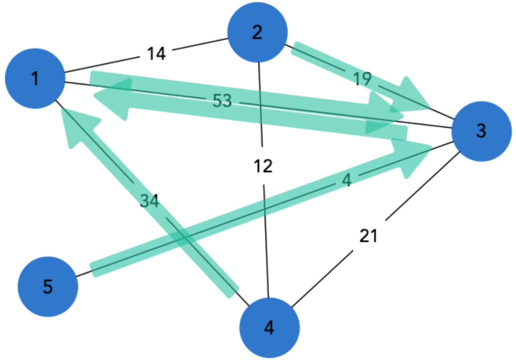
Motivation

We explore how to **build similarity graphs** in an **efficient** way

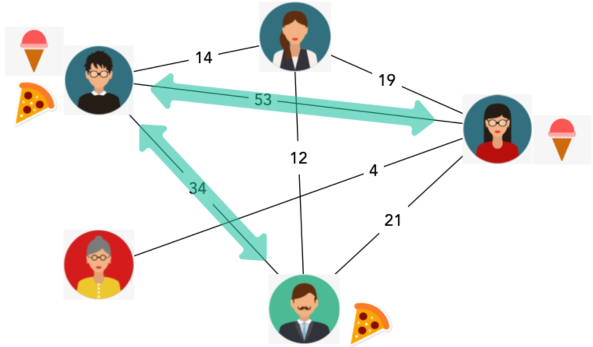


Applications

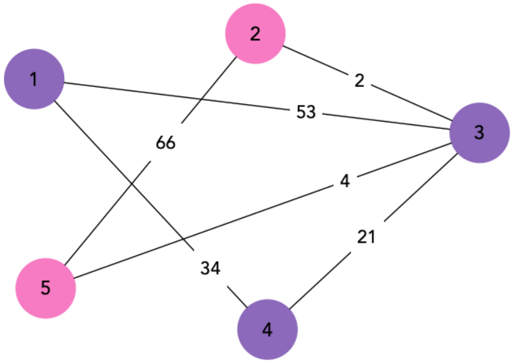
NN Search



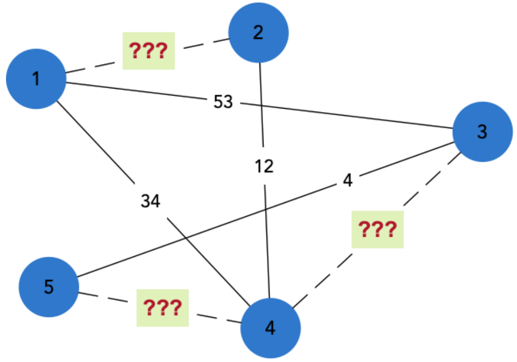
Collaborative Filtering



Clustering



Link Prediction



Similarity-based Graphs

- Similarity Graph
- ϵ-Graph
- Nearest Neighbour (NN) Graph

Similarity graph

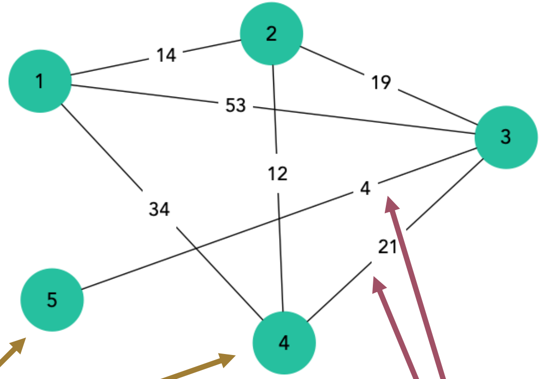
The Dataset

Attributes

Entities

UserID	Item1	Item2	Item3	Item4	Item5
1	3	0	2	0	10
2	4	5	1	0	0
3	1	3	0	1	5
4	0	2	2	0	3
5	0	0	0	4	0

