Are You My Neighbor? Bringing Order to Neighbor Computing Problems.

David C. Anastasiu[1,2] , Huzefa Rangwala[3], and Andrea Tagarelli[4]

[1]Computer Engineering, San Jose State University, CA

[1]Computer Science & Engineering, Santa Clara University, CA

[2]Computer Science & Engineering, George Mason University, VA

[3]DIMES, University of Calabria, Italy

25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, August 2019

Part V:
Filtering-Based Search

David C. Anastasiu, San José State University [david.anastasiu@sjsu.edu]

Starting September:
Department of Computer Science and Engineering
Santa Clara University

Tutorial Outline

- **Part I: Problems and Data Types**
  - Dense, sparse, and asymmetric data
  - Bounded nearest neighbor search
  - Nearest neighbor graph construction
  - Classical approaches and limitations
- **Part II: Neighbors in Genomics, Proteomics, and Bioinformatics**
  - Mass spectrometry search
  - Microbiome analysis
- **Part III: Approximate Search**
  - Locality sensitive hashing variants
  - Permutation and graph-based search
  - Maximum inner product search
- **Part IV: Neighbors in Advertising and Recommender Systems**
  - Collaborative filtering at scale
  - Learning models based on the neighborhood structure
- **Part V: Filtering-Based Search**
  - Massive search space pruning by partial indexing
  - Effective proximity bounds and when they are most useful
- **Part VI: Neighbors in Learning and Mining Problems in Graph Data**
  - Neighborhood as cluster in a complex network system
  - Neighborhood as influence trigger set

Talk Outline

- What is filtering-based search?
- Massive search space pruning by partial indexing [and other pruning strategies]
    - So what in the world is partial indexing?
    - Search using a partial index
    - The case of k-NNG construction
    - Approximate methods could use filtering too
    - What if we used parallelism
    - When both length and angles matter
- Effective proximity bounds and when they are most useful
    - Not all pruning is created equal
    - When less is more
- Open questions

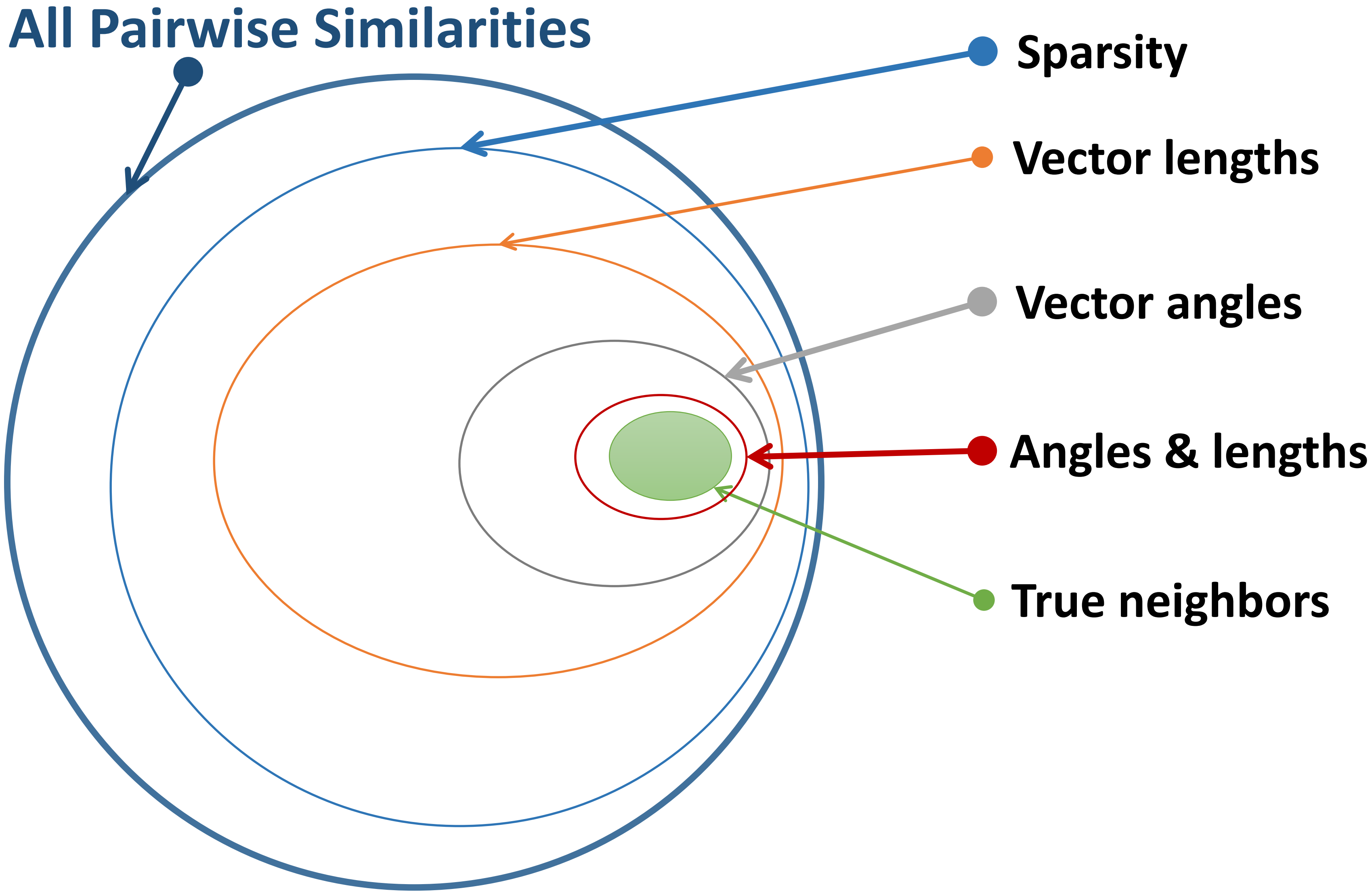
What is filtering-based search?

