An Efficient Similarity Join Algorithm with Cosine Similarity Predicates

Dongjoo Lee, Jaehui Park, Sang-goo Lee

Seoul National University, Korea

Junho Shim

Sookmyung Women's University, Korea

Agenda

- Introduction
- Previous Filtering Techniques
- **MMJoin**
- Experimental evaluation
- Extension to Dice and Tanimoto Similarity Joins
- Conclusion & Future Work

Similarity Join

- Similarity Join
  - Finds all similar pairs of objects
  - $\text{simjoin}(\mathcal{D}, \text{sim}(), t) = \{(x, y, \text{sim}(x, y) | x, y \in \mathcal{D} \wedge \text{sim}(x, y) \geq t\}$
    - $\text{sim}()$: similarity function
    - t : similarity threshold

- Similarity functions
  - Jaccard coefficient, Dice coefficient
  - Cosine similarity, Tanimoto coefficient
  - Edit distance (similarity)

Application Area

- Mining
  - Clustering
  - Bio informatics

- Near Duplicate Detection (NDD)
- Data integration & cleansing

Won Kim Data Mining: Promises, Reality, and Future.
Won Kim Data Mining: Promises, Reality, and Future - Part 2.
Won Kim Data Mining: Promises, Reality, and Future - Part 3.

Bon Jovi, Livin' on a player
Bon Jovi, Living on a player

Two Issues

- How to measure similarity between objects
  - Various similarity functions

- **Computation time**
  - $O(n^2)$, n is the total number of objects
  - Efficient algorithms

Motivation

- There are many application domains in which finding similar object pairs is fundamental operation.
- Considering importance of features of each object is also important.
- Weight vector is a good representation for various objects to consider importance of features, but **there is no efficient similarity join algorithm over weight vectors**.
  - ppjoin+ for sets (binary vectors)
  - EdJoin for sequences
  - All-pairs for weight vectors but not efficient

Contribution

- We propose new similarity upper bounds that is computable with little overhead.
- We propose an efficient algorithm for cosine similarity joins by exploiting the proposed similarity upper bounds.
- We extended the algorithm to Dice and Tanimito similarity joins.
- We show that our algorithm outperforms a state-of-the-art algorithm using empirical evaluation with large scale datasets.

Problem Statement

- **Self**-similarity join.
- Similarity between objects is quantified by **cosine similarity.**
- Each object is represented as a vector.
  - Feature (token) universe
    - $\mathcal{U} = \{t_1, t_2, \ldots, t_i, \ldots, t_m\}$
  - Weight Vector
    - $\vec{x} = \langle x[1], x[2], \ldots, x[i], \ldots, x[m] \rangle$
    - $\vec{y} = \langle y[1], y[2], \ldots, y[i], \ldots, y[m] \rangle$
    - $x[i]$ and $y[i]$ are weights for the ith token of $\vec{x}$ and $\vec{y}$

Cosine Similarity Join

- Cosine Similarity between Vectors

$$C(x, y) = \frac{\sum_{i=1}^{m} x[i] \cdot y[i]}{\sqrt{\sum_{i=1}^{m} x[i]^2}\sqrt{\sum_{i=1}^{m} y[i]^2}} = \frac{\mathrm{dot}(x, y)}{\|x\|\|y\|}$$

- Vector normalization
  - Original vector: $x_0 = \langle x_0[1], x_0[2], \dots, x_0[i], \dots, x_0[m] \rangle$
  - Normalized vector: x, $x[i] = \frac{x_0[i]}{\|x_0\|}$

- For normalized vectors, $C(x, y) = \mathrm{dot}(x, y)$

$$\mathrm{dot}(x, y) = \sum_{i=1}^{m} x[i] \cdot y[i]$$

Calculating Dot-Product

We want to find pairs whose cosine similarity is above 0.9 from a dataset.

$\text{simjoin}(\mathcal{D}, \text{C}(), 0.9)$

Input

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
o1	0.46	0.31			0.31			0.31		0.62	0.31				0.15
o2	0.46	0.31		0.31	0.31			0.31		0.62					0.15
o3		0.33		0.17		0.33	0.33		0.33	0.50	0.50		0.17		
o4		0.55	0.18	0.18	0.37	0.18		0.37	0.18		0.37		0.18		0.37
o5	0.15		0.30		0.30			0.61				0.46		0.46	

Output

	o1	o2	o3	o4	o5
o1		0.91	0.57	0.57	0.35
o2			0.47	0.51	0.35
o3				0.55	0.00
o4					0.39
o5					

Calculating Dot-Product

- Naïve pair-wise computation.
  - NestedLoopJoin
- Using inverted index.
  - InvertedIndexJoin
  - Merits?
    - We can avoid unnecessary computation using filtering techniques.

NestedLoopJoin

Algorithm 1 `NestedLoopJoin`($\mathcal{D}$, t)

Input: a set of vectors $\mathcal{D}$, similarity threshold t

Output: $\{(x, y) | x, y \in \mathcal{D} \text{ and } sim(x, y) \geq t\}$

$O \leftarrow \phi$
for each $x \in \mathcal{D}$ **do**
    for each $y \in \mathcal{D}$ such that $x \succ y$ **do**
        if $sim(x, y) \geq t$ **then**
            $O \leftarrow O \cup \{(x, y)\}$
        end if
    end for
end for
return O

InvertedIndexJoin

Algorithm 2 InvertedIndexJoin($\mathcal{D}$, t)

Input: a set of vectors $\mathcal{D}$, similarity threshold t

Output: $\{(x, y) | x, y \in \mathcal{D}$ and $sim(x, y) \geq t\}$

$O \leftarrow \phi$

$I_1, \ldots, I_m \leftarrow \phi$

for each $x \in \mathcal{D}$ **do**

  $C \leftarrow$ empty map from id to weight

  for $i = 1$ **to** m such that $x[i] > 0$ **do**

    for each $(y, y[i]) \in I_i$ **do**

      $C[y] \leftarrow C[y] + x[i] \cdot y[i]$

    end for

    $I_i \leftarrow I_i \cup \{(x, x[i])\}$

  end for

  for each $y \in C$ **do**

    if $C[y] \geq t$ **then**

      $O \leftarrow O \cup \{(x, y)\}$

    end if

  end for

end for

return O

$$\mathrm{dot}(x, y) = \sum_{i=1}^{m} x[i] \cdot y[i]$$

InvertedIndexJoin

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
o1	0.46	0.31			0.31			0.31		0.62	0.31				0.15
o2	0.46	0.31		0.31	0.31			0.31		0.62					0.15
o3		0.33		0.17		0.33	0.33		0.33	0.50	0.50		0.17		
o4		0.55	0.18	0.18	0.37	0.18		0.37	0.18		0.37		0.18		0.37
o5	0.15		0.30		0.30			0.61				0.46		0.46	

	o1	o2	o3	o4	o5
o1		0.91	0.57		
o2			0.47		
o3					
o4					
o5					

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
o1 0.46	o1 0.31		o2 0.31	o1 0.31	o3 0.33	o3 0.33	o1 0.31	o3 0.33	o1 0.62	o1 0.31		o3 0.17		o1 0.15
o2 0.46	o2 0.31		o3 0.17	o2 0.31			o2 0.31		o2 0.62	o3 0.50				o2 0.15
	o3 0.33								o3 0.50					