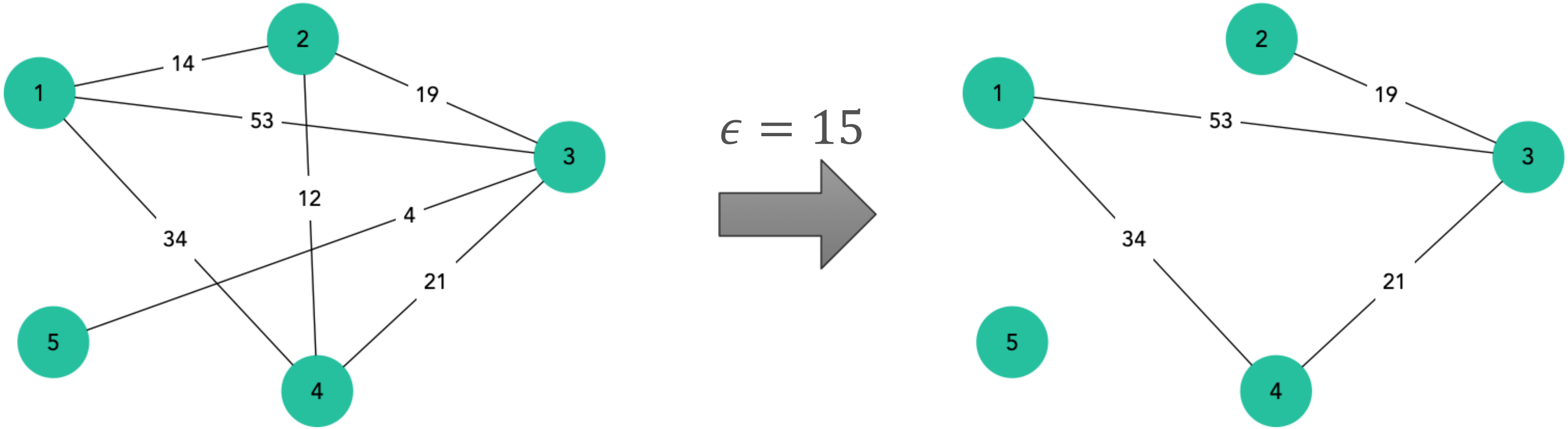
Dataset's Similarity Graph



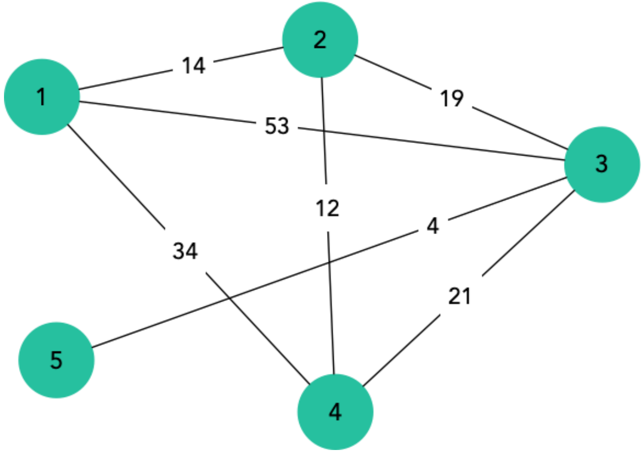
Entities

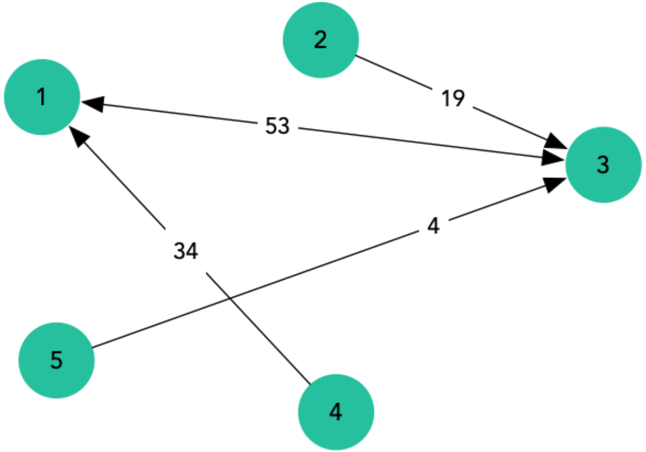
Similarity of Entities Based on Attributes

ϵ-graph



Nearest Neighbour (NN) Graph





If we have *k* nearest neighbours for each node,
graph would be a *k*NN graph

Similarity-based graphs

- Each of these graphs lead to **different problems**

- Each of them have **different solutions**

- In our work, we focus on the **ϵ-similarity graph**, a similarity graph whose edges are above the ϵ threshold

Similarity Graph Construction Challenges

When we have to build similarity graphs many times like in data streams for different snapshots or windows

$$O(n^2 d)$$

Scalability

D Attributes

N Entities

UserID	Item1	Item2	Item3	Item4	Item5
1	3	0	2	0	10
2	4	5	1	0	0
3	1	3	0	1	5
4	0	2	2	0	3
5	0	0	0	4	0

Similarities should be computed for all pairs of entities based on all attributes

Main Objective

Proposing an **efficient** and **effective** method

for **similarity graph construction** from **high-dimensional** data

Approaches

- Distributed solutions on MapReduce

- GPU-based solutions

- Efficient algorithmic optimizations
  - Using inverted index
  - Sampling/sketching based methods

Our work

Inverted index

Doc1: New Home Sales

Doc2: Home Sales In July

The Forward Index

Doc1 →	New	Home	Sales	
Doc2 →	Home	Sales	In	July

The Inverted Index

New →	Doc1	
Home →	Doc1	Doc2
Sales →	Doc1	Doc2
In →	Doc2	
July →	Doc2	

Data Sketching

Summarizing data that might be thought of as a **high dimensional** vector, or matrix

Data sketches have mathematically **proven error bounds**

Problem Statement

Sparse Vector Representation of Matrix

The Matrix

	A1	A2	A3	A4	A5
R1	5	1	0	0	4
R2	0	0	3	0	0
R3	0	0	2	0	1
R4	3	2	0	0	4
R5	0	3	0	0	2

Its Sparse Vector Representation

R1 = {(A1, 5), (A2, 1), (A5, 4)}

R2 = {(A3, 3)}

R3 = {(A3, 2), (A5, 1)}

R4 = {(A1, 3), (A2, 2), (A5, 4)}

R5 = {(A2, 3), (A5, 2)}

Approximate Similarity Graph Construction

- Given:
  - a **similarity threshold** ϵ
  - a **data matrix** or its **sparse vector representation**

- the problem is:
  - to build a **similarity graph** $G(V, E)$ where
    - V is the set of entities in the data matrix and
    - E is the set of edges representing the similarity between two nodes and
  - the **similarity is above the ϵ threshold**

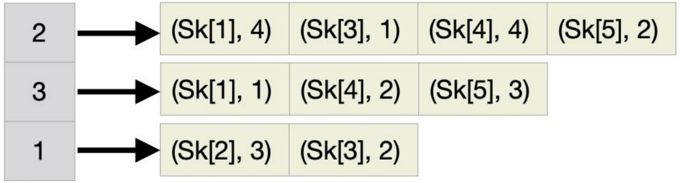
Proposed Methodology

Overview of the Algorithm

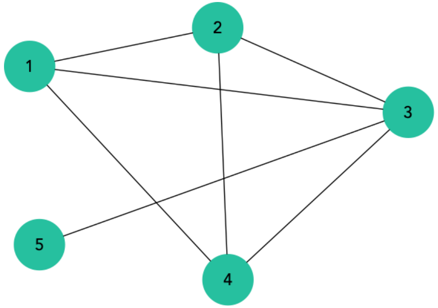
Step 1:
Data Sketching

Sk[1] = {(2, 4), (3, 1)}
Sk[2] = {(1, 3)}
Sk[3] = {(1, 2), (2, 1)}
Sk[4] = {(2, 4), (3, 2)}
Sk[5] = {(2, 2), (3, 3)}