- Given a similarity bounding threshold, there is no need to compute the full similarity between a pair of vectors to tell if they are similar enough
  - Compute an upper bound similarity estimate
  - Prune/filter the pair if the similarity estimate is below the threshold

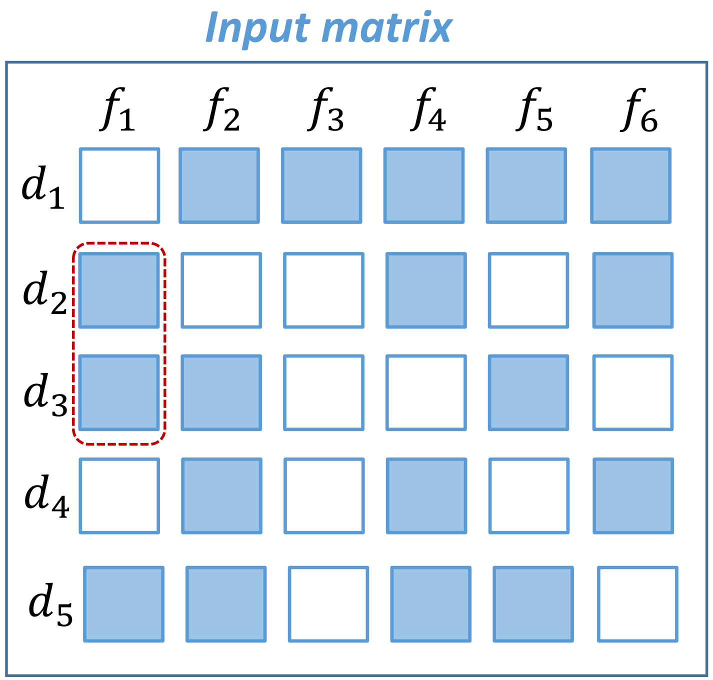
$$\widehat{\mathrm{sim}}(q,c) \geq \mathrm{sim}(q,c)$$

$$\begin{aligned} \textbf{if:}\ & \widehat{\mathrm{sim}}(q,c) < \epsilon \\ \textbf{then:}\ & \mathrm{sim}(q,c) < \epsilon \\ & \textbf{prune} \end{aligned}$$