Escaping Unnecessary Computation

Input

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
o1	0.46	0.31			0.31										
o2	0.46	0.31		0.31	0.31						?				
o3		0.33		0.17											
o4		0.55	0.18	0.18	0.37										
o5	0.15		0.30		0.30			0.61				0.46		0.46	

1 → p - - - Incremental computation - - - → m

Output

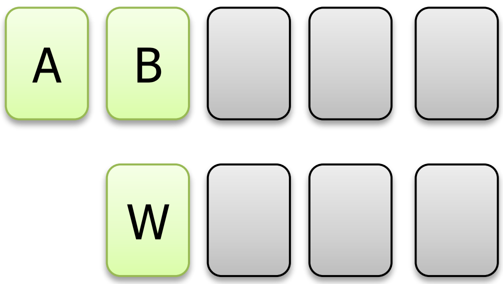
	o1	o2	o3	o4	o5
o1		0.91	0.57	0.57	0.16
o2			0.47	0.51	0.16
o3				0.55	0.00
o4					0.17
o5					

$$\mathrm{dot}(o_1, o_5) = 0.16 + 0.61 \cdot o_1[H] + 0.46 \cdot o_1[L] + 0.46 \cdot o_1[N] < t = 0.9$$
$$\mathrm{dot}(o_2, o_5) = 0.16 + 0.61 \cdot o_2[H] + 0.46 \cdot o_2[L] + 0.46 \cdot o_2[N] < t = 0.9$$
$$\mathrm{dot}(o_3, o_5) = 0.00 + 0.61 \cdot o_3[H] + 0.46 \cdot o_3[L] + 0.46 \cdot o_3[N] < t = 0.9$$
$$\mathrm{dot}(o_4, o_5) = 0.17 + 0.61 \cdot o_4[H] + 0.46 \cdot o_4[L] + 0.46 \cdot o_4[N] < t = 0.9$$

Can we know these? If we can, how?

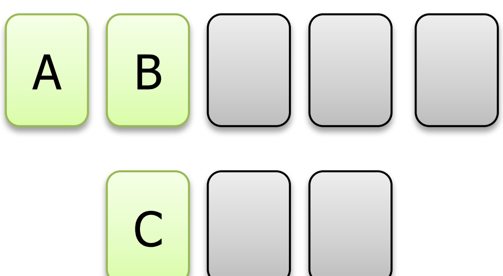
Card Example

- From Wei Wang. Efficient Exact Similarity Join Algorithms. Seminar at University of Technology, Sydney, Oct 2009.

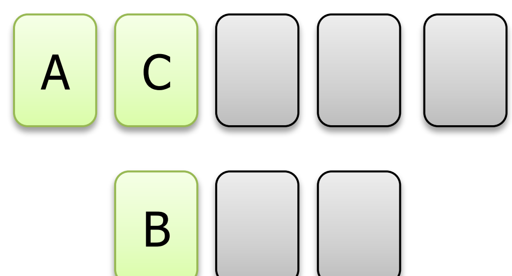


Possible maximum overlap is 3

- If we want to know whether the overlap between two cards lists is more than 4, we do not need to see more cards.
- Other examples



Possible maximum overlap is 3



Possible maximum overlap is 2

Preliminary: Prefix and Suffix

- $x = \langle x[1], \dots, x[p], x[p+1], \dots, x[m] \rangle$

  Prefix at p : $x'_p = \langle x[1], \dots, x[p], 0, \dots, 0 \rangle$

  Suffix at p : $x''_p = \langle 0, \dots, 0, x[p+1], \dots, x[m] \rangle$

- Properties

  $x = x'_p + x''_p$

  $|x| = |x'_p| + |x''_p|$

  $\|x\|^2 = \|x'_p\|^2 + \|x''_p\|^2$

  $\mathrm{dot}(x'_p, y) = \mathrm{dot}(x, y'_p) = \mathrm{dot}(x'_p, y'_p)$

  $\mathrm{dot}(x'_p, y'_q) = \mathrm{dot}(x'_{\min(p,q)}, y) = \mathrm{dot}(x, y'_{\min(p,q)})$

Filtering Principle

- We can avoid unnecessary similarity computation through calculating upper-bounds given **known information** with little overhead.

$$
\begin{aligned}
\mathrm{dot}(x, y) &= \sum_{i=1}^{m} x[i] \cdot y[i] \\
&= \sum_{i=1}^{p} x[i] \cdot y[i] + \sum_{i=p+1}^{m} x[i] \cdot y[i] \\
&= \mathrm{dot}\left(x'_p, y'_p\right) + \mathrm{dot}\left(x''_p, y''_p\right) \\
&\leq \mathrm{dot}\left(x'_p, y'_p\right) + \mathrm{ubdot}\left(x''_p, y''_p\right)
\end{aligned}
$$

$$
\boxed{
\begin{aligned}
dot\left(x'_p, y'_p\right) + \mathrm{ubdot}\left(x''_p, y''_p\right) < t &\Rightarrow \mathrm{dot}(x, y) < t \\
\mathrm{ubdot}(x, y) < t &\Rightarrow \mathrm{dot}(x, y) < t
\end{aligned}
}
$$

Filtering Statements

Upper-Bounds with Maximum Weight (1)

- Let $\mathrm{maxw}_i(\mathcal{D})$ be the maximum weight $y[i]$ for all y in $\mathcal{D}$
- $\mathcal{M}$ is a vector whose ith weight is $\mathrm{maxw}_i(\mathcal{D})$, then

$$\begin{aligned} \mathrm{dot}(x''_p, y) &= \sum_{i=p+1}^{m} x[i] \cdot y[i] \\ &\le \sum_{i=p+1}^{m} x[i] \cdot \mathrm{maxw}_i(\mathcal{D}) \\ &= \mathrm{dot}(x'', \mathcal{M}) \end{aligned}$$

Bayardo et al., WWW 2007

Upper-Bounds with Maximum Weight (2)

- Let $\mathrm{maxw}(x)$ be the maximum weight $x[i]$ for all i over 1…m.
- $\mathcal{M}(x)$ is a vector whose ith weight is $\min\left(\mathrm{maxw}(x), \mathrm{maxw}_i(\mathcal{D})\right)$
- If $\mathrm{maxw}(x) \geq \mathrm{maxw}(y)$, then

$$
\begin{aligned}
\mathrm{dot}\left(x''_p, y\right) &= \sum_{i=p+1}^{m} x[i] \cdot y[i] \\
&\leq \sum_{i=p+1}^{m} x[i] \cdot \min\left(\mathrm{maxw}(x), \mathrm{maxw}_i(\mathcal{D})\right) \\
&= \mathrm{dot}(x'', \mathcal{M}(x))
\end{aligned}
$$