Step 2:
Pairwise Similarity Computations



Step 3:
Similarity Graph Construction

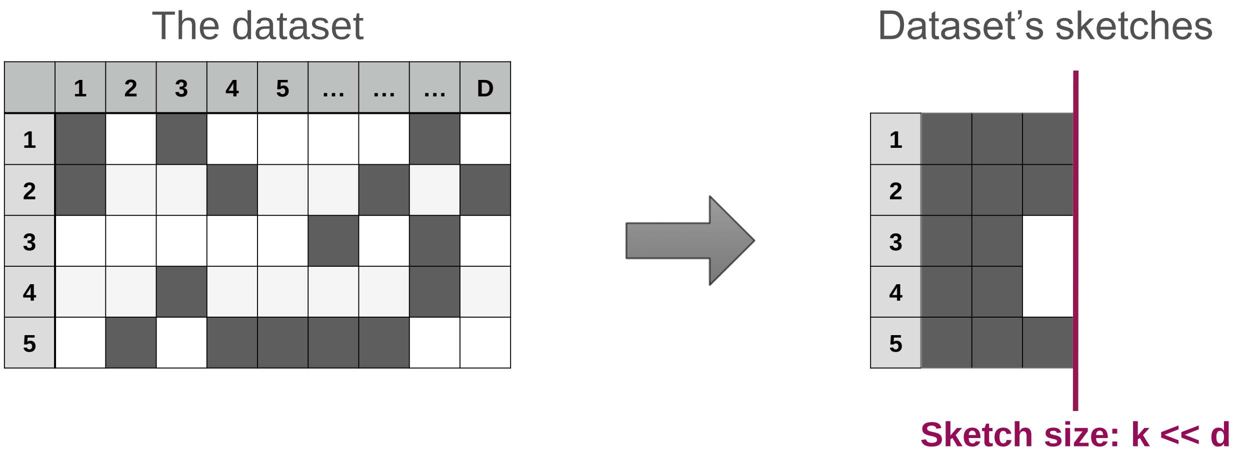


Step 1: Data Sketching

Main goal:

Start from a large dataset

Make it smaller



Step 1: Data Sketching

- Input
    - Two different kinds of dataset
        - A data matrix
        - Sparse vector representation of a data matrix
    - The sketch size (k)
- Output
    - Sketches of data

Data Sketching From a Data Matrix

Dataset: D

	1	2	3	4	5
1	5	1	0	0	4
2	0	0	3	0	0
3	0	0	2	0	1
4	3	2	0	0	4
5	0	3	0	0	2

Random
Column Permutation

[1, 2, 3, 4, 5]
To
[4, 3, 1, 5, 2]

Permuted Dataset

	1	2	3	4	5
1	0	4	1	5	0
2	3	0	0	0	0
3	2	1	0	0	0
4	0	4	2	3	0
5	0	2	3	0	0

Taking the first k column ID/value pairs with nonzero values

Sketches of D
Sketch size (K) = 2

Sk[1] = {(2, 4), (3, 1)}
Sk[2] = {(1, 3)}
Sk[3] = {(1, 2), (2, 1)}
Sk[4] = {(2, 4), (3, 2)}
Sk[5] = {(2, 2), (3, 3)}

Recording the highest column ID in each sketch

Sk_max_id[1] = 3
Sk_max_id[2] = 1
Sk_max_id[3] = 2
Sk_max_id[4] = 3
Sk_max_id[5] = 3

Sketch Max Column IDs

Data Sketching From Sparse Vector Representation

Original Dataset

R1 = {(1, 5), (2, 1), (5, 4)}
R2 = {(3, 3)}
R3 = {(3, 2), (5, 1)}
R4 = {(1, 3), (2, 2), (5, 4)}
R5 = {(2, 3), (5, 2)}

Column Permutation

[1, 2, 3, 4, 5]
To
[4, 3, 1, 5, 2]

Permuted Dataset

R1 = {(4, 5), (3, 1), (2, 4)}
R2 = {(1, 3)}
R3 = {(1, 2), (2, 1)}
R4 = {(4, 3), (3, 2), (2, 4)}
R5 = {(3, 3), (2, 2)}

Sorting

R1 = {(2, 4), (3, 1), (4, 5)}
R2 = {(1, 3)}
R3 = {(1, 2), (2, 1)}
R4 = {(2, 4), (3, 2), (4, 3)}
R5 = {(2, 2), (3, 3)}

Sketch size (K) = 2

Sketch Max IDs

Sk_max_id[1] = 3
Sk_max_id[2] = 1
Sk_max_id[3] = 2
Sk_max_id[4] = 3
Sk_max_id[5] = 3

Sketches

Sk[1] = {(2, 4), (3, 1)}
Sk[2] = {(1, 3)}
Sk[3] = {(1, 2), (2, 1)}
Sk[4] = {(2, 4), (3, 2)}
Sk[5] = {(2, 2), (3, 3)}

Data Sketching - Time Complexity

dimensionality

\# entities

average position of k^{th} nonzero entry

- From Data Matrix: $O(d) + O(nm)$
- From Sparse Vector Rep.: $O(d) + O(n \times (f + f.log(f) + k))$
- $O(n \times (f + f.log(f) + k)) < O(nm)$

average number of nonzero entries

sketch size

Step 2: Pairwise similarity computations

Input: Data sketches

Output: Similarities of all pairs of sketches

Our similarity measure is the **inner-product**

Structure of An Inverted Index

Sketches of D (k = 2)