How to prune the search space



Take advantage of sparsity



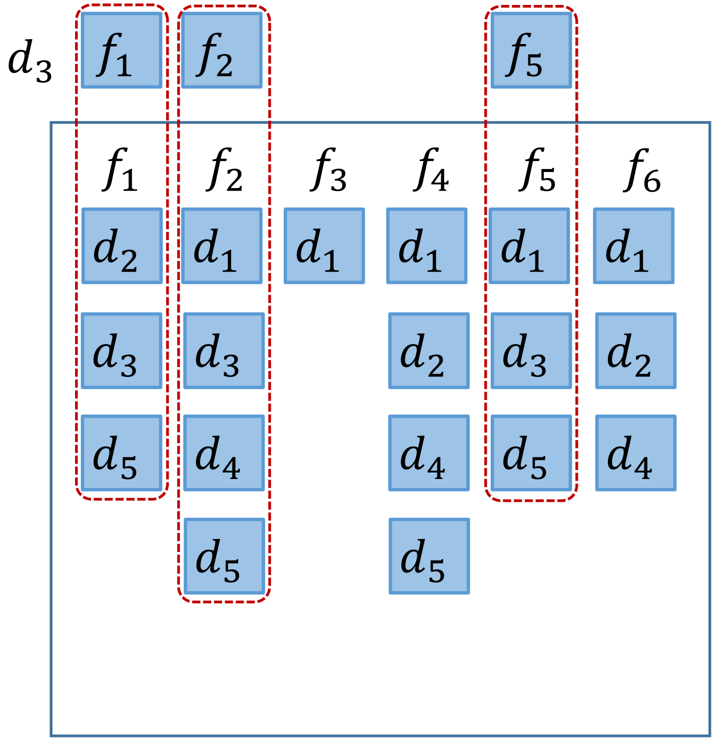
$$\mathrm{T}(d_i, d_j) = \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{||\mathbf{d}_i||_2^2 + ||\mathbf{d}_j||_2^2 - \langle \mathbf{d}_i, \mathbf{d}_j \rangle}$$

$$\mathrm{C}(d_i, d_j) = \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{||\mathbf{d}_i||_2 \times ||\mathbf{d}_j||_2}$$

$$\langle \mathbf{d}_2, \mathbf{d}_3 \rangle = \boxed{d_{2,1} \times d_{3,1}} + d_{2,2} \times d_{3,2} + d_{2,3} \times d_{3,3} + d_{2,4} \times d_{3,4} + d_{2,5} \times d_{3,5} + d_{2,6} \times d_{3,6}$$

Index and Accumulator: a match made in heaven

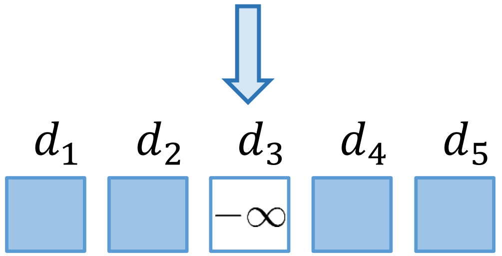
- Inverted index: set of lists, one for each feature, containing documents and their associated values



Inverted Index

$$A[d_2] \mathrel{+}= d_{3,1} \times d_{2,1}$$
$$A[d_5] \mathrel{+}= d_{3,1} \times d_{5,1}$$
$$A[d_1] \mathrel{+}= d_{3,2} \times d_{1,2}$$
$$A[d_4] \mathrel{+}= d_{3,2} \times d_{4,2}$$
$$A[d_5] \mathrel{+}= d_{3,2} \times d_{5,2}$$

[...]