Bayardo et al., WWW 2007

Upper-Bounds with Vector Size

$$\mathrm{dot}(x,y) = \sum_{i=1}^{p} x[i]\cdot y[i] + \sum_{i=p+1}^{m} x[i]\cdot y[i]$$

$$\le \mathrm{dot}(x'_p, y'_p) + \min(|x''_p|, |y''_p|)\cdot \mathrm{maxw}(x''_p)\cdot \mathrm{maxw}(y''_p)$$

$$\le \min(|x|,|y|)\cdot \mathrm{maxw}(x)\cdot \mathrm{maxw}(y)$$

Bayardo et al., WWW 2007

Upper-Bounds with Vector Length

$$\mathrm{dot}\left(x_p'', y_p''\right) = \sum_{i=p+1}^{m} x[i] \cdot y[i]$$

$$\leq \sum_{i=p+1}^{m} \frac{x[i]^2 + y[i]^2}{2} = \frac{1}{2} \sum_{i=p+1}^{m} x[i]^2 + \frac{1}{2} \sum_{i=p+1}^{m} y[i]^2$$

$$= \frac{1}{2} \left\|x_p''\right\|^2 + \frac{1}{2} \left\|y_p''\right\|^2$$

$$\leq \frac{1}{2} \left\|x_p''\right\|^2 + \frac{1}{2}$$

Filtering Statements

- **Prefix filtering**

$$\mathrm{dot}(x'_p, y) = 0 \wedge \mathrm{dot}(x''_p, \mathcal{M}) < t \Longrightarrow \mathrm{dot}(x, y) < t$$

$$\mathrm{dot}(x'_p, y) = 0 \wedge \mathrm{maxw(x)} \geq \mathrm{maxw(y)} \wedge \mathrm{dot}(x'', \mathcal{M}(x)) < t \Longrightarrow \mathrm{dot}(x, y) < t$$

$$\mathrm{dot}(x'_p, y) = 0 \wedge \frac{1}{2}\left\|x''_p\right\|^2 + \frac{1}{2} < t \Longrightarrow \mathrm{dot}(x, y) < t$$

- **Size filtering**

$$\min(|x|, |y|) \cdot \mathrm{maxw}(x) \cdot \mathrm{maxw}(y) < t \Longrightarrow \mathrm{dot}(x, y) < t$$

$$|y| \cdot \mathrm{maxw}(x) \cdot \mathrm{max}w(y) < t \Longrightarrow \mathrm{dot}(x, y) < t$$

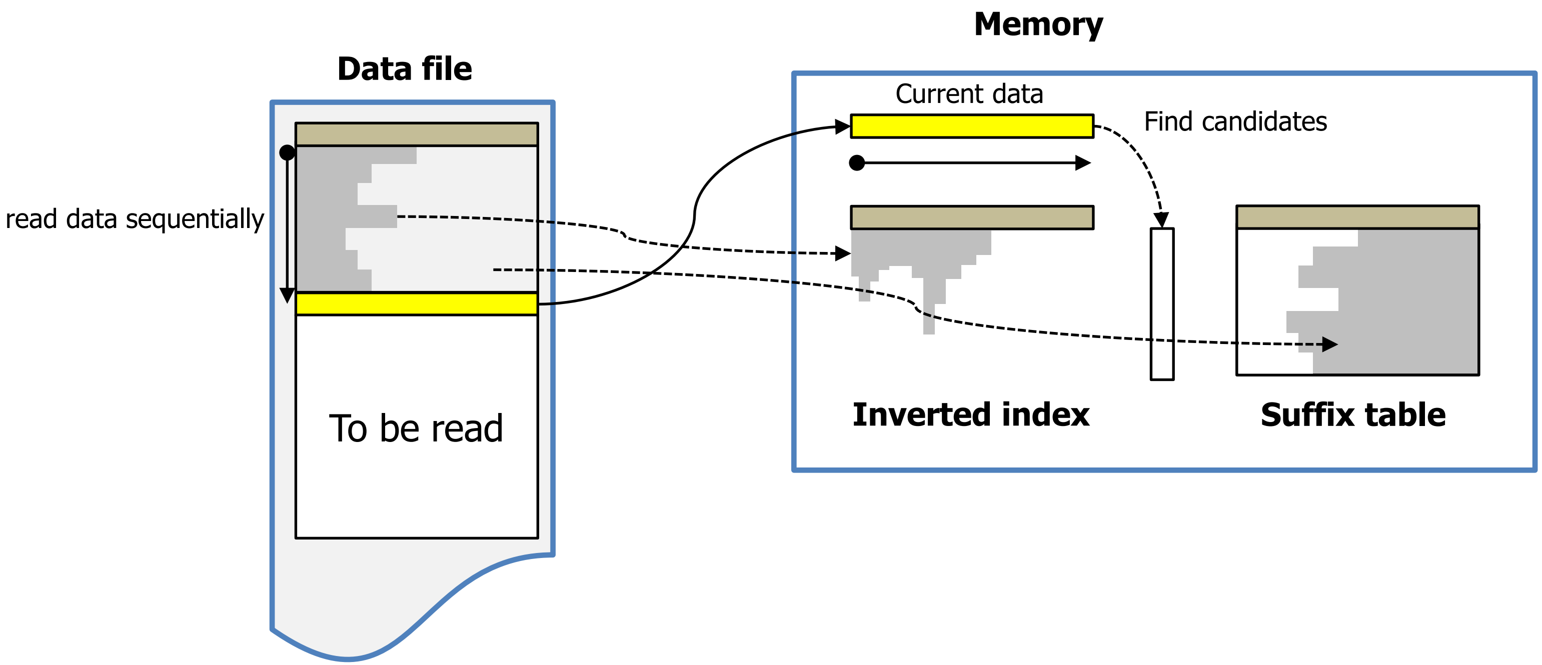
$$|y| \cdot \mathrm{maxw}(y) < \frac{t}{\mathrm{maxw}(x)} \Longrightarrow \mathrm{dot}(x, y) < t$$

$$\mathrm{dot}(x, y'_q) + \min(|x|, |y''_q|) \cdot \mathrm{maxw}(x) \cdot \mathrm{maxw}(y''_q) \Longrightarrow \mathrm{dot}(x, y) < t$$

- **Length filtering**

$$\mathrm{dot}(x'_p, y'_p) + \frac{1}{2}\left\|x''_p\right\|^2 + \frac{1}{2}\left\|y''_p\right\|^2 < t \Longrightarrow \mathrm{dot}(x, y) < t$$

General Architecture of Filtering-based Methods



Preprocessing

1) Store maxw(x) and maxwi(D) for all vectors and dimensions
2) Sort dimensions as least nonblank entries come first
3) Sort vectors in descending order of maxw(x)

MMJoin

Algorithm 2 $\texttt{MMJoin}(\mathcal{D}, t)$

Input: $\mathcal{D} = \{o_1, o_2, \ldots, o_n\}$, similarity threshold t
Output: $\{(x, y, \mathrm{dot}(x, y)) | x, y \in \mathcal{D} \wedge \mathrm{dot}(x, y) \geq t\}$

