L2AP: Fast Cosine Similarity Search With Prefix L-2 Norm Bounds

David C. Anastasiu and George Karypis
University of Minnesota, Minneapolis, MN, USA

April 3, 2014

All-Pairs Similarity Search (APSS)

Goal

- For each object in a set, find all other set objects with a similarity value of at least t (its neighbors)

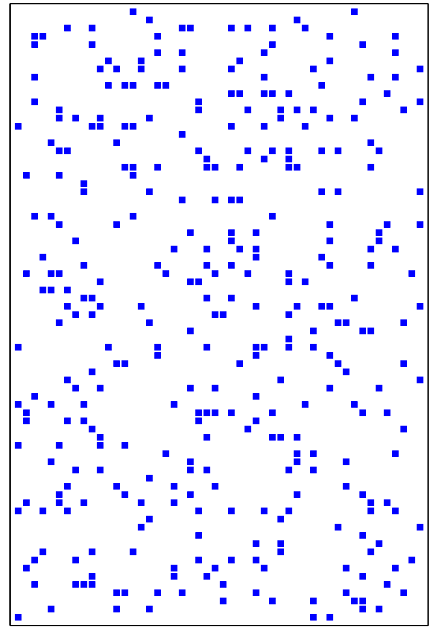
Applications

- Near-duplicate Document Detection
- Clustering
- Query Refinement
- Collaborative Filtering
- Semi-supervised Learning
- Information Retrieval

Outline

Problem Description

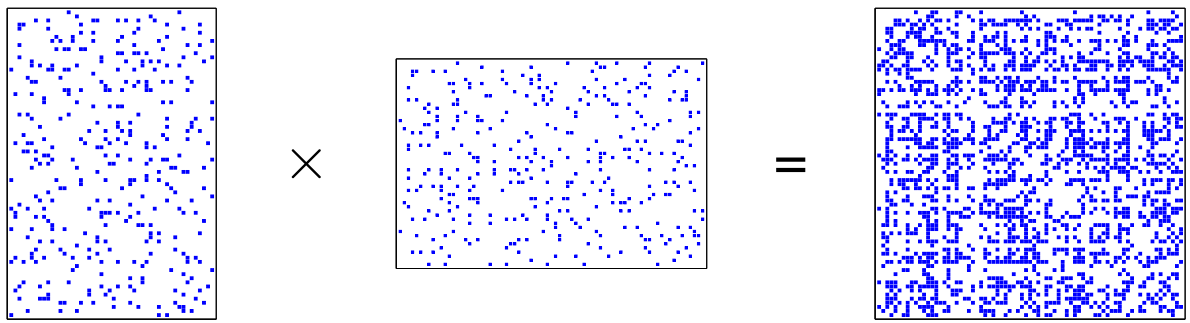
- $\mathbf{D}$, sparse matrix of size $n \times m$
- $\mathbf{x}$, row vector for row x in $\mathbf{D}$
- rows unit-length normalized, $\mathbf{x} = \frac{\mathbf{x}}{||\mathbf{x}||} \Rightarrow ||\mathbf{x}|| = 1$
- $\text{sim}(\mathbf{x}, \mathbf{y}) = \cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}\mathbf{y}^T}{||\mathbf{x}|| \times ||\mathbf{y}||} = \mathbf{x}\mathbf{y}^T = \sum_{j=1}^{m} x_j \times y_j$



Problem Description

- Naïve solution: compute similarity of each object with all others, keep results $\geq t$.
- Equivalent to sparse matrix-matrix multiplication, followed by a filter operation: APSS $\sim \mathbf{D}\mathbf{D}^T . \geq t$

for each row $x = 1, \ldots, n$ **do**
  for each row $y = 1, \ldots, n$ **do**
    if $x \neq y$ & $\text{sim}(\mathbf{x}, \mathbf{y}) > t$ **then**
      Add $\{x, y, \text{sim}(\mathbf{x}, \mathbf{y})\}$ to result



$\mathbf{D}$ $\mathbf{D}^T$ $\mathbf{DD^T}$

Problem Description

- **Main idea**: Use the similarity threshold t and theoretical bounds to prune the search space

$$\mathbf{a} = \langle \texttt{0.12, , 0.37, 0.22, , 0.47, 0.75, 0.13} \rangle$$
$$\mathbf{b} = \langle \texttt{, 0.50, 0.65, , 0.05, 0.35, 0.45, } \rangle$$
$$\mathbf{c} = \langle \texttt{0.96, 0.28, 0.01, , , , , } \rangle$$

$A[b] = 0.0000 \quad A[c] = 0.0000 \quad t = 0.5$

Problem Description

- **Main idea**: Use the similarity threshold t and theoretical bounds to prune the search space

$$\mathbf{a} = \langle \texttt{0.12}, \quad , \texttt{0.37}, \texttt{0.22}, \quad , \texttt{0.47}, \texttt{0.75}, \texttt{0.13} \rangle$$
$$\mathbf{b} = \langle \quad , \texttt{0.50}, \texttt{0.65}, \quad , \texttt{0.05}, \texttt{0.35}, \texttt{0.45}, \quad \rangle$$
$$\mathbf{c} = \langle \texttt{0.96}, \texttt{0.28}, \texttt{0.01}, \quad , \quad , \quad , \quad , \quad \rangle$$

$A[b] = 0.0000 \quad A[c] = 0.1152 \quad t = 0.5$

Problem Description

- **Main idea**: Use the similarity threshold t and theoretical bounds to prune the search space

$$\mathbf{a} = \langle 0.12, \quad , 0.37, 0.22, \quad , 0.47, 0.75, 0.13 \rangle$$
$$\mathbf{b} = \langle \quad , 0.50, 0.65, \quad , 0.05, 0.35, 0.45, \quad \rangle$$
$$\mathbf{c} = \langle 0.96, 0.28, 0.01, \quad , \quad , \quad , \quad , \quad \rangle$$

$A[b] = 0.0000 \quad A[c] = 0.1152 \quad t = 0.5$

Problem Description

- **Main idea**: Use the similarity threshold t and theoretical bounds to prune the search space

$$\mathbf{a} = \langle 0.12,\ ,\ 0.37,\ 0.22,\ ,\ 0.47,\ 0.75,\ 0.13 \rangle$$
$$\mathbf{b} = \langle\ ,\ 0.50,\ 0.65,\ ,\ 0.05,\ 0.35,\ 0.45,\ \rangle$$
$$\mathbf{c} = \langle 0.96,\ 0.28,\ 0.01,\ ,\ ,\ ,\ ,\ \rangle$$

$A[b] = 0.2405 \quad A[c] = 0.1189 \quad t = 0.5$

Problem Description

- **Main idea**: Use the similarity threshold t and theoretical bounds to prune the search space

$$\mathbf{a} = \langle \texttt{0.12, , 0.37, 0.22, , 0.47, 0.75, 0.13} \rangle$$
$$\mathbf{b} = \langle \texttt{, 0.50, 0.65, , 0.05, 0.35, 0.45, } \rangle$$
$$\mathbf{c} = \langle \texttt{0.96, 0.28, 0.01, , , , , } \rangle$$

$A[b] = 0.2405 \quad A[c] = 0.1189 \quad t = 0.5$

Problem Description

- **Main idea**: Use the similarity threshold t and theoretical bounds to prune the search space

$$\mathbf{a} = \langle 0.12, \quad , 0.37, 0.22, \quad , 0.47, 0.75, 0.13 \rangle$$
$$\mathbf{b} = \langle \quad , 0.50, 0.65, \quad , 0.05, 0.35, 0.45, \quad \rangle$$
$$\mathbf{c} = \langle 0.96, 0.28, 0.01, \quad , \quad , \quad , \quad , \quad \rangle$$

$A[b] = 0.2405 \quad A[c] = 0.1189 \quad t = 0.5$

Problem Description

- **Main idea**: Use the similarity threshold t and theoretical bounds to prune the search space

$$\mathbf{a} = \langle 0.12, \quad , 0.37, 0.22, \quad , 0.47, 0.75, 0.13 \rangle$$
$$\mathbf{b} = \langle \quad , 0.50, 0.65, \quad , 0.05, 0.35, 0.45, \quad \rangle$$
$$\mathbf{c} = \langle 0.96, 0.28, 0.01, \quad , \quad , \quad , \quad , \quad \rangle$$