Accumulator

IdxJoin: A straight-forward solution

- Method:

  Compute and store vector norms
  Construct an inverted index from the objects
  For each query object:
  - Compare only with objects with features in common
  - Select neighbors

- Results in *EXACT* solution
- Advantage:
  - Skips some object comparisons and many meaningless multiply-adds

Datasets

Dataset	n	m	nnz (M)	mrl	mcl	$type$
Patents	100,000	759,044	46.3	464	61	text
WW100k	100,528	339,944	79	787	233	text
Twitter	146,170	143,469	200	1370	1395	graph
WW500	243,223	660,600	202	830	306	text
MLSMR	325,164	20,021	56.1	173	2803	chem
WW500k	494,244	343,622	197	399	574	text
RCV1	804,414	43,001	61	76	1417	text
WW200	1,017,531	663,419	437	430	659	text
Wiki	1,815,914	1,648,879	44	24	27	graph
Orkut	3,072,626	3,072,441	223	73	73	graph
SC	11,519,370	7,415	1,784.5	155	262,669	chem

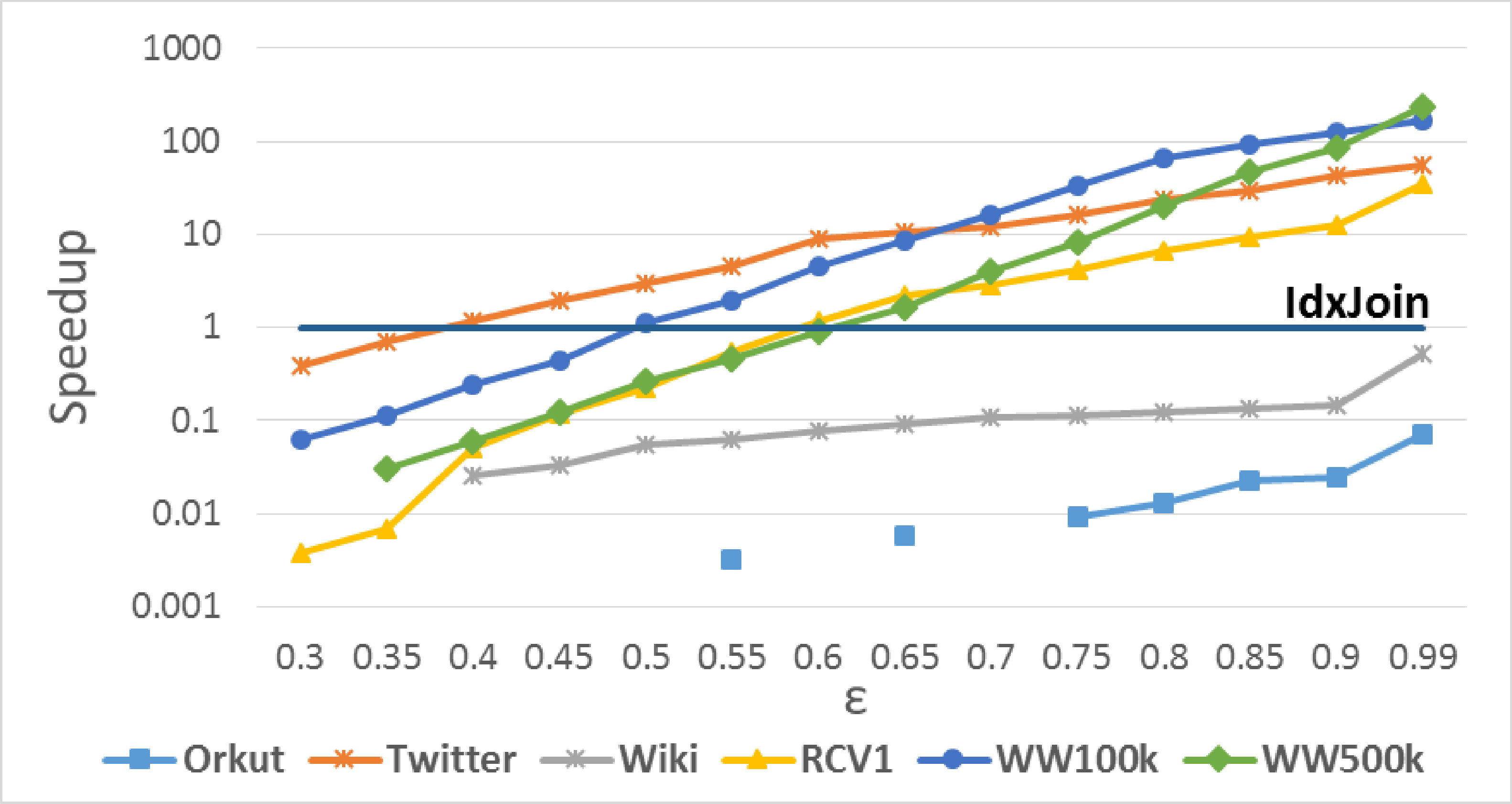
The sparsity advantage: IndexJoin vs. $n^2/2$