Reorder the dimension $1 \ldots m$ such that dimension with the least non-zero entries in $\mathcal{D}$ appear first
Denote the max. of $x[i]$ over all $x \in \mathcal{D}$ as $\mathrm{maxw}_i(\mathcal{D})$
Denote the max. of $x[i]$ for $1 \ldots m$ as $\mathrm{maxw}(x)$
$O \leftarrow \emptyset$
$I_1, I_2, \ldots, I_m \leftarrow \emptyset$
for each $x \in \mathcal{D}$ in decreasing order of $\mathrm{maxw}(x)$ **do**
    $C \leftarrow \texttt{Find-Candidates}(x, I_1, I_2, \ldots, I_m, t)$
    $O \leftarrow O \cup \texttt{Verify}(x, C, t)$
    $\texttt{Index}(x, I_1, I_2, \ldots, I_m, t)$
return O

Find-Candidates

Algorithm 3 Find-Candidates$(x, I_1, I_2, \ldots, I_m, t)$

Input: a vector x, inverted lists $I_1, I_2, \ldots, I_m$, similarity threshold t

Output: $\{(y, \mathrm{dot}(x, y'_q)) | (y, y[i], l_y) \in I_i \land \mathrm{dot}(x'_p, y'_q) > 0\}$

$C \leftarrow$ empty map from id to weight
$b_1 \leftarrow \sum_{i=1}^{m} x[i] \cdot \mathrm{maxw}_i(\mathcal{D})$
$b_2 \leftarrow 1$
$l_x \leftarrow \frac{1}{2}$
$minsize \leftarrow \frac{t}{\mathrm{maxw}(x)}$
for $i \leftarrow 1$ **to** m such that $x[i] > 0$ **do**
  $\boxed{\text{Remove } (y, y[i], l_y) \text{ from } I_i \text{ s.t. } |y| \cdot \mathrm{maxw}(y) < minsize}$ size filtering
  $l_x \leftarrow l_x - \frac{1}{2} x[i]^2$
  for each $(y, y[i], l_y) \in I_i$ **do**
    $\boxed{\textbf{if } \min(b_1, b_2) \geq t \textbf{ or } C[y] > 0 \textbf{ then}}$ prefix filtering
      $C[y] \leftarrow C[y] + x[i] \cdot y[i]$
      $\boxed{\textbf{if } C[y] + l_x + l_y) < t \textbf{ then}}$ length filtering
        $C[y] \leftarrow 0$
  $b_1 \leftarrow b_1 - x[i] \cdot \mathrm{maxw}_i(\mathcal{D})$
  $b_2 \leftarrow l_x + \frac{1}{2}$
  $x.l[i] \leftarrow l_x$
return C

Index

Algorithm 4 Index(x, $I_1, I_2, \ldots, I_m$, t)

Input: a vector x, inverted lists $I_1, I_2, \ldots, I_m$, similarity threshold t

$b_1 \leftarrow \sum_{i=1}^{m} x[i] \cdot \min(\text{maxw}_i(\mathcal{D}), \text{maxw}(x))$
$b_2 \leftarrow 1$
$l \leftarrow \frac{1}{2}$
for $i \leftarrow 1$ **to** m **such that** $x[i] > 0$ **do**
    $l \leftarrow l - \frac{1}{2}x[i]^2$
    $I_i \leftarrow I_i \cup \{(x, x[i], l)\}$
    remove $x[i]$ and $x.l[i]$
    $b_1 \leftarrow b_1 - x[i] \cdot \min(\text{maxw}(x), \text{maxw}_i(\mathcal{D}))$
    $b_2 \leftarrow b_2 - \frac{1}{2}x[i]^2$
    if $\min(b_1, b_2) < t$ **then**
        break;

Verify

Algorithm 5 Verify(x, C, t)

Input: a vector x, a map from id to weight C, threshold t
Output: $\{(x, y, \mathrm{dot}(x,y)) | C[y] > 0 \wedge \mathrm{dot}(x,y) \geq t\}$

```
O ← ∅
for each y ∈ C do
    if C[y] + min(|x|, |y''_q|) · maxw(x) · maxw(y''_q) ≥ t then
        unmatched ← 0
        d ← C[y]
        for i ← q + 1 to m such that y[i] > 0 do
            if x[i] > 0 then
                d ← d + x[i] · y[i]
                if unmatched > 1 then
                    if d + x.l[i] + y.l[i] < t then
                        d ← 0
                        break;
                    unmatched ← 0
            else
                unmatched ← unmatched + 1
        if d ≥ t then
            O ← O ∪ {(x, y, d)}
return O
```

size filtering

length filtering

MMJoin

maxw		G	L	N	C	F	I	M	A	D	J	K	O	B	E	H
0.62	o1								0.46		0.62	0.31 0.16	0.15 0.14	0.31 0.10	0.31 0.05	0.31 0.00
0.62	o2								0.46	0.31	0.62		0.15 0.14	0.31 0.10	0.31 0.05	0.31 0.00
0.61	o5		0.46	0.46 0.29	0.30 0.24				0.15 0.23						0.30 0.19	0.61 0.00
0.55	o4				0.18 0.49	0.18 0.47	0.18 0.46	0.18 0.45		0.18 0.44		0.37 0.36	0.37 0.29	0.55 0.14	0.37 0.07	0.37 0.00
0.50	o3	0.33				0.33	0.33	0.17		0.17	0.50	0.50		0.33		
$maxw_i$		0.33	0.46	0.46	0.30	0.33	0.33	0.18	0.46	0.31	0.62	0.50	0.37	0.55	0.37	0.61

C[y]	o1	o2	o5	o4	o3
o1		0.91			
o2				0.06	
o5					
o4					
o3					

Candidate map

G	L	N	C	F	I	M	A	D	J	K	O	B	E	H
	o5 0.46 0.39						o1 0.46 0.40	o2 0.31 0.35	o1 0.62 0.20					
							o2 0.46 0.40		o2 0.62 0.16					

Inverted index

$$\mathrm{dot}(x'_D, y'_D) + \frac{1}{2}\|x''_D\|^2 + \frac{1}{2}\|y''_D\|^2 < t \Longrightarrow \mathrm{dot}(x, y) < t$$

$$0.06 + 0.44 + 0.35 = 0.85 < 0.9$$

Experimental Evaluation

- Time Performance of Filtering Techniques
  - Average prefix size, candidate size, computation time

- Previous filtering techniques (Bayardo et al., WWW 2007)
  - All-Pairs0: Basic prefix filtering
  - All-Pairs1: Exploiting a vector sort order
  - All-Pairs2: Size filtering in the candidate generation phase
  - All-Pairs: Size filtering in the verification phase
- Our filtering techniques
  - MMJoin0: Improved prefix selection
  - MMJoin1: Length filtering in the candidate generation phase
  - MMJoin2: Length filtering in the verification phase