Sk[1] = {(2, 4), (3, 1)}
Sk[2] = {(1, 3)}
Sk[3] = {(1, 2), (2, 1)}
Sk[4] = {(2, 4), (3, 2)}
Sk[5] = {(2, 2), (3, 3)}

The Inverted Index

2 → (Sk[1], 4) (Sk[3], 1) (Sk[4], 4) (Sk[5], 2)
3 → (Sk[1], 1) (Sk[4], 2) (Sk[5], 3)
1 → (Sk[2], 3) (Sk[3], 2)

Pairwise similarity computations

Sketches

Sk[1] = {(2, 4), (3, 1)}
Sk[2] = {(1, 3)}
Sk[3] = {(1, 2), (2, 1)}
Sk[4] = {(2, 4), (3, 2)}
Sk[5] = {(2, 2), (3, 3)}

Current Sketch Iterating Over

Inverted Index

2	→	(Sk[1], 4)	(Sk[3], 1)
3	→	(Sk[1], 1)	
1	→	(Sk[2], 3)	(Sk[3], 2)

(Sk[1], 16), (Sk[3], 4)

(Sk[1], 16+2)

Similar Sketches of Sk[4]

(Sk[1], 18), (Sk[3], 4)

Effective sample size (d_s) Computation

Our similarity measure is the **inner-product**

Dimensionality

Rows of original dataset

Original data $$a = \sum_{i=1}^{d} u_{1,i} u_{2,i}$$

Sketches

Sketches $$\hat{a} = \frac{d}{d_s} \sum_{i=1}^{d_s} \tilde{u}_{1,i} \tilde{u}_{2,i}$$

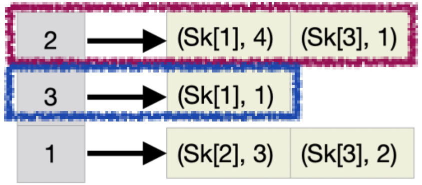
Computed pairwise

$$\boxed{d_s = min(max(ID(\tilde{u}_1)),\ max(ID(\tilde{u}_2)))}$$

Three approaches for d_s computation

Online

Sk[4] = {(2, 4), (3, 2)}



Offline via Sorting

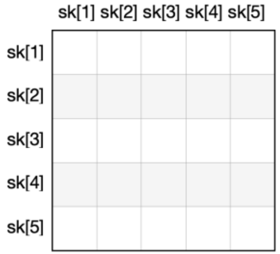
Sorting sketches based on their max ID

Sk[2] = {(1, 3)}
Sk[3] = {(1, 2), (2, 1)}
Sk[1] = {(2, 4), (3, 1)}
Sk[4] = {(2, 4), (3, 2)}
Sk[5] = {(2, 2), (3, 3)}

Ascending

Offline via Matrix Precomputation

d_s pairwise matrix



Preprocessing

Extra Space Needed for d_s computations

- Online: $O(1)$
- Offline via sorting: $O(n)$
- Offline via matrix precomputations: $O(n^2)$

Pairwise Similarity Computation - Time Complexity

entities

sketch size

average size of lists
in the inverted index

- Online: $O(\boxed{c} \times n \times k \times l)$

operations
in the inner-most loop

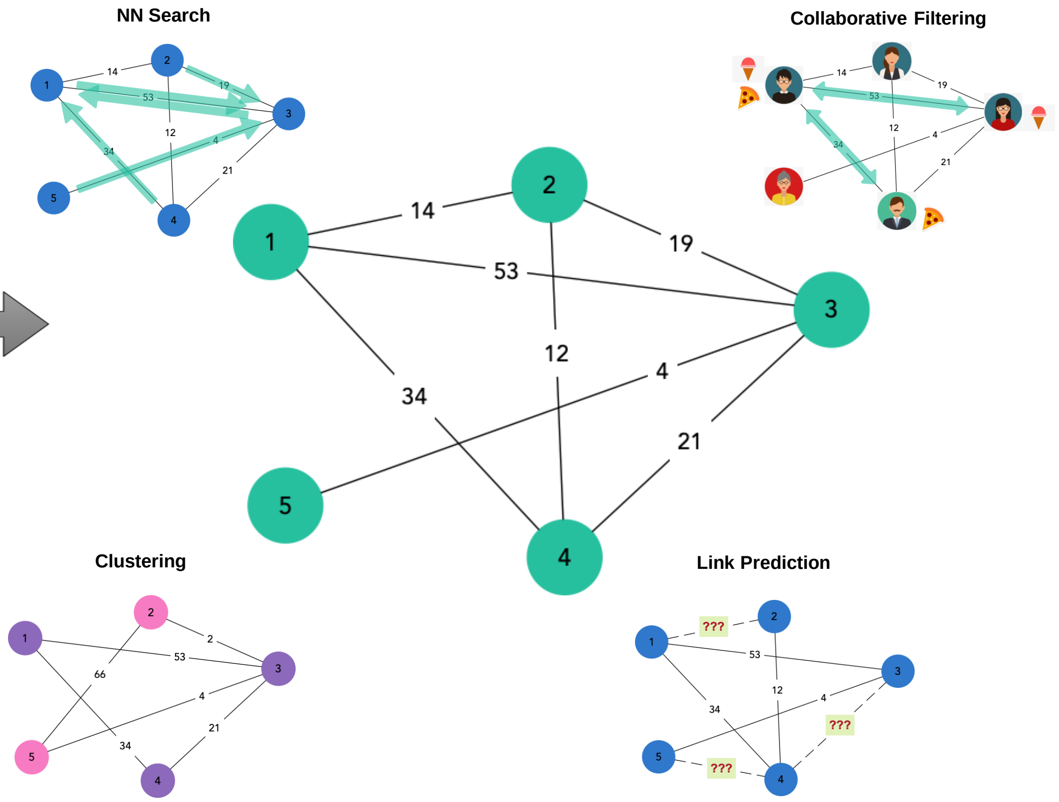
- Offline via sorting: $O(c' \times n \times k \times l) + O(n.log(n))$