Dataset	# *sims*	% *all sims*
Orkut	23,308,569,172	0.12
Wiki	71,700,509,475	1.09
Twitter	8,516,651,351	19.93
RCV1	289,612,531,857	22.38
WW100k	5,052,763,937	25.00
WW500k	122,095,368,297	25.00

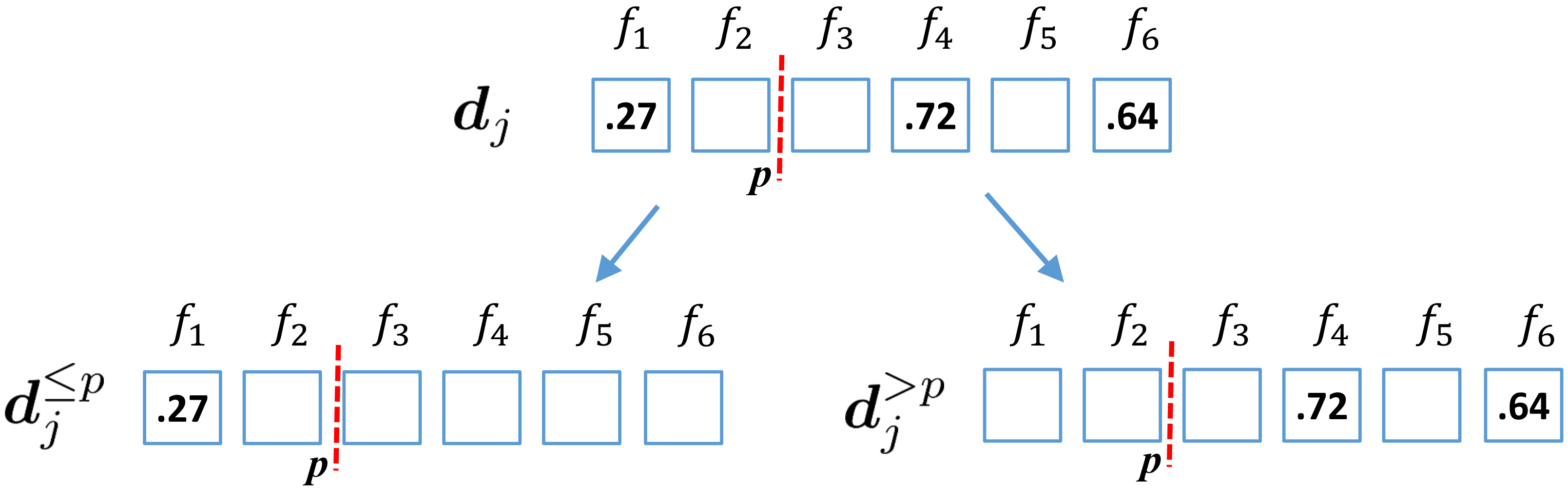
- A large number of comparisons are filtered by the sparsity constraints
- For Orkut and Wiki, 99% or more of the comparisons are simply ignored

LSH vs. IndexJoin

- In all experiments, LSH parameters were tuned to achieve at least 95% accuracy.
- LSH outperforms IndexJoin at high thresholds.
- Performs poorly at low thresholds and for high dimensional datasets (Orkut, Wiki).