Experimental Setup

- Environment
  - Java 1.6, standard libraries
  - 2.83 GHz Intel Core2 Quad, 8 Gbytes of RAM and two 7200 RPM SATA II-IDE hard drives.
- Datasets
  - DBLP (Bibliographic dataset), TREC filtering dataset (Text REtrieval Conference), LASTFM (Song, title and artist)
  - UKBENCH (University of Kentucky, image benchmark dataset)
    - 10200 images (640*480), 4 picture images for the same scene with different angles or distances
    - Feature vectors are extracted and abstracted as visual words

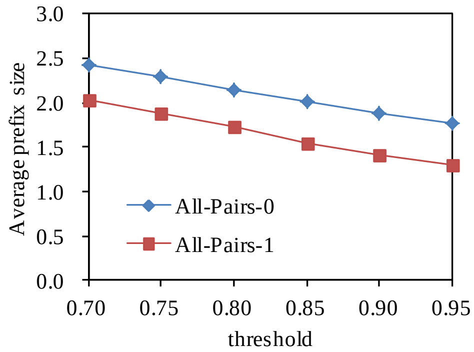
Experimental Setup

- Tokenization
  - by punctuation.
  - by punctuation and into 4grams.
- Weighting
  - tf-idf weighting scheme

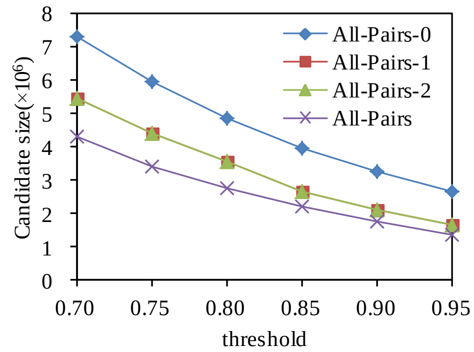
- Dataset Statistics

Dataset	n	avg($size$)	$\lvert\mathcal{U}\rvert$	avg(DF)
LAST.FM	134,949	4.8	47,295	13.8
LAST.FM 4GRAM		11.2	44,272	34.3
DBLP	1,298,016	8.6	381,450	29.3
DBLP 4GRAM		23.9	135,204	224.5
TREC	348,566	77.1	298,302	90.1
TREC 4GRAM		232.4	121,252	658.8
UKBENCH	10,200	425.7	533,412	6.9

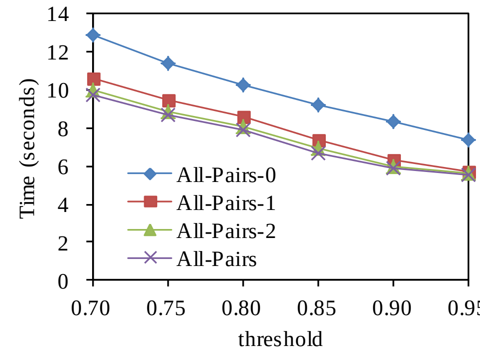
Effects of Previous Filtering Techniques



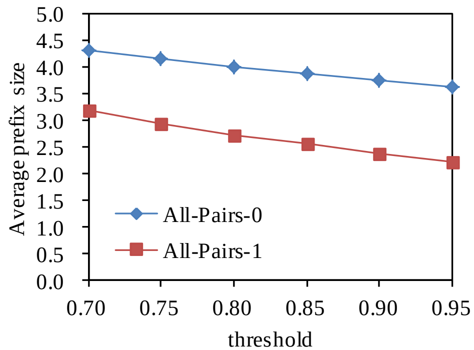
LASTFM, Average prefix size



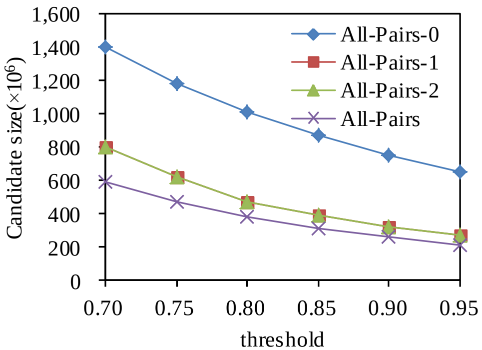
LASTFM, Candidate size



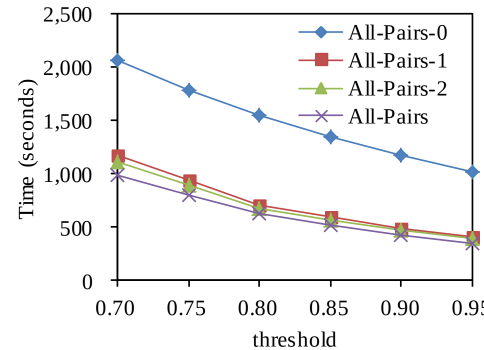
LASTFM, Computation Time



DBLP, Average prefix size

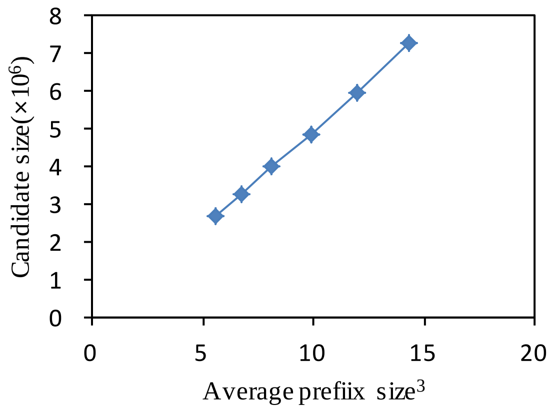


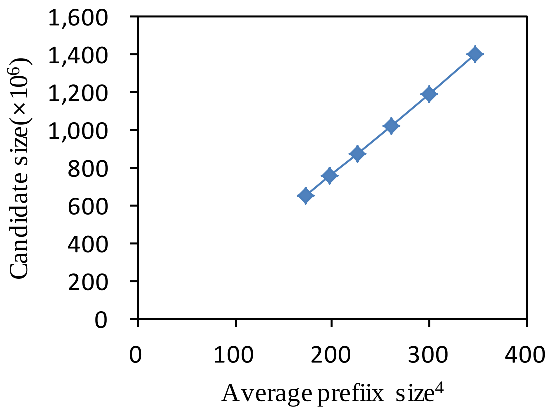
DBLP, Candidate size



DBLP, Computation Time

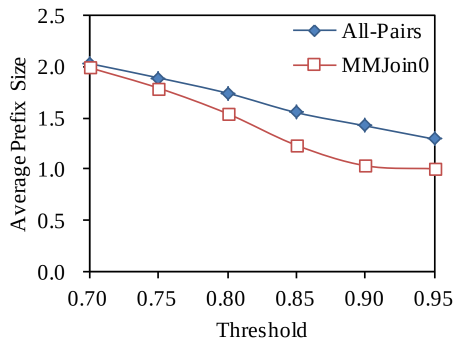
Relationship between Prefix Size and Candidate Size