$A[b] = 0.4050 \quad A[c] = 0.1189 \quad t = 0.5$

Problem Description

- **Main idea**: Use the similarity threshold t and theoretical bounds to prune the search space

$$\mathbf{a} = \langle \texttt{0.12, , 0.37, 0.22, , 0.47, 0.75, 0.13} \rangle$$
$$\mathbf{b} = \langle \texttt{, 0.50, 0.65, , 0.05, 0.35, 0.45, } \rangle$$
$$\mathbf{c} = \langle \texttt{0.96, 0.28, 0.01, , , , , } \rangle$$

$A[b] = 0.7425 \quad A[c] = 0.1189 \quad t = 0.5$

Problem Description

- **Main idea**: Use the similarity threshold t and theoretical bounds to prune the search space

$$\mathbf{a} = \langle \texttt{0.12, , 0.37, 0.22, , 0.47, 0.75, 0.13} \rangle$$
$$\mathbf{b} = \langle \texttt{, 0.50, 0.65, , 0.05, 0.35, 0.45, } \rangle$$
$$\mathbf{c} = \langle \texttt{0.96, 0.28, 0.01, , , , , } \rangle$$

$A[b] = 0.7425 \quad A[c] = 0.1189 \quad t = 0.5$

Extensions to the naïve approach

- Leverage sparsity in $\mathbf{D}$. Build an *inverted index*.
- Leverage *commutativity* of $\cos(\mathbf{x}, \mathbf{y})$. (Sarawagi and Kirpal, 2004).
- Build a *partial index*. (Chaudhuri et al., 2006)

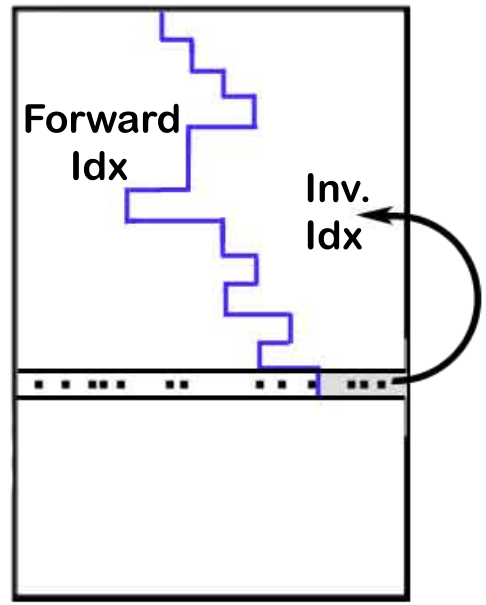
AllPairs Framework

`AllPairs`:
for each row $x = 1, \ldots, n$ **do**

Find similarity candidates for **x** using current inverted index (candidate generation)

Complete similarity computation and prune unpromising candidates (candidate verification)

Index enough of **x** to ensure all valid similarity pairs are discovered (index construction)



What we index

- We index $\mathbf{x}''$, the suffix of $\mathbf{x}$.
  $\mathbf{x}''_p = \langle 0, \ldots, 0, x_p, \ldots, x_m \rangle$ is the suffix of $\mathbf{x}$ starting at feature p.
- $\mathbf{x}'$ is the un-indexed prefix of $\mathbf{x}$.
  $\mathbf{x}'_p = \langle x_1, \ldots, x_{p-1}, 0, \ldots, 0 \rangle$ is $\mathbf{x}$'s prefix ending at $p-1$.

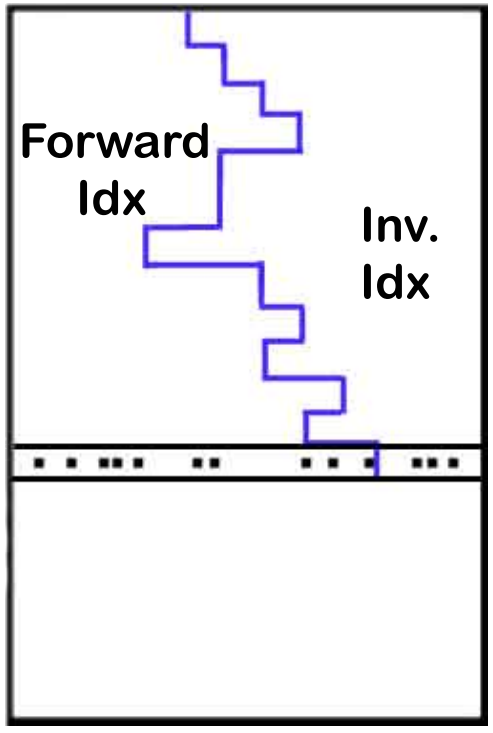
$$\mathbf{a} = \langle \texttt{0.12, , 0.37, 0.22, , 0.47, 0.75, 0.13} \rangle$$
$$\mathbf{a}''_4 = \langle \texttt{ , , , 0.22, , 0.47, 0.75, 0.13} \rangle$$
$$\mathbf{a}'_4 = \langle \texttt{0.12, , 0.37, , , , , } \rangle$$

- $\mathbf{x} = \mathbf{x}' + \mathbf{x}''$
- $$\begin{aligned} \mathbf{x}\mathbf{y}^T &= \textstyle\sum_{j=1}^{p-1} x_j \times y_j + \sum_{j=p}^{m} x_j \times y_j \\ &= \mathbf{x}'_p \mathbf{y}^T + \mathbf{x}''_p \mathbf{y}^T \end{aligned}$$
- $\cos(\mathbf{x}, \mathbf{y}) \leq ||\mathbf{x}|| \times ||\mathbf{y}||$ (Cauchy–Schwarz inequality)

Index Construction

- Add a minimum number of non-zero features j of $\mathbf{x}$ to the inverted index lists I_j (index filtering).

for each column $j = 1, \ldots, m$ s.t. $x_j > 0$ **do**
  if $\text{sim}(\mathbf{x}'_{j+1}, \mathbf{y}) \geq t,\ \forall y > x$ **then**
    $I_j \leftarrow I_j \cup \{(x, x_j)\}$



Index Construction

- By the *Cauchy–Schwarz inequality*, $\cos(\mathbf{x}'_j, \mathbf{y}) \leq ||\mathbf{x}'_j|| \times ||\mathbf{y}|| = ||\mathbf{x}'_j||$, since $||\mathbf{y}|| = 1$ (bound b_3).
- We store $||\mathbf{x}'_j||$ along with x_j in the index to use for later pruning.

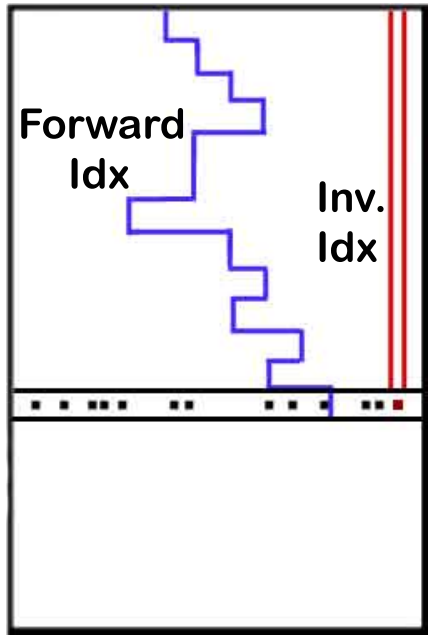
$$\mathbf{a} = \langle \texttt{0.12, , 0.37, 0.22, , 0.47, 0.75, 0.13} \rangle$$
$$||\mathbf{a}_j|| = \langle \texttt{0.12, 0.12, 0.39, 0.45, 0.45, 0.65, 0.99, 1.00} \rangle$$

Index Construction

- Let $\mathbf{w} = \langle \max_z z_1, \ldots, \max_z z_m \rangle$, the vector of max column values in $\mathbf{D}$. We can estimate $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \leq \text{sim}(\mathbf{x}'_j, \mathbf{w})$.
- Leverage an order of $\mathbf{D}$'s rows. Order rows in decreasing max row value ($||\mathbf{z}||_\infty$) order. Let $\hat{\mathbf{w}} = \langle \min(x_1, \max_z z_1), \ldots, \min(x_n, \max_z z_m) \rangle$.
  Then $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \leq \text{sim}(\mathbf{x}'_j, \hat{\mathbf{w}})$, since the y's we seek follow x in the row order (bound b_1, Bayardo et al., 2007).
- We use the minimum of the two bounds, $\min(b_1, b_3)$.
- We store $ps[x] \leftarrow \min(\text{sim}(\mathbf{x}'_j, \hat{\mathbf{w}}), ||\mathbf{x}_j||)$ to use in later pruning.

Candidate generation

- Traverse the inverted index lists I_j corresponding to non-zero features j of $\mathbf{x}$ and keep track of a partial dot product ($A[y]$) for the candidates encountered.