A prelude: prefix and suffix vectors



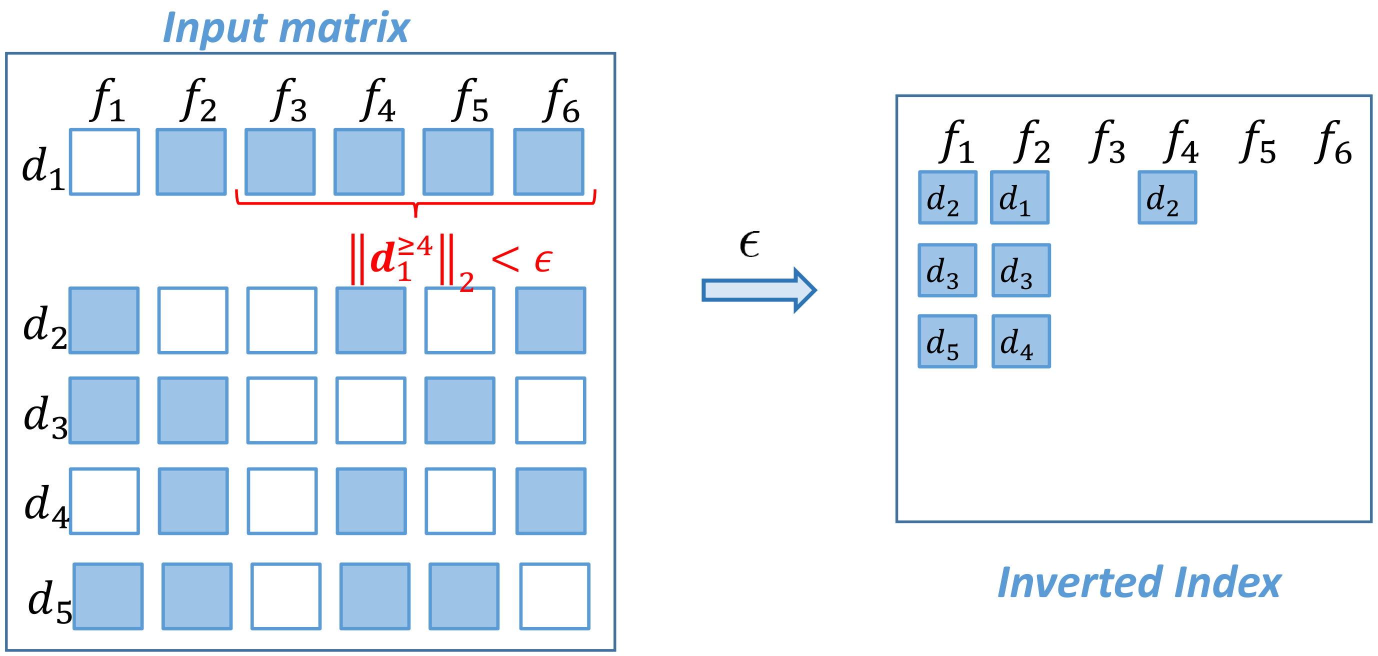
$$
\begin{aligned}
\boldsymbol{d}_i &= \boldsymbol{d}_i^{\leq p} + \boldsymbol{d}_i^{>p} \\
||\boldsymbol{d}_i||_0 &= ||\boldsymbol{d}_i^{\leq p}||_0 + ||\boldsymbol{d}_i^{>p}||_0 \\
||\boldsymbol{d}_i||_2^2 &= ||\boldsymbol{d}_i^{\leq p}||_2^2 + ||\boldsymbol{d}_i^{>p}||_2^2 \\
\langle \boldsymbol{d}_i, \boldsymbol{d}_j \rangle &= \langle \boldsymbol{d}_i, \boldsymbol{d}_j^{\leq p} \rangle + \langle \boldsymbol{d}_i, \boldsymbol{d}_j^{>p} \rangle \\
\langle \boldsymbol{d}_i, \boldsymbol{d}_j \rangle &= \langle \boldsymbol{d}_i^{\leq p}, \boldsymbol{d}_j^{\leq p} \rangle + \langle \boldsymbol{d}_i^{>p}, \boldsymbol{d}_j^{>p} \rangle
\end{aligned}
$$

So what in the world is partial indexing?