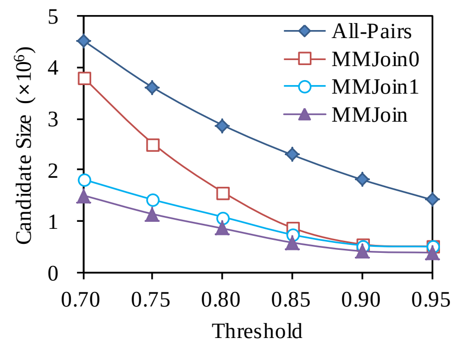


CEBT

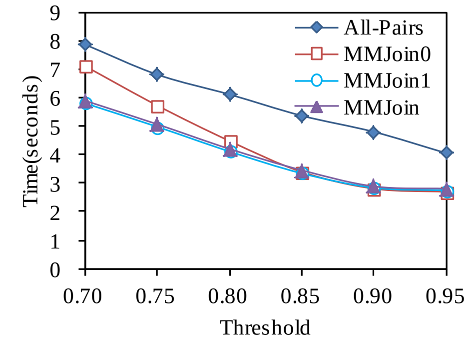
Effects of Proposed Filtering Techniques (1/3)



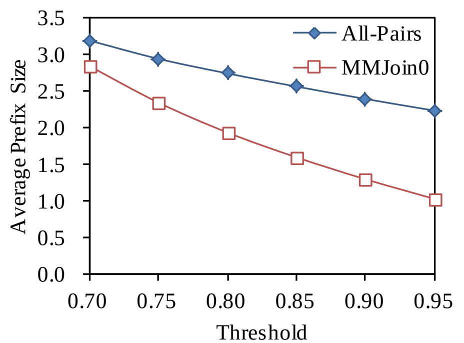
LASTFM, Average prefix size



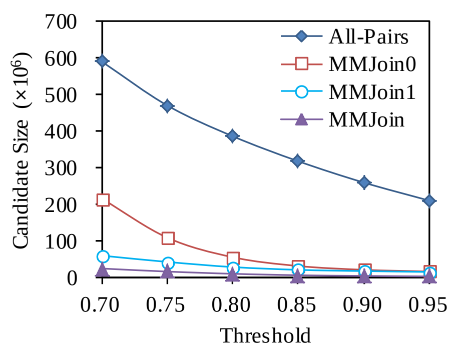
LASTFM, Candidate size



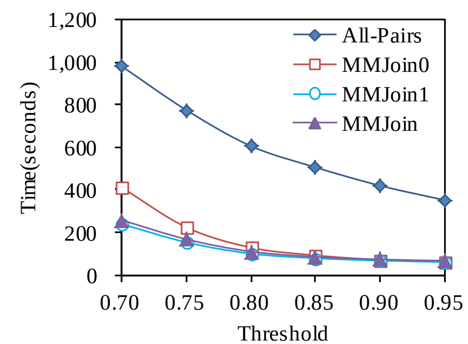
LASTFM, Computation Time



DBLP, Average prefix size



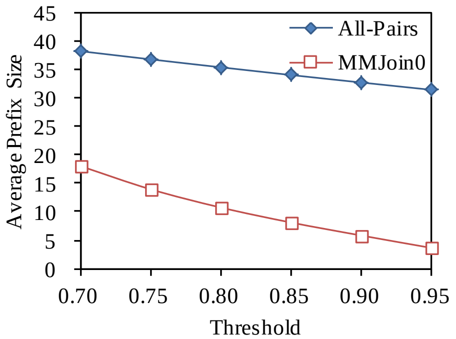
DBLP, Candidate size



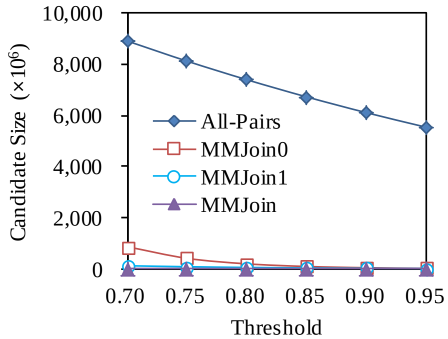
DBLP, Computation Time

CEBT

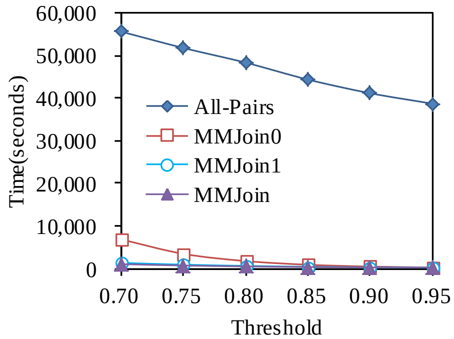
Effects of Proposed Filtering Techniques (2/3)