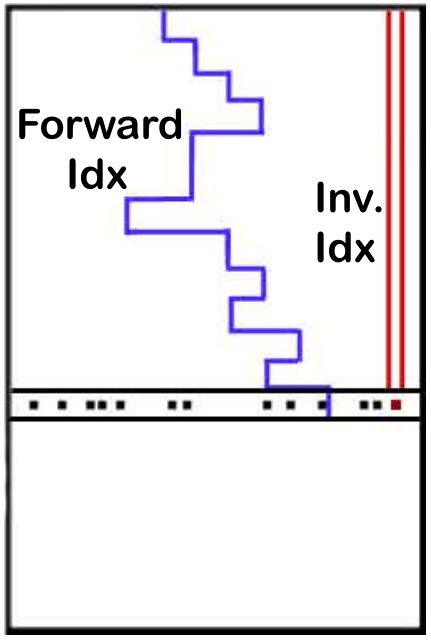
for each column $j = m, \ldots, 1$ s.t. $x_j > 0$ **do**
  for each $(y, y_j) \in I_j$ **do**
    if $A[y] > 0$ **or** $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \geq t$ **then**
      $A[y] \leftarrow A[y] + x_j \times y_j$
      $A[y] \leftarrow 0$ **if** $A[y] + \text{sim}(\mathbf{x}'_j, \mathbf{y}'_j) < t$

- Note that we are accumulating the *suffix dot product*, $\text{sim}(\mathbf{x}'', \mathbf{y})$.

Candidate generation

- Leverage t to prune potential candidates (residual filtering).

for each column $j = m, \ldots, 1$ s.t. $x_j > 0$ **do**
  for each $(y, y_j) \in I_j$ **do**
    if $A[y] > 0$ **or** $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \geq t$ **then**
      $A[y] \leftarrow A[y] + x_j \times y_j$
      $A[y] \leftarrow 0$ **if** $A[y] + \text{sim}(\mathbf{x}'_j, \mathbf{y}'_j) < t$



- Accumulate only if $A[y] > 0$ or $||\mathbf{x}'_j|| \geq t$, since $\cos(\mathbf{x}'_j, \mathbf{y}) \leq ||\mathbf{x}'_j||$ (bound rs_4).
- Once the ℓ^2 norm of $\mathbf{x}'_j$ falls below t, we ignore potential candidates $\mathbf{y}$ if $A[y] = 0$.

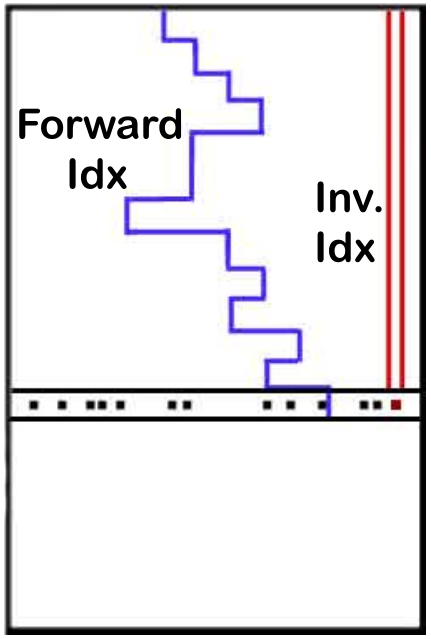
Candidate generation

- Given $\mathbf{w}$ defined as before, $\text{sim}(\mathbf{x}', \mathbf{y}) \leq \text{sim}(\mathbf{x}', \mathbf{w})$ (bound rs_1, Bayardo et al., 2007). We pre-compute $rs_1 = \mathbf{x}\mathbf{w}^T$, and roll back the computation as we process each inverted index column j. We stop accumulating *new candidates* once $rs_1 < t$.
- Candidates can only be those vectors with lower ids. We can improve rs_1 by using max column values of processed columns instead, $\widetilde{\mathbf{w}} = \langle \max_{z<x} z_1, \ldots, \max_{z<x} z_m \rangle$, thus $\text{sim}(\mathbf{x}', \mathbf{y}) \leq \text{sim}(\mathbf{x}', \widetilde{\mathbf{w}})$ (bound rs_3).
- We use the best of both bounds, $\min(rs_3, rs_4)$, during residual filtering.

Candidate generation

- Leverage t at common features to prune actual candidates (positional filtering).

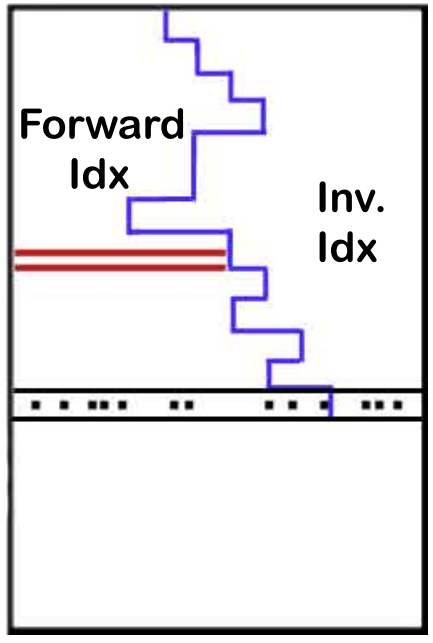
for each column $j = m, \ldots, 1$ s.t. $x_j > 0$ **do**
  for each $(y, y_j) \in I_j$ **do**
    if $A[y] > 0$ **or** $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \geq t$ **then**
      $A[y] \leftarrow A[y] + x_j \times y_j$
      $A[y] \leftarrow 0$ **if** $A[y] + \text{sim}(\mathbf{x}'_j, \mathbf{y}'_j) < t$



- We estimate $\text{sim}(\mathbf{x}'_j, \mathbf{y}'_j) \leq ||\mathbf{x}'_j|| \times ||\mathbf{y}'_j||$ ($||\mathbf{y}'_j||$ is stored in the index), to prune some of the candidates (bound *l2cg*).
- We store $||\mathbf{x}'_j||$ for forward index features to use in future pruning.

Candidate verification

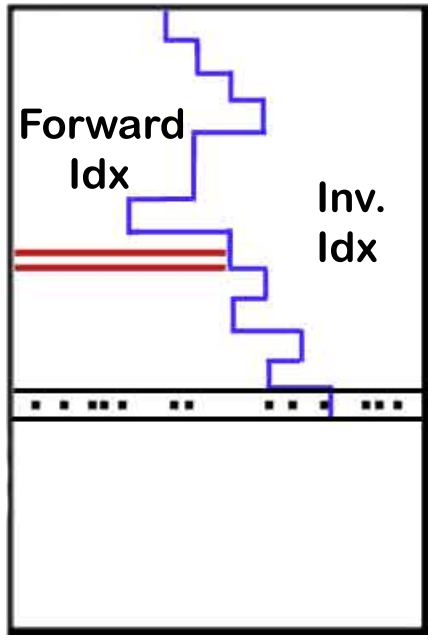
- We use the forward index to finish computing the dot products for the encountered candidates, vectors **y** with $A[y] > 0$.



for each y s.t. $A[y] > 0$ **do**
  next y **if** $A[y] + \text{sim}(\mathbf{x}, \mathbf{y}') < t$
  for each column j s.t. $y_j > 0 \wedge y_j \notin I_j \wedge x_j > 0$ **do**
    $A[y] \leftarrow A[y] + x_j \times y_j$
    next y **if** $A[y] + \text{sim}(\mathbf{x}'_j, \mathbf{y}'_j) < t$
  Add $\{x, y, A[y]\}$ to result **if** $A[y] > t$

Candidate verification

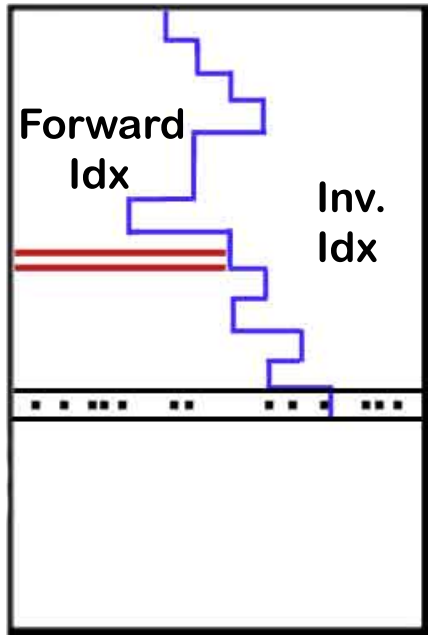
- Leverage t and the stored pscore $ps[y]$ obtained when indexing $\mathbf{y}$, to prune candidates (pscore filtering).
- Note that, after candidate generation, $A[y] = \text{sim}(\mathbf{x}, \mathbf{y}'')$. $ps[y]$ is an estimate of $\text{sim}(\mathbf{z}, \mathbf{y}'), \forall z > y$, including x, i.e., $\text{sim}(\mathbf{x}, \mathbf{y}') \leq ps[y]$.
- Prune if $A[y] + ps[y] < t$.