Main idea:

Only need to index enough non-zeros to guarantee correct result.

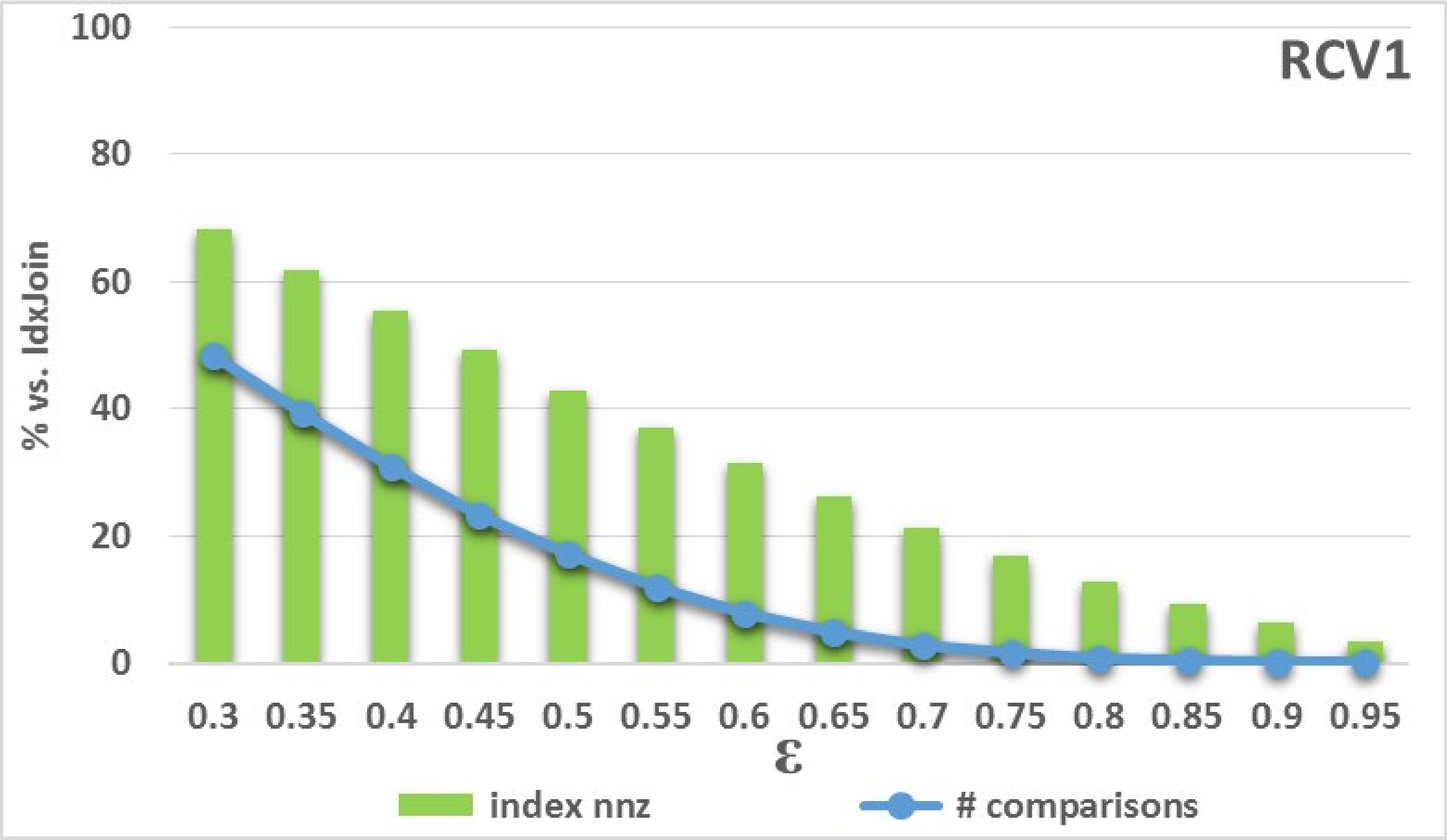
$$\begin{aligned} \mathrm{sim}(d_1^{\leq}, d_c) &\leq \|\mathbf{d_1^{\leq}}\|_2 \times \|\mathbf{d_c}\|_2 \\ &< \|\mathbf{d_1^{\leq}}\|_2 \times 1 \\ &< \epsilon \end{aligned}$$