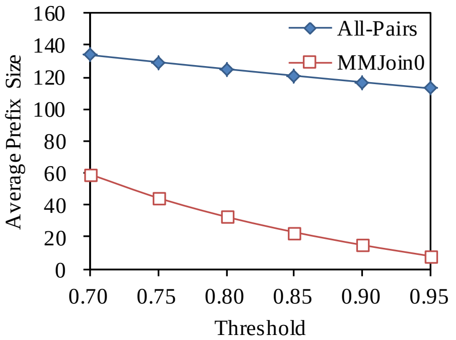
TREC, Average prefix size



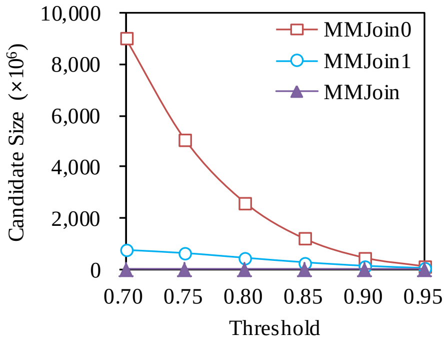
TREC, Candidate size



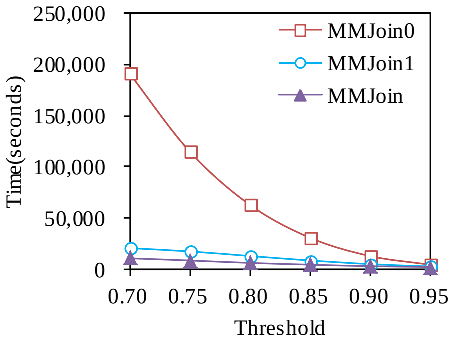
TREC, Computation Time



TREC 4GRAM, Average prefix size

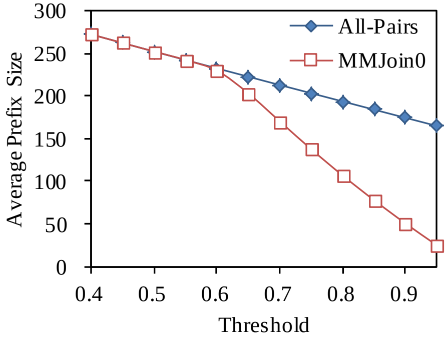


TREC 4GRAM, Candidate size

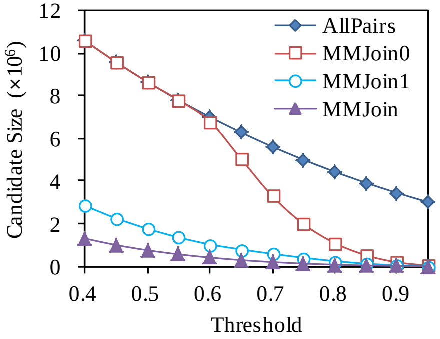


TREC 4GRAM, Computation Time

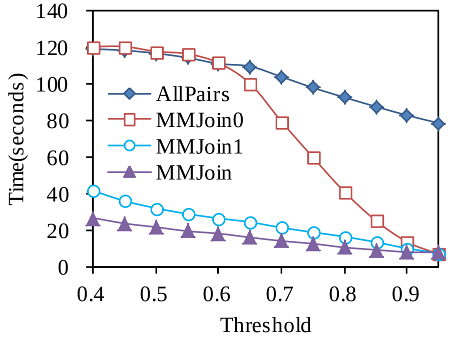
Effects of Proposed Filtering Techniques (3/3)



UKBENCH, Average prefix size



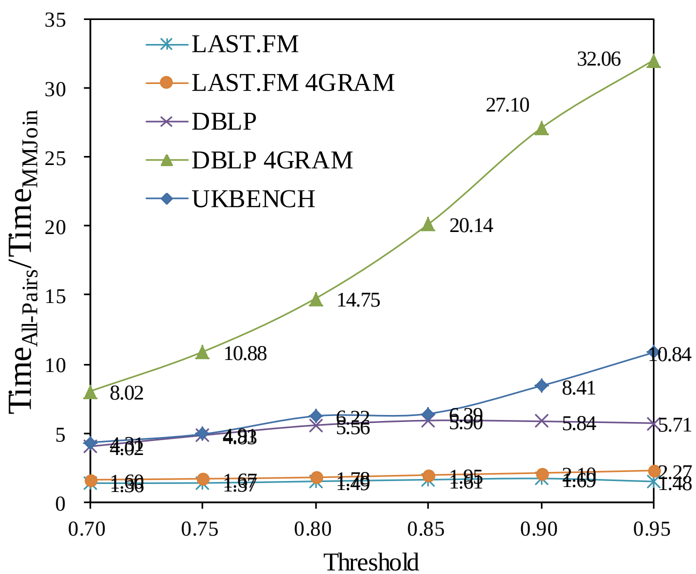
UKBENCH, Candidate size

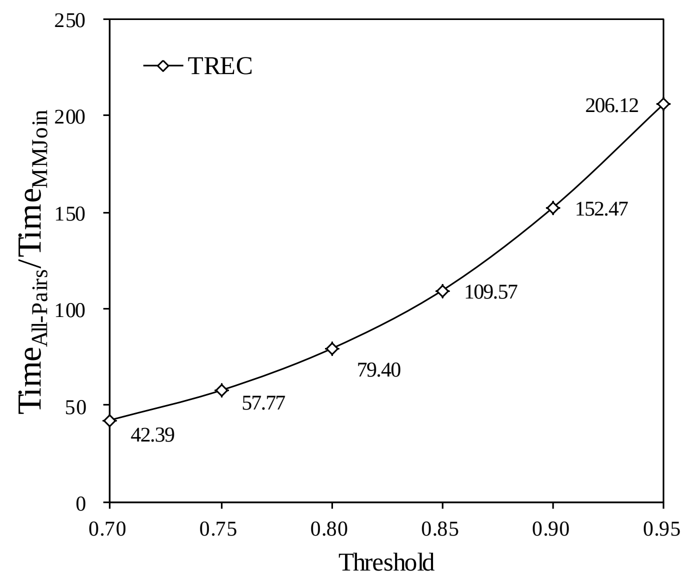


UKBENCH, Computation Time

Performance Differences

- MMJoin outperforms All-Pairs in all experimental settings





Inner Product Family

- Inner product

$$\mathrm{dot}(x,y) = \sum_{i=1}^{m} x[i] \cdot y[i]$$

- Cosine

$$\mathrm{C}(x,y) = \frac{\sum_{i=1}^{m} x[i] \cdot y[i]}{\sqrt{\sum_{i=1}^{m} x[i]^2}\sqrt{\sum_{i=1}^{m} y[i]^2}} = \frac{\mathrm{dot}(x,y)}{\|x\|\|y\|}$$

- Tanimoto (extended Jaccard)

$$\mathrm{T}(x,y) = \frac{\sum_{i=1}^{m} x[i] \cdot y[i]}{\sum_{i=1}^{m} x[i]^2 + \sum_{i=1}^{m} y[i]^2 - \sum_{i=1}^{m} x[i] \cdot y[i]} = \frac{\mathrm{dot}(x,y)}{\|x\|^2 + \|y\|^2 - \mathrm{dot}(x,y)}$$

- Dice

$$\mathrm{D}(x,y) = \frac{2\sum_{i=1}^{m} x[i] \cdot y[i]}{\sum_{i=1}^{m} x[i]^2 + \sum_{i=1}^{m} y[i]^2} = \frac{\mathrm{dot}(x,y)}{\frac{1}{2}(\|x\|^2 + \|y\|^2)}$$

Extension to Dice and Tanimoto Similarity

- Tanimoto similarity (Extended Jaccard similarity)
  - $\mathrm{T}(x, y) = \frac{\mathrm{dot}(x,y)}{\|x\|^2+\|y\|^2-\mathrm{dot}(x,y)}$
- Dice similarity
  - $\mathrm{D}(x, y) = \frac{\mathrm{dot}(x,y)}{\frac{\|x\|^2+\|y\|^2}{2}}$

Observation 1.

- Relationship among similarity functions

$$\mathrm{T}(x,y) = \frac{\mathrm{dot}(x,y)}{\|x\|^2 + \|y\|^2 - \mathrm{dot}(x,y)} \le \mathrm{D}(x,y) = \frac{\mathrm{dot}(x,y)}{\frac{\|x\|^2 + \|y\|^2}{2}} \le \mathrm{C}(x,y) = \frac{\mathrm{dot}(x,y)}{\|x\|\|y\|}$$

- Cosine similarity constraints is the weakest constraint.

$$\mathrm{T}(x,y) \ge t \Longrightarrow \mathrm{D}(x,y) \ge t \Longrightarrow \mathrm{C}(x,y) \ge t$$

$$\mathrm{C}(x,y) < t \Longrightarrow \mathrm{D}(x,y) < t \Longrightarrow \mathrm{T}(x,y) < t$$

Observation 2.

- x_0 and y_0 are unnormalized vector, let x and y be normalized versions of two vectors, i.e., $x[i] = \frac{x_0[i]}{\|x\|}$ and $y[i] = \frac{y_0[i]}{\|y\|}$.