for each $\mathbf{y}$ s.t. $A[y] > 0$ **do**
  next y **if** $A[y] + \text{sim}(\mathbf{x}, \mathbf{y}') < t$
  for each column j s.t. $y_j > 0 \wedge y_j \notin I_j \wedge x_j > 0$ **do**
    $A[y] \leftarrow A[y] + x_j \times y_j$
    next y **if** $A[y] + \text{sim}(\mathbf{x}'_j, \mathbf{y}'_j) < t$
  Add $\{x, y, A[y]\}$ to result **if** $A[y] > t$

Candidate verification

- While computing the final dot product, we estimate $\text{sim}(\mathbf{x}, \mathbf{y}'_j) \leq ||x'_j|| \times ||y'_j||$ and use it to prune additional candidates. (bound *l2cv*).
- Additional pruning obtained via `dpscore` and `minsize` filtering is described in the paper.



for each y s.t. $A[y] > 0$ **do**
  next y **if** $A[y] + \text{sim}(\mathbf{x}, \mathbf{y}') < t$
  for each column j s.t. $y_j > 0 \wedge y_j \notin l_j \wedge x_j > 0$ **do**
    $A[y] \leftarrow A[y] + x_j \times y_j$
    next y **if** $A[y] + \text{sim}(\mathbf{x}'_j, \mathbf{y}'_j) < t$
  Add $\{x, y, A[y]\}$ to result **if** $A[y] > t$

Datasets

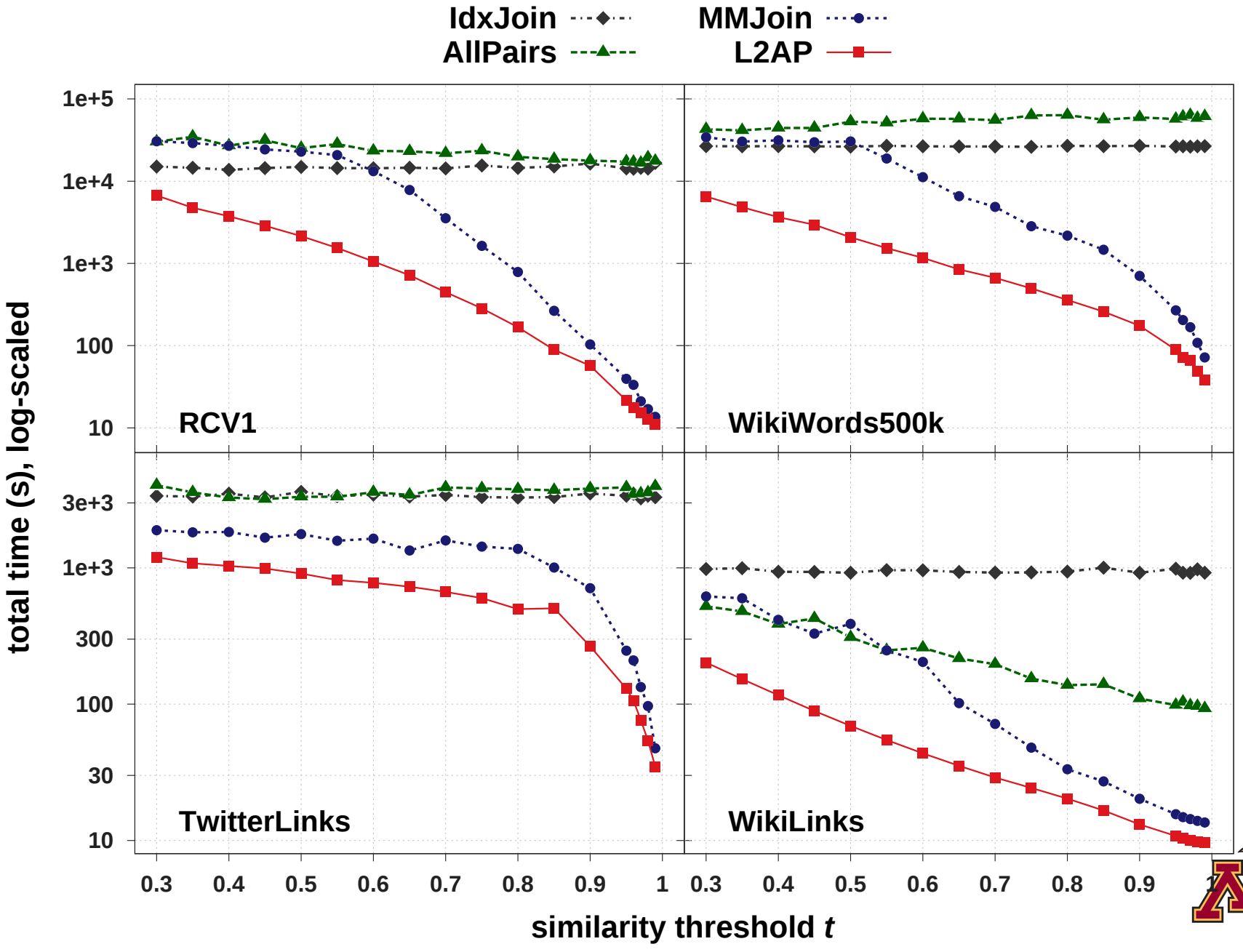
Dataset	n	m	nnz	$\frac{nnz}{n}$	$\frac{nnz}{m}$
RCV1	804,414	43,001	61e6	76	1417
WikiWords500k	494,244	343,622	197e6	399	574
WikiWords100k	100,528	339,944	79e6	787	233
TwitterLinks	146,170	143,469	200e6	1370	1395
WikiLinks	1,815,914	1,648,879	44e6	24	27
OrkutLinks	3,072,626	3,072,441	223e6	73	73

- **RCV1**: standard corpus of $> 800,000$ newswire stories.
- **WikiWords500k**: Wikipedia articles, min length 200.
- **WikiWords100k**: Wikipedia articles, min length 500.
- **TwitterLinks**: *follow* relationships of Twitter users that follow min 1,000 other users.
- **WikiLinks**: directed graph of hyperlinks between Wikipedia articles.
- **OrkutLinks**: Orkut friendship network.