- Offline via matrix precomputation: $O(c'' \times n \times k \times l) + O(n^2)$

$$c', c'' < c$$

Step 3: Similarity Graph Construction

Similarities of all pairs of entities



Evaluation

Evaluation Scenarios

- Runtime Cost
- Accuracy
- Effectiveness

Datasets

Name	Distribution	Density	Size
Normal2	Normal(μ=200, σ=50)	2%	10k × 10k
Normal4	Normal(μ=400, σ=100)	4%	
Normal6	Normal(μ=600, σ=100)	6%	
Normal8	Normal(μ=800, σ=100)	8%	
Binomial2	Binomial(n=10k, p=0.02)	2%	10k × 10k
Binomial4	Binomial(n=10k, p=0.04)	4%	
Binomial6	Binomial(n=10k, p=0.06)	6%	
Binomial8	Binomial(n=10k, p=0.08)	8%	

Evaluated methods

- Original

UserID	Item1	Item2	Item3	Item4	Item5
1	3	0	2	0	10
2	4	5	1	0	0
3	1	3	0	1	5
4	0	2	2	0	3
5	0	0	0	4	0

sim. operations on the vectors of the original dataset

- Normal Random Projection (NRP)

$$A \in \mathbb{R}^{n \times D} \quad R \in \mathbb{R}^{D \times k}$$
$$AR \in \mathbb{R}^{n \times k}$$