- Property of the Dice and Tanimoto similarity

  $\mathrm{T}(x_0, y_0) \le \mathrm{T}(x, y)$

  $\mathrm{D}(x_0, y_0) \le \mathrm{D}(x, y) = \mathrm{C}(x, y)$

Proof of Observation 2.

$$\mathrm{T}(x_0, y_0) = \frac{\mathrm{dot}(x_0, y_0)}{\|x_0\|^2 + \|y_0\|^2 - \mathrm{dot}(x_0, y_0)}$$

$$= \frac{\dfrac{\mathrm{dot}(x_0, y_0)}{\|x_0\|\|y_0\|}}{\dfrac{\|x_0\|^2 + \|y_0\|^2}{\|x_0\|\|y_0\|} - \dfrac{\mathrm{dot}(x_0, y_0)}{\|x_0\|\|y_0\|}}$$

$$= \frac{\mathrm{dot}(x, y)}{\dfrac{\|x_0\|^2 + \|y_0\|^2}{\|x_0\|\|y_0\|} - \mathrm{dot}(x, y)}$$

$$\frac{\|x_0\|^2 + \|y_0\|^2}{\|x_0\|\|y_0\|} \geq 2$$

$$\leq \frac{\mathrm{dot}(x, y)}{2 - \mathrm{dot}(x, y)} = \mathrm{T}(x, y)$$

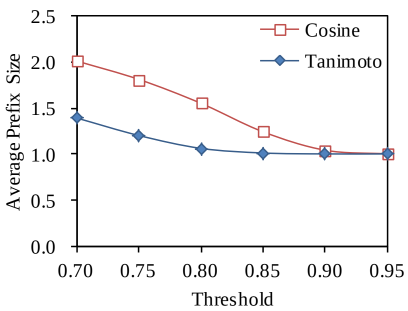
Extended Algorithm

Algorithm 6 $\texttt{MMJoin}(\mathcal{D}, \text{sim}(), t)$

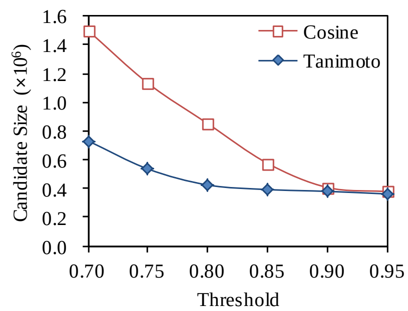
Input: $\mathcal{D} = \{o_1, o_2, \ldots, o_n\}$, similarity function sim(), similarity threshold t
Output: $\{(x_0, y_0, \text{sim}(x_0, y_0)) | x_0, y_0 \in \mathcal{D} \wedge \text{sim}(x_0, y_0) \geq t\}$

1: Store the length of each vector and normalize all the vectors
2: $\alpha \leftarrow \frac{2t}{1+t}$ in case of Tanimoto similarity, otherwise $\alpha \leftarrow t$
3: $C \leftarrow \texttt{MMJoin}(\mathcal{D}, \alpha)$
4: $O \leftarrow \emptyset$
5: **for each** $(x, y, \text{dot}(x, y)) \in C$ **do**
6: $\quad s \leftarrow \texttt{Calculate}_{\text{sim}}(\|x_0\|, \|y_0\|, \text{dot}(x, y))$
7: $\quad$ **if** $s \geq t$ **then**
8: $\quad\quad O \leftarrow O \cup \{(x_0, y_0, s)\}$
9: **return** O

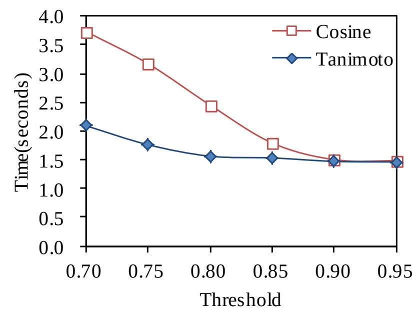
Experiments



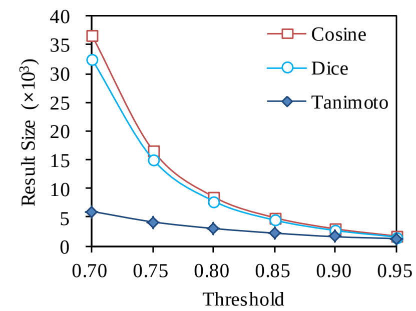
LASTFM, Average prefix size



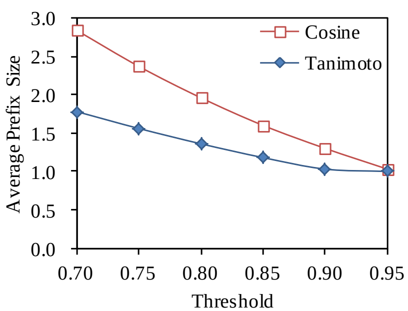
LASTFM, Candidate size



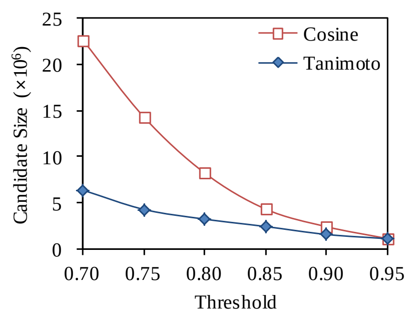
LASTFM, Computation Time



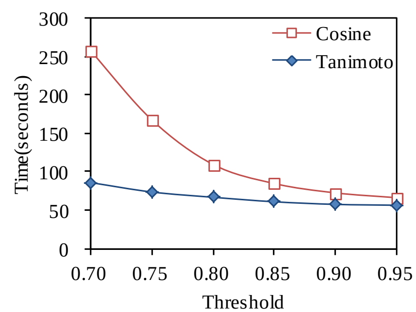
LASTFM, Result Size



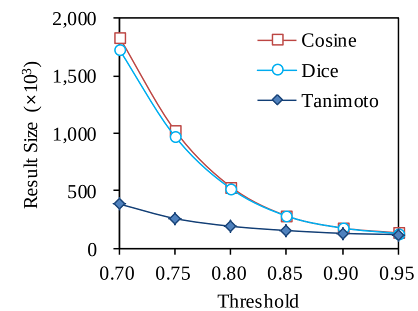
DBLP, Average prefix size



DBLP, Candidate size



DBLP, Computation Time



DBLP, Result Size

Conclusion

- We formalized filtering techniques as **filtering statements** in mathematical way.
- We proposed new cosine similarity upper bounds that is computable with little overhead.
- We proposed an efficient algorithm (MMJoin) for cosine similarity joins by exploiting the proposed similarity upper bounds with little overhead.
- We extended MMJoin to Dice and Tanimito similarity joins.
- We showed that MMJoin outperforms a state-of-the-art algorithm using empirical evaluation with large scale datasets.

Future Work

- General filtering framework for other similarity functions
  - 8 groups of 38 distance (divergence) and **13 similarity measures** [Cha, 2007]
    1) Intersection family
    2) Inner Product family
    3) Fidelity family (or squared-chord family)

- Empirical study on measuring similarity
  - with various feature extraction methods
  - with various weighting schemes
  - with various similarity functions and thresholds