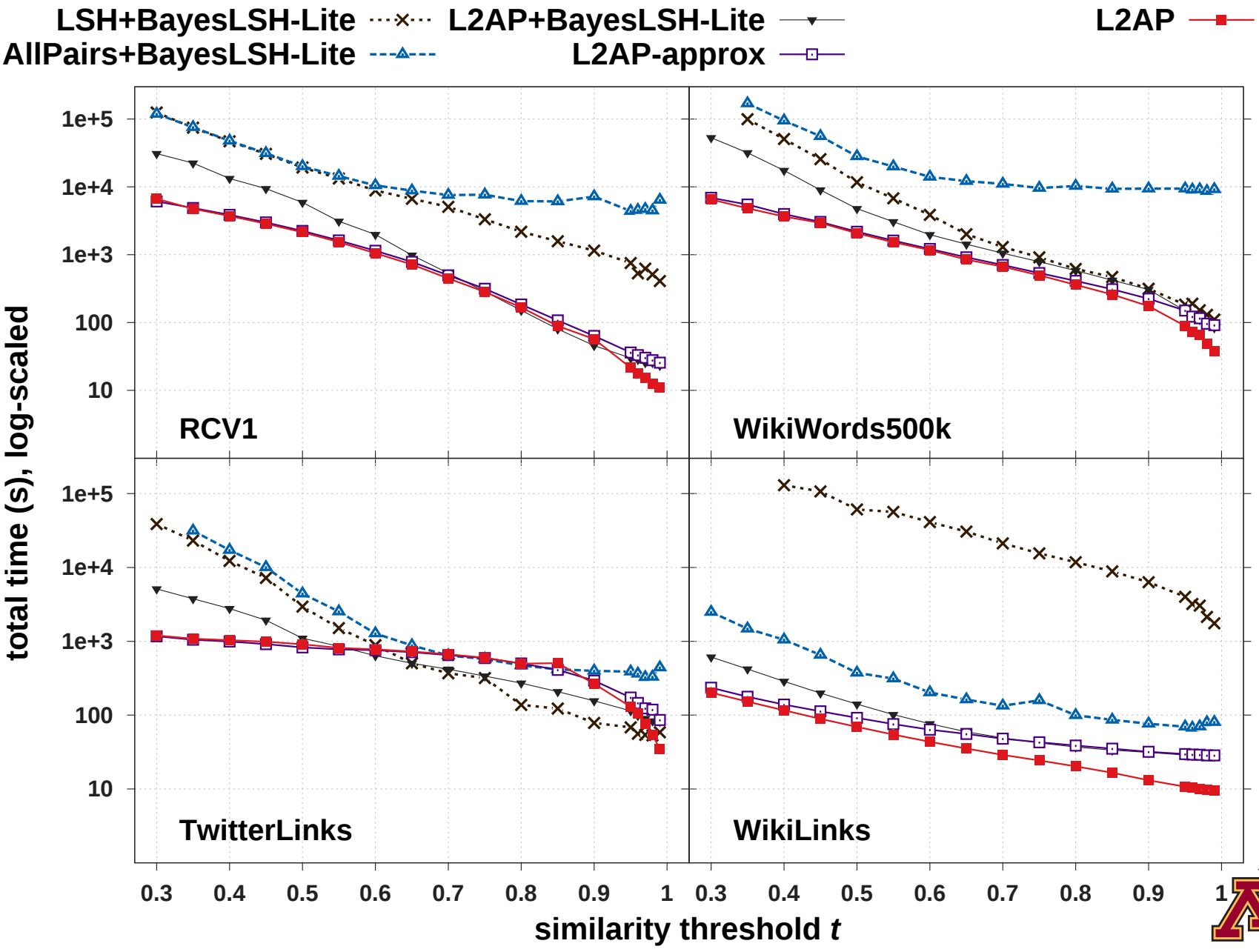
Baseline approaches

- `IdxJoin` builds an inverted index and uses it to find $\text{sim}(\mathbf{x}, \mathbf{y}), \forall y < x$, without pruning.
- `AllPairs` uses max vector $\mathbf{w}$ in similarity estimates (bounds b_1 and rs_1)
- `MMJoin` (Lee et al., 2010) enhances `AllPairs` by adding *length filtering* and a tighter `minsize` bound. *Length filtering* estimates $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \leq \frac{1}{2}||\mathbf{x}'_j||^2 + \frac{1}{2}||\mathbf{y}||^2 = \frac{1}{2}||\mathbf{x}'_j||^2 + \frac{1}{2}$, which is not as tight as our ℓ^2 norm estimate, $\text{sim}(\mathbf{x}'_j, \mathbf{y}) \leq ||\mathbf{x}'_j||$, especially for low t values.
- `AllPairs+BayesLSH-Lite` and `LSH+BayesLSH-Lite` are variants of `BayesLSH` that take as input the candidate set generated by `AllPairs` and `LSH`, respectively.
- Source code for all methods, including `L2AP` and `L2AP-approx`, available at `http://cs.umn.edu/~dragos/l2ap`.