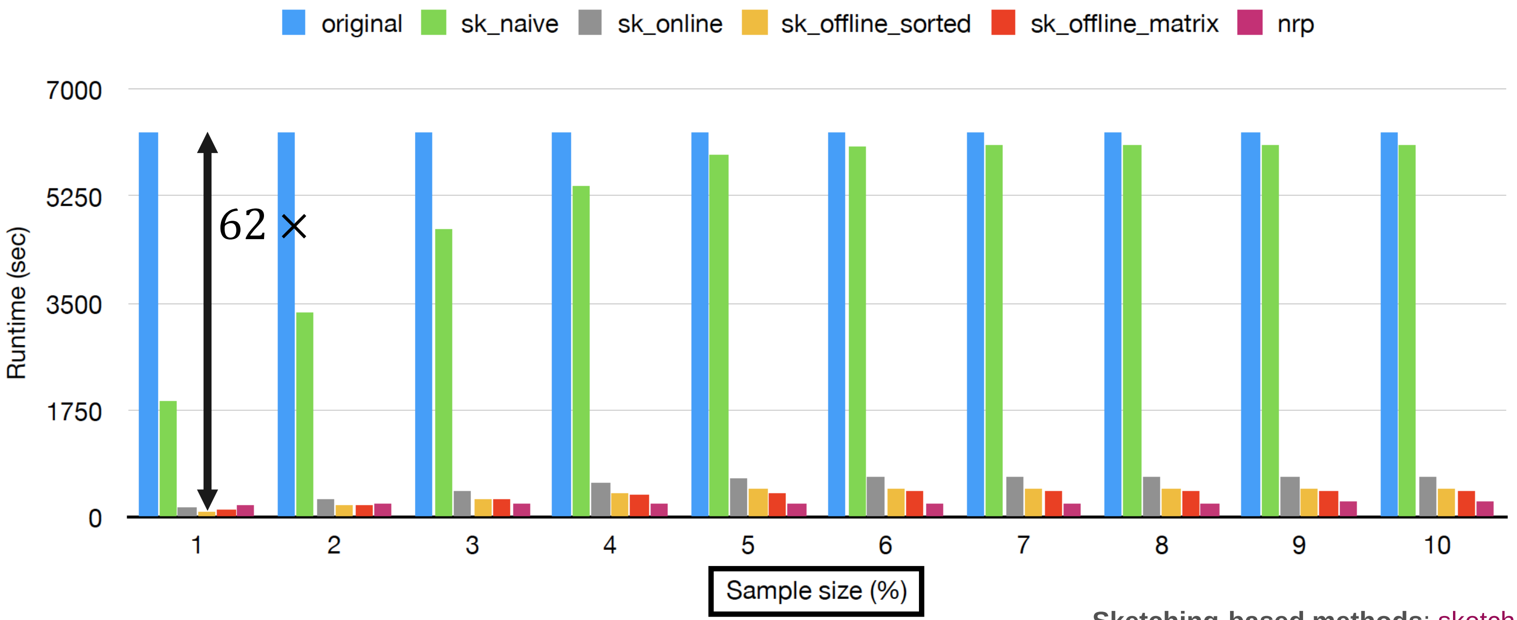
- Sk_naive

Data sketching without using an inverted index

- Sk_online
- Sk_offline_sorted
- Sk_offline_matrix

Our proposed methods

Runtime vs. sample size



(a) normal4 dataset

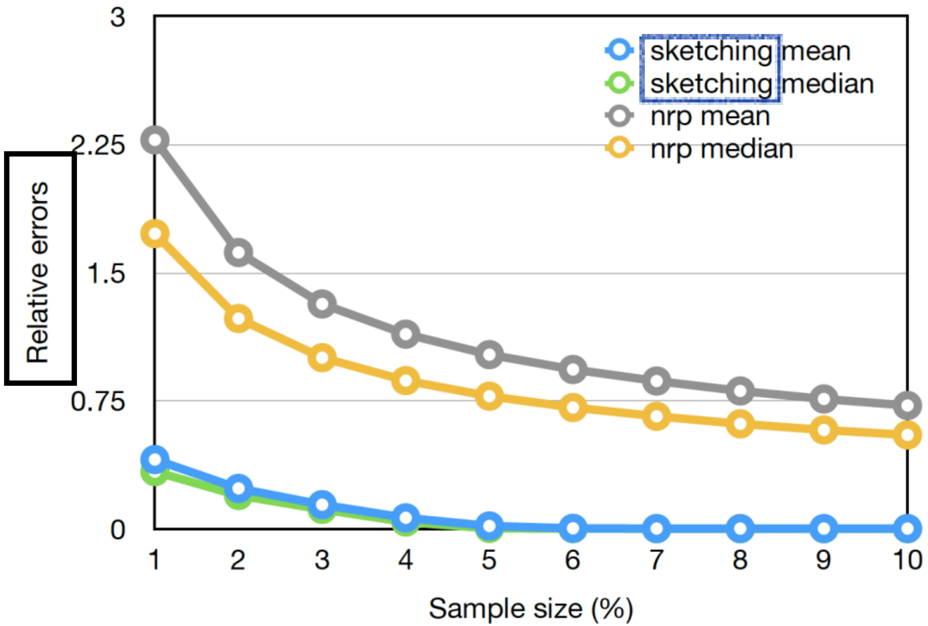
Sketching-based methods: sketch size (k)

NRP: # dimensions in the projected dataset

Accuracy vs. sample size

Relative error for each pair (i, j):

$$e_{ij} = \frac{|s_{ij}^{sample} - s_{ij}^{original}|}{s_{ij}^{original}}$$



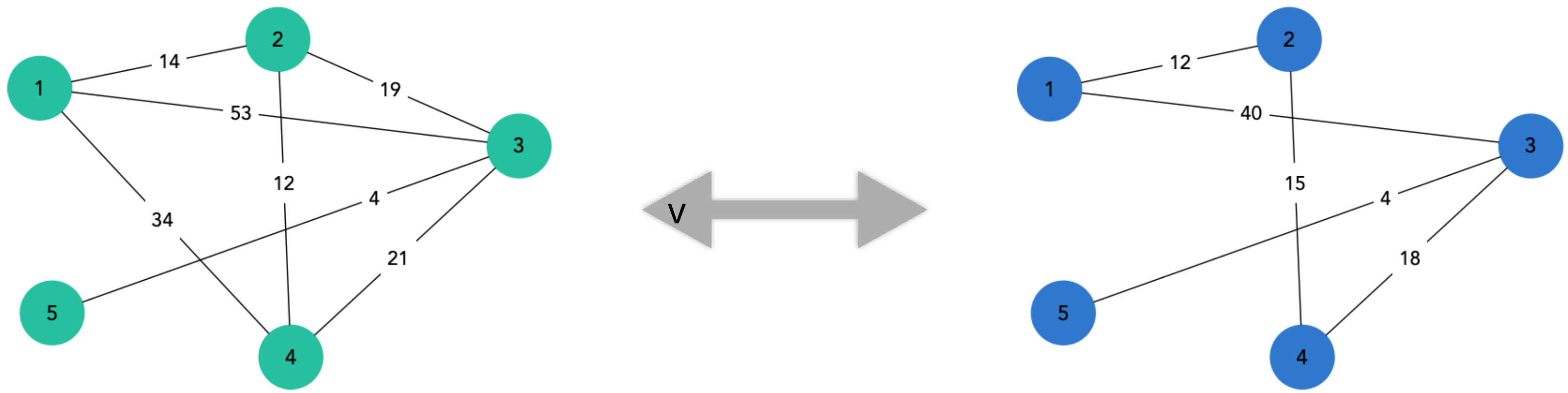
(a) normal4 dataset

The lower, the better

Effectiveness

Original Graph

Approximated Graph



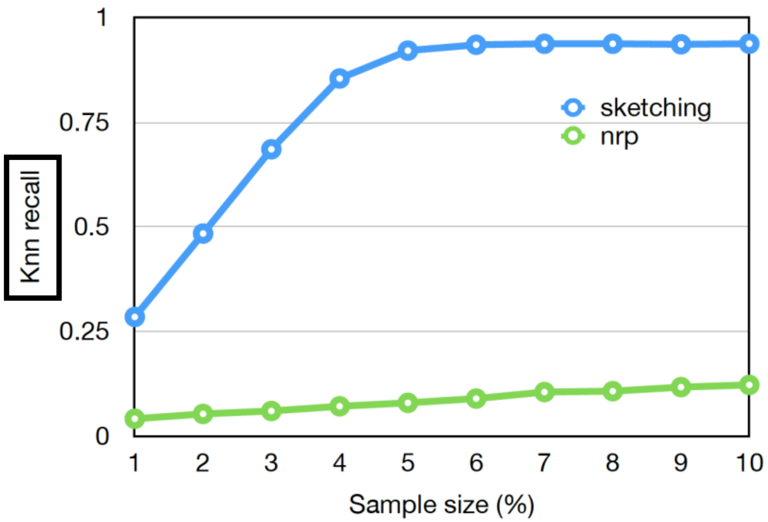
Effectiveness

- *K*-nearest neighbours

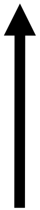
- Node centrality values
  - show ***importance*** of the nodes in the graph
  - we work with **eigenvector centrality**

- Node rankings
  - Based on the centrality of the nodes in the graph, we have a ranking for them
  - We use **Spearman's ranking correlation coefficient (ρ)**

Effectiveness - kNN



(a) normal4 dataset



The higher, the better

kNN recall:

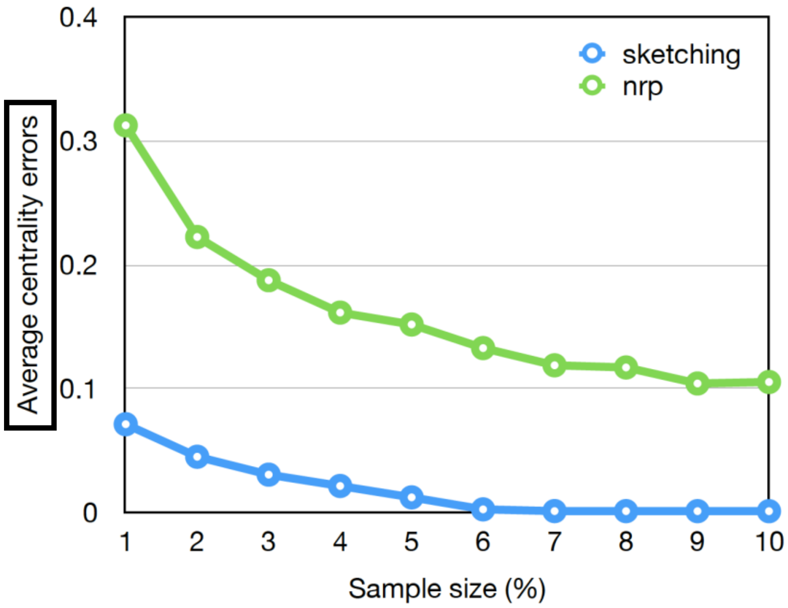
How many of the *k* real nearest neighbours of each node we are returning

Precision and recall are the same in this case

Effectiveness - Centrality Errors

Centrality error for node i:

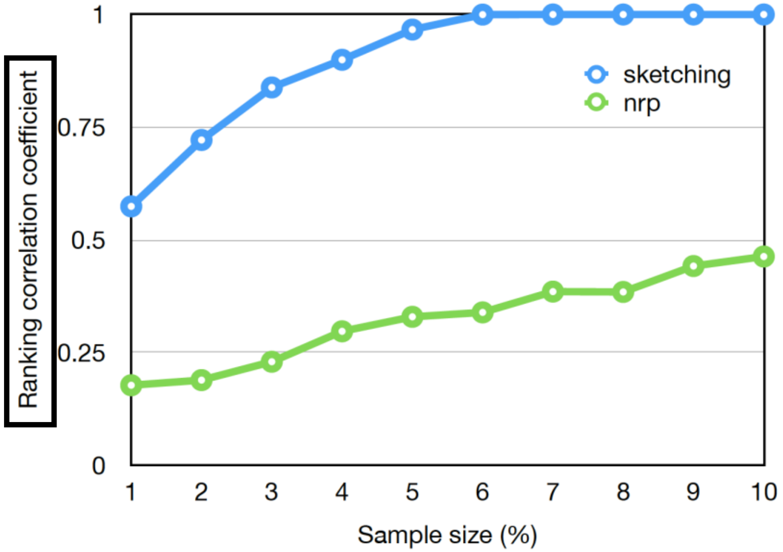
$$e_i = \frac{|c_i^{sample} - c_i^{original}|}{c_i^{original}}$$



(a) normal4 dataset

The lower, the better

Effectiveness - Node Ranking Correlations



(a) normal4 dataset

The higher, the better

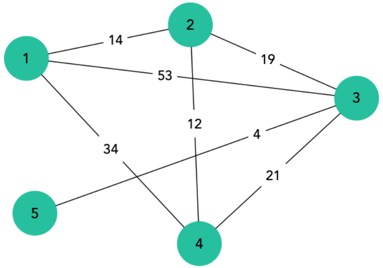
We use Spearman’s ranking correlation coefficient:

A measure to see how well node rankings of the approximated graphs are compared to the original.

Conclusion & Future Work

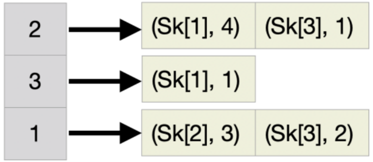
Summary of contributions

building similarity graphs
from high-dimensional data



$$O(n^2 d)$$

Data Sketching + **Inverted Index**



efficient, accurate and effective way of
construction of similarity graphs

Summary of contributions

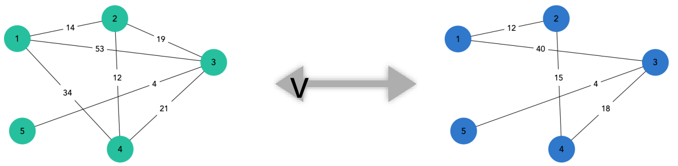
three algorithms each of which has a

trade-off for speed and space

Time efficient

Accurate

Effective on **different graph analysis tasks**



Limitations

Scalability for very large datasets

Storage overhead for the inverted index + maintenance cost

Future Work

Providing theoretical bounds
for the quality of graph downstream task results

Making the methods distributed
to increase scalability

Working with multi-dimensional arrays instead of matrix

Thank You!