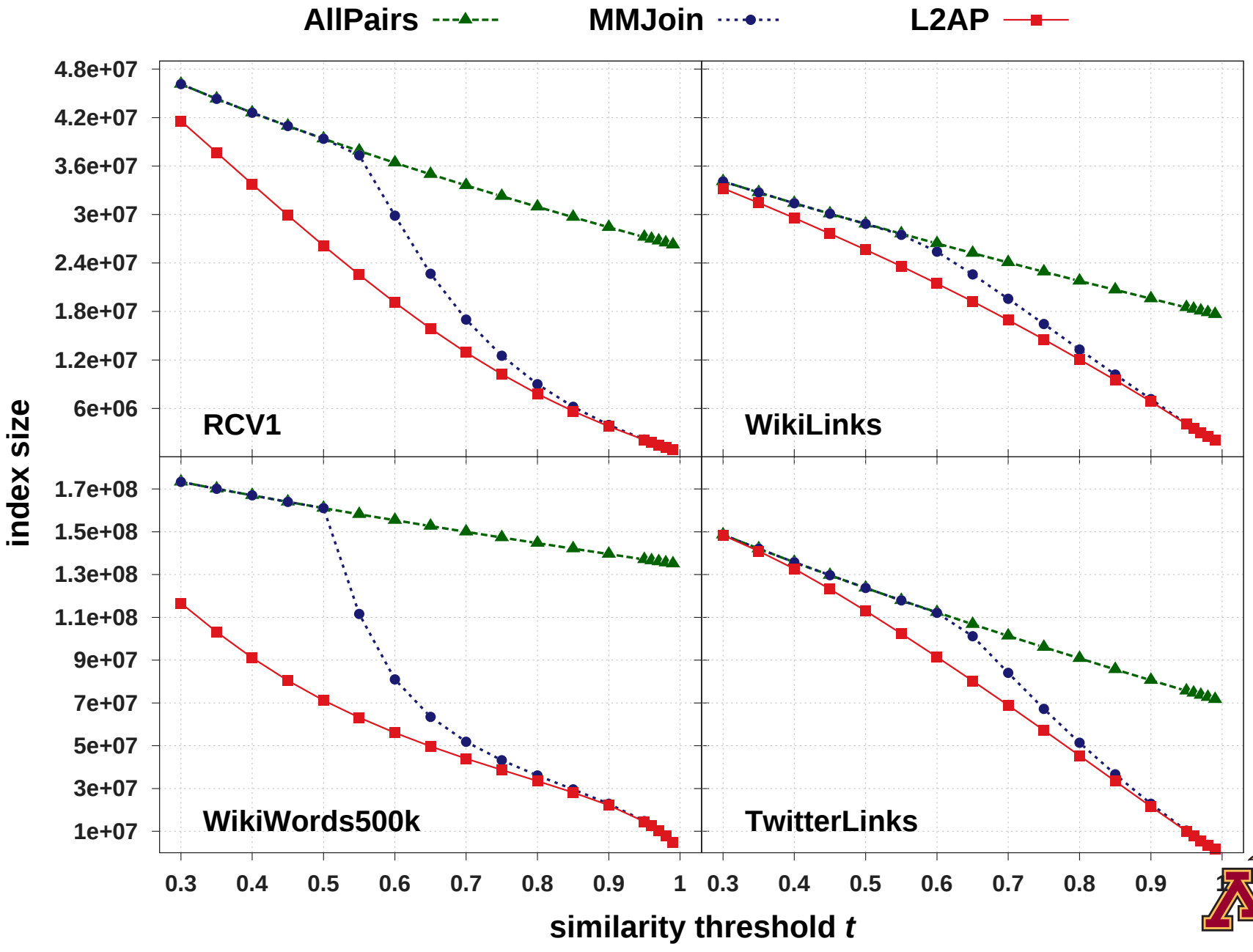
Comparison with exact baselines



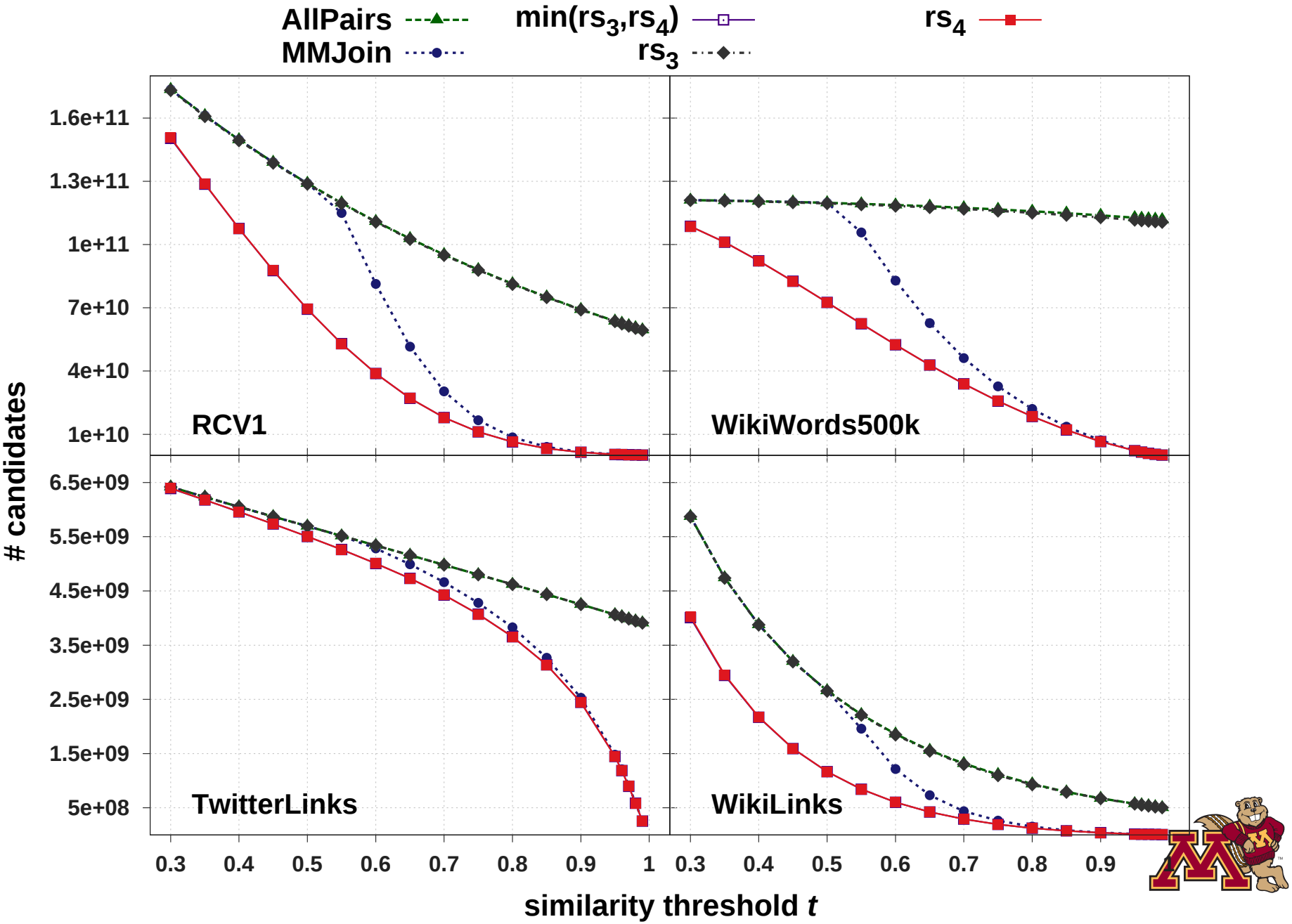
Comparison with approximate baselines



ℓ^2-norm effectiveness for index reduction



Residual filtering effectiveness



Conclusion

Lessons learned from `L2AP`

- Filtering is efficient
  - `L2AP` achieved significant speedups over exact baselines.
  - `BayesLSH-Lite` approximate pruning cannot significantly improve over `L2AP`.
- Filtering is effective
  - Improved *index*, *residual*, and *positional filtering* via ℓ^2-norm bounds.
  - Introduced *`pscore` filtering*, which is able to prune many generated candidates.
  - Strengthened other bounds, e.g. *`dpscore`*, detailed in the paper.

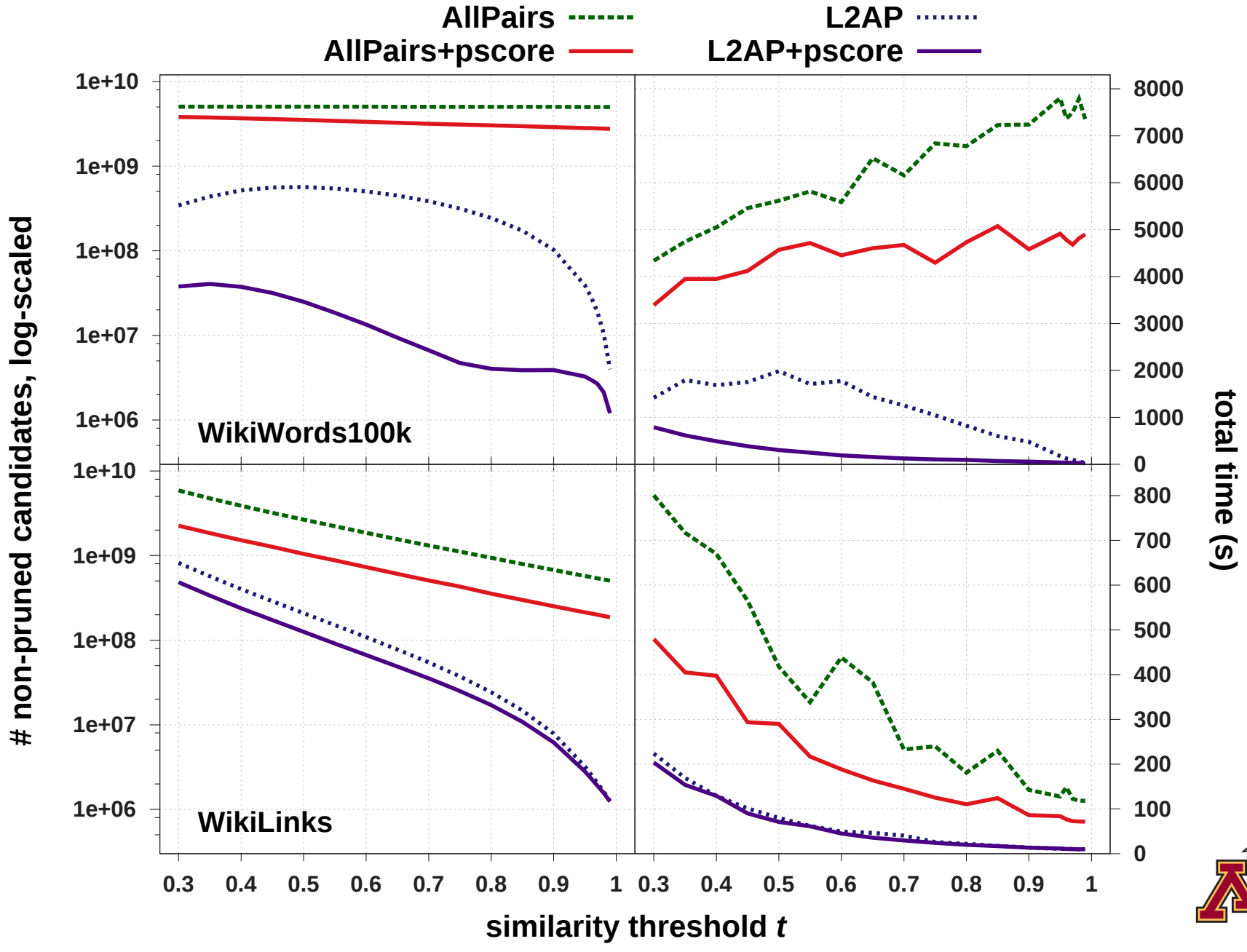
Thank You

- Questions?

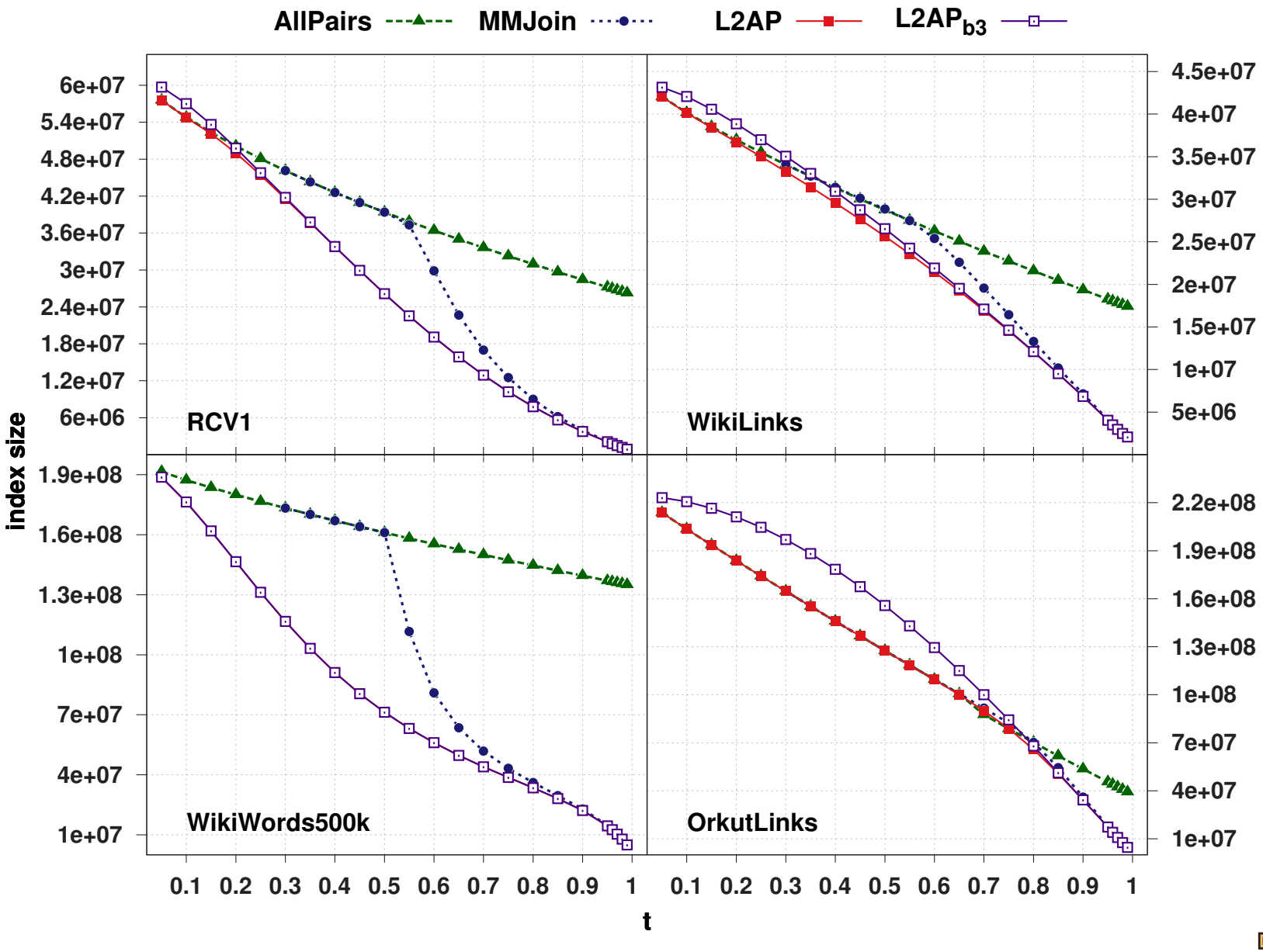
Acknowledgements:

This work was supported in part by the NSF (IOS-0820730, IIS-0905220, OCI-1048018, CNS-1162405, and IIS-1247632), the Digital Technology Center at the University of Minnesota, and Minnesota Supercomputing Institute.

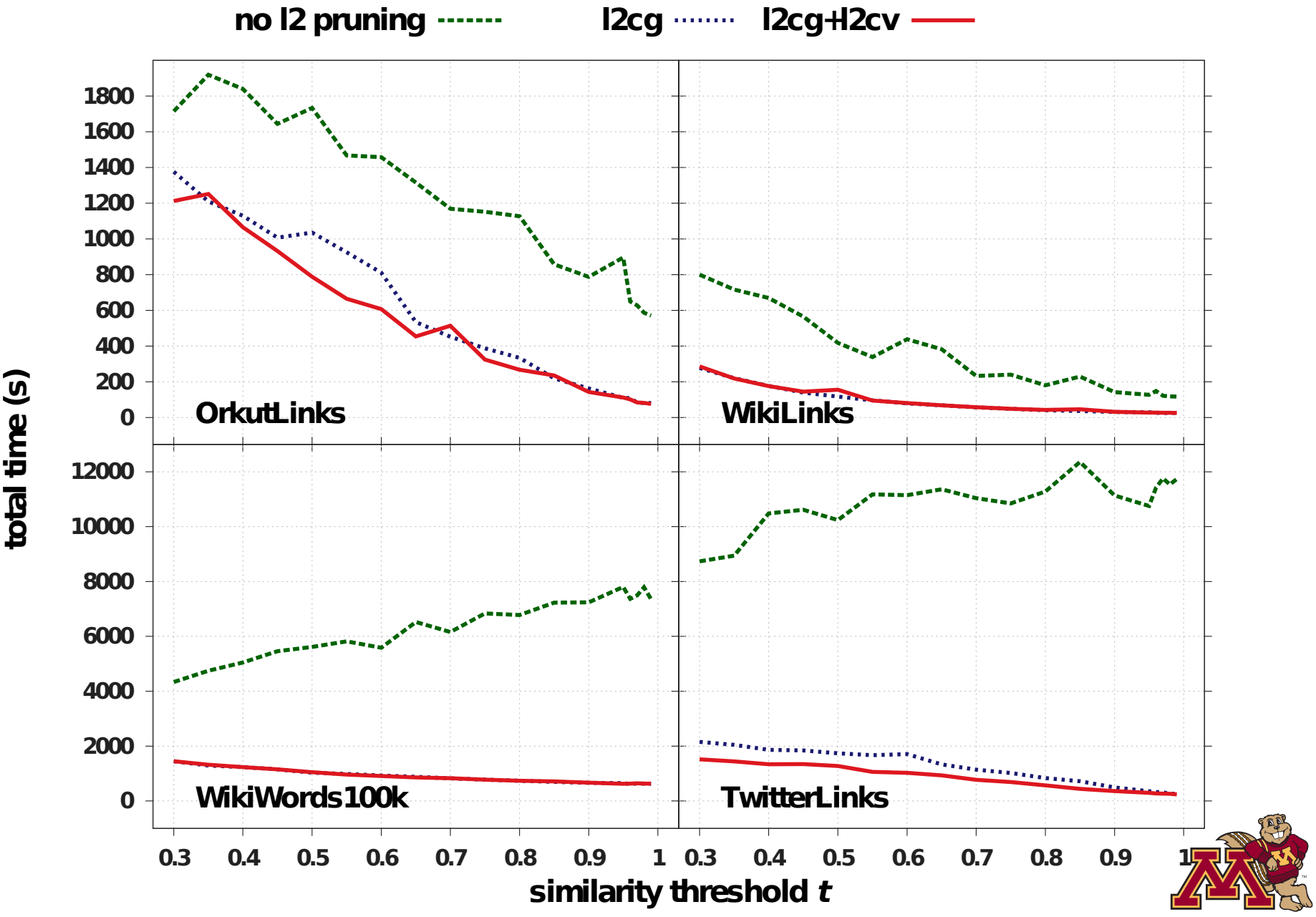
pscore effectiveness for candidate pruning



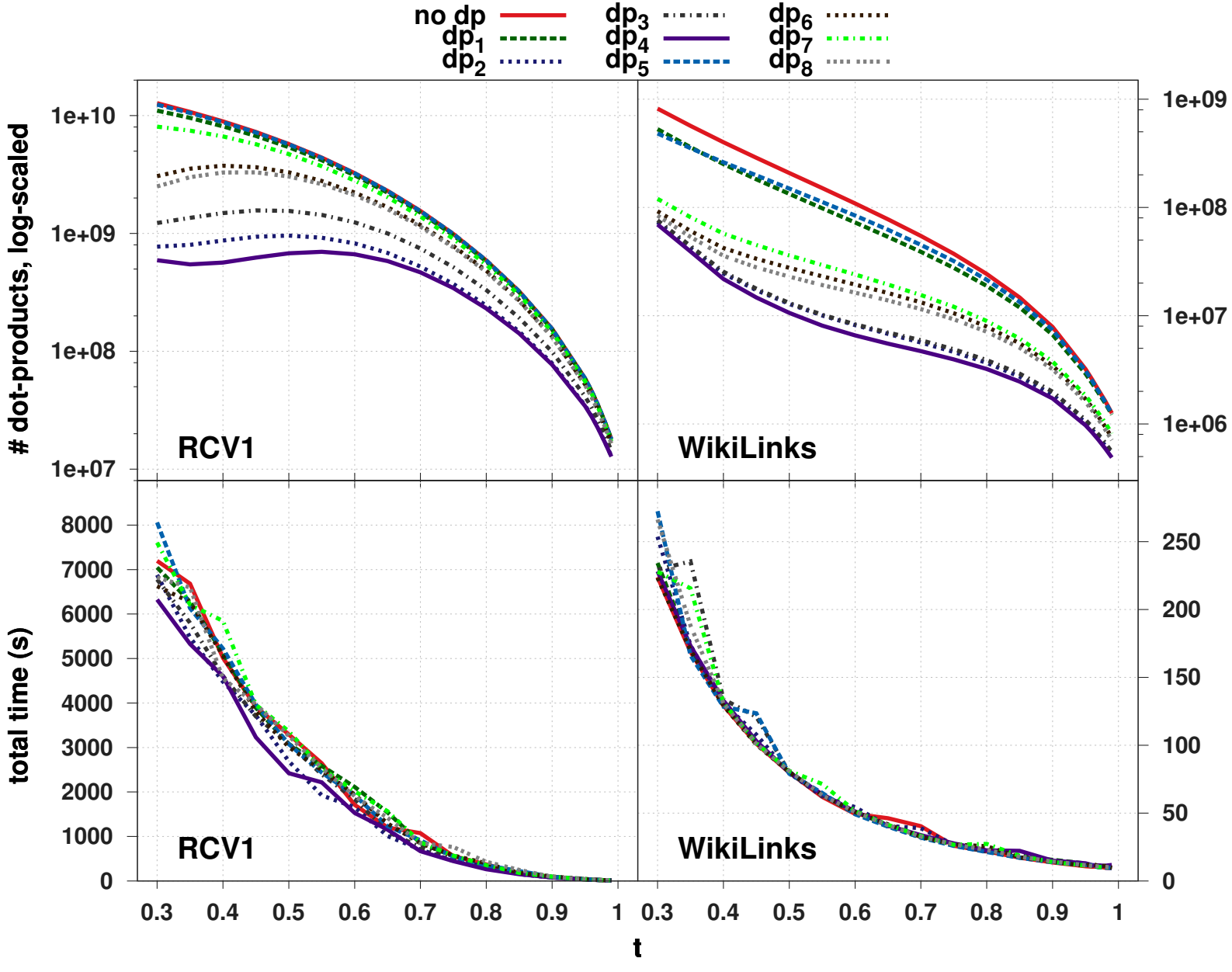
ℓ^2-norm only effectiveness for the `pscore` bound