Appendix

Online Pairwise Similarities

Algorithm 2: Online Pairwise Similarities

Input: sketches of D: **Sk**, sketch max ids: **Sk_Max_Id**, similarity threshold: ϵ, dimensionality of original data: d

Output: pairwise similarities: $\boldsymbol{S}$

$S \leftarrow \emptyset$

$I_1, I_2, ..., I_d \leftarrow \emptyset$ (I_i is the entry for the ith attribute in the inverted index. It will contain a list of (x, v)'s, where x is a row/sketch id and v is the value of x for the ith attribute.)

for $x \in Sk$ **do**

    $M \leftarrow \emptyset$ (M holds the similarity values between x and each of the skethes before x in Sk)

    for $(a, v) \in x$ **do**

        for $(y, y_v) \in I_a$ **do**

            $d_s = min(Sk\_Max\_Id[x],\ Sk\_Max\_Id[y])$

            if $a \leq d_s$ **then**

                $M[y] \leftarrow M[y] + (d/d_s) \cdot v \cdot y_v$

        $I_a \leftarrow I_a \cup \{(x, v)\}$

    $S_x \leftarrow$ **Filter_Similarities**(M, ϵ) (Remove similarities in M whose value is less than ϵ)

    $S \leftarrow S \cup (x, S_x)$

return $\boldsymbol{S}$

Offline Pairwise Similarities via Sorting

Algorithm 3: Offline Pairwise Similarities via Sorting

Input: sketches of D: **Sk**, sketch max ids: **Sk_Max_Id**, similarity threshold: ϵ, dimensionality of original data: d

Output: pairwise similarities: $\boldsymbol{S}$

$S \leftarrow \emptyset$

$I_1, I_2, ..., I_d \leftarrow \emptyset$

$Sorted\_Sk\_Indices \leftarrow$ **arg_sort**$(Sk\_Max\_Id, ascending)$

for $i \in range(|Sk|)$ **do**

  $x_{id} = Sorted\_Sk\_Indices[i]$

  $x \leftarrow Sk[x_{id}]$

  $M \leftarrow \emptyset$

  for $(a, v) \in x$ **do**

    for $(y, y_v) \in I_a$ **do**

      $d_s = Sk\_Max\_Id[y]$

      $M[y] \leftarrow M[y] + (d/ds) \cdot v \cdot y_v$

    $I_a \leftarrow I_a \cup \{(x, v)\}$

  $S_x \leftarrow$ **Filter_Similarities**(M, ϵ)

  $S \leftarrow S \cup (x, S_x)$

return $\boldsymbol{S}$

Offline Pairwise Similarities via Matrix Precomputations

Algorithm 4: Offline Pairwise Similarities via Matrix Precomputations

Input: sketches of D: **Sk**, sketch max ids: **Sk_Max_Id**, similarity threshold: ϵ, dimensionality of original data: d

Output: pairwise similarities: $\boldsymbol{S}$

$S \leftarrow \emptyset$

$I_1, I_2, ..., I_d \leftarrow \emptyset$

$n \leftarrow |Sk|$

for $i \in range(n)$ **do**

  for $j \in range(n)$ **do**

    $d_s\_pairwise[i][j] \leftarrow min(Sk\_Max\_Id[i], Sk\_Max\_Id[j])$

for $x \in Sk$ **do**

  $M \leftarrow \emptyset$

  for $(a, v) \in x$ **do**

    for $(y, y_v) \in I_a$ **do**

      $d_s = d_s\_pairwise[x][y]$

      if $a < d_s$ **then**

        $M[y] \leftarrow M[y] + (d/ds) \cdot v \cdot y_v$

    $I_a \leftarrow I_a \cup \{(x, v)\}$

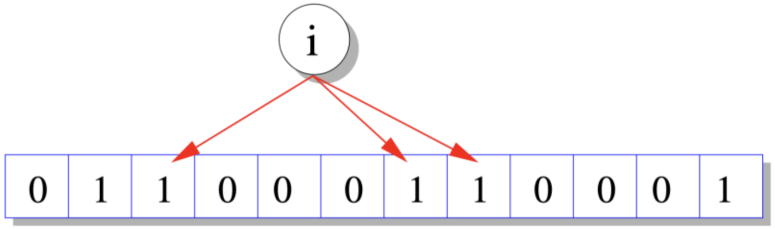
  $S_x \leftarrow$ **Filter_Similarities**(M, ϵ)

  $S \leftarrow S \cup (x, S_x)$

return $\boldsymbol{S}$

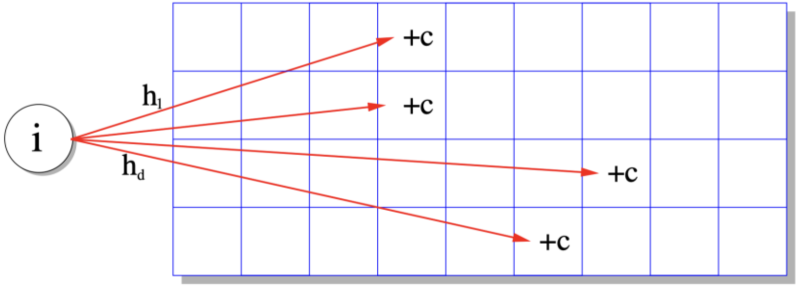
Bloom filters for set summarization

- Set membership
- The item has *definitely not been stored*, or the item has *probably been stored*
- Having *k* hash functions and map each item with each of them
- Set all the corresponding bits to 1. If all were one for an item, say it is a member, if any of them were 0, say it is not a member



Counting with count-min sketch

- Counts the number of items of a certain type
- Sketch: an array of counters, and a set of hash functions which map items into the array
- Count of the desired item is to take the smallest of counters in each row as our estimate.



PCA vs. RP

- PCA
  - Extracting a small number of directions from the data which captures most of variation of dataset
  - Finding the direction requires finding eigenvectors of the covariance matrix

Substantial amount of work

- Random projection
  - Rather than finding “the best” directions, it suffices to use random vectors.
  - Picking a moderate number of random directions captures a *comparable amount of variation*, while requiring *much less computation*.