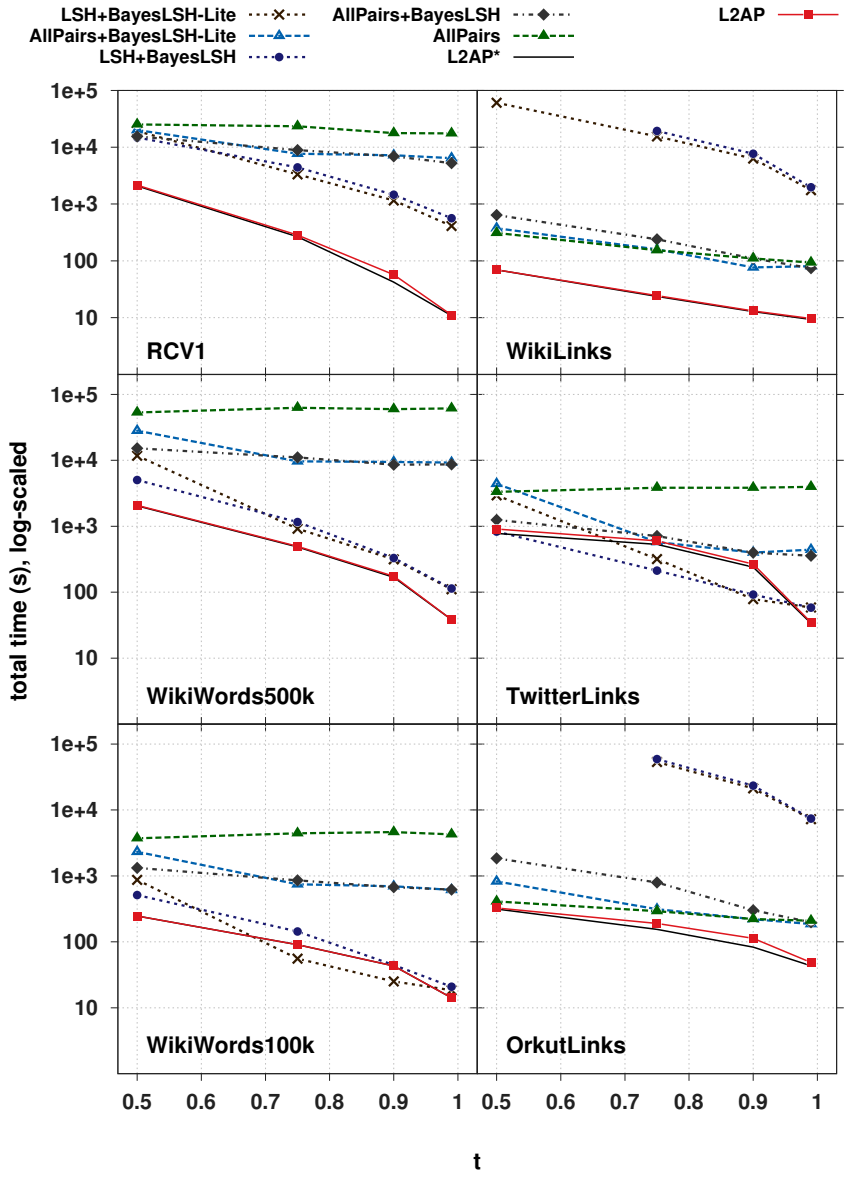
Effectiveness of new ℓ^2-norm filtering



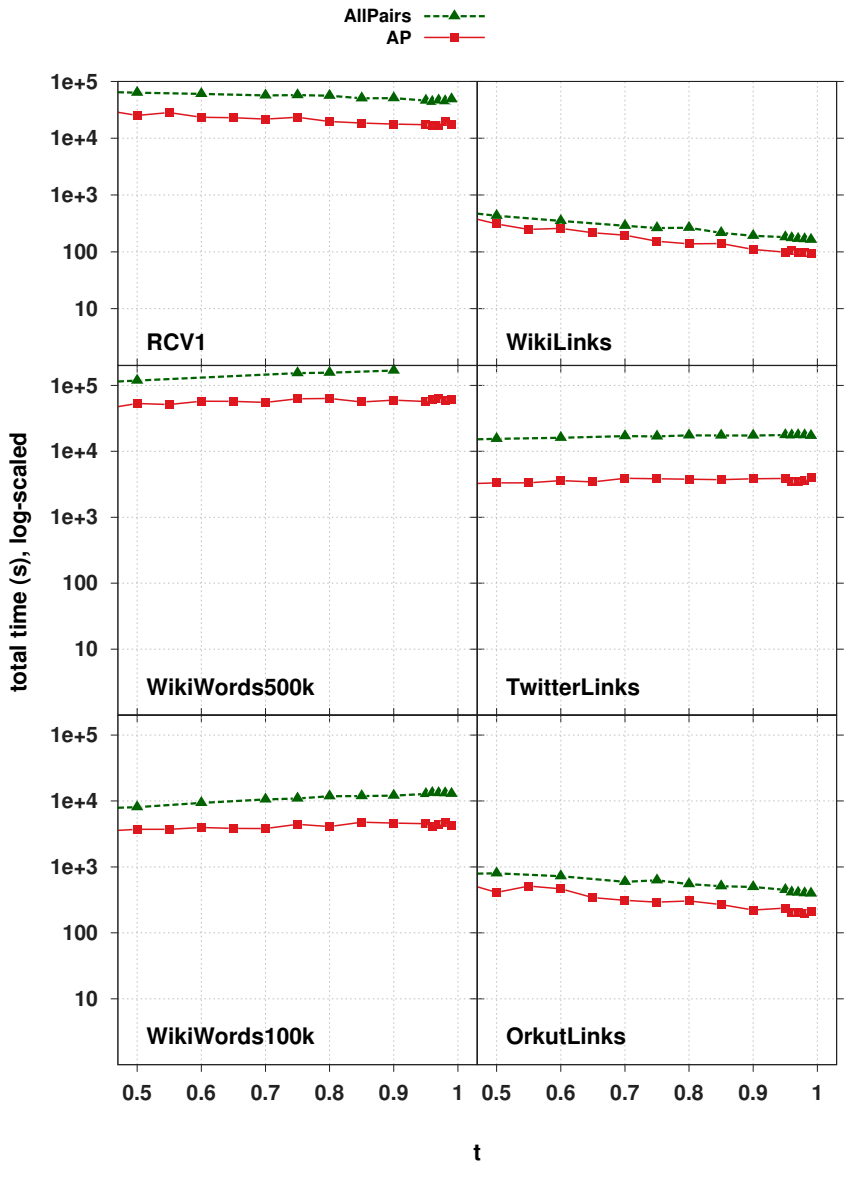
dpscore bounds effectiveness for positional filtering



Comparison with BayesLSH



Comparison of `AllPairs` implementations



Approximate extensions

- `BayesLSH-Lite` (Satuluri and Parthasarathy, 2012) finds the probability that $\text{sim}(\mathbf{x}, \mathbf{y}) > t$, conditional on observed LSH hash matches, after checking h hashes.
- We created two approximate APSS methods by combining `BayesLSH-Lite` with `L2AP`:
  - `L2AP`**+**`BayesLSH-Lite` - replace candidate verification with `BayesLSH-Lite`
  - `L2AP-approx` - replace only *l2cv* bound pruning with `BayesLSH-Lite`