- We'll focus initially on the min-ϵ graph construction problem.

Partial indexing in practice

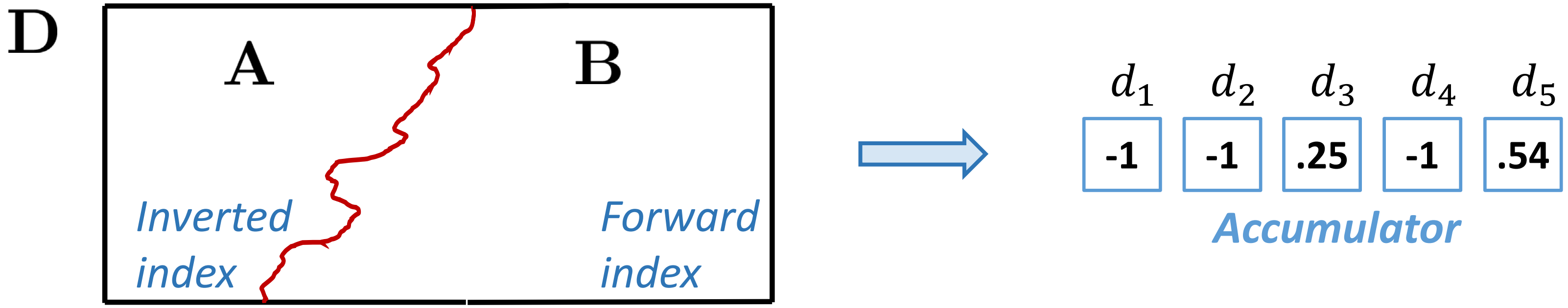
- L2AP indexes fewer non-zeros than previous approaches
- Leads to greatly improved execution runtime



Search using a partial index

L2AP follows a two-step process:

1. Accumulate similarity using partial inverted index



$$\mathbf{D}\mathbf{D}^T = \mathbf{D}\mathbf{A}^T + \mathbf{D}\mathbf{B}^T$$

2. For each un-pruned object, finish similarity computation using forward index
   - Only need to compute a subset of similarities: $\mathrm{sim}(d_q, d_3^{\geq}) \ \& \ \mathrm{sim}(d_q, d_5^{\geq})$
   - Can do further filtering

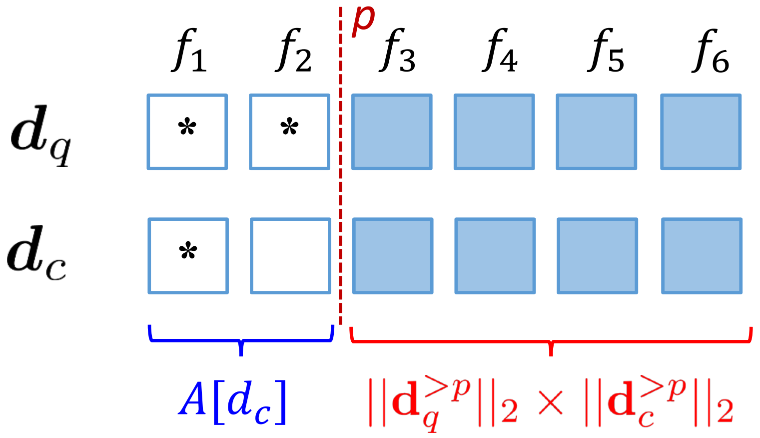
http://davidanastasiu.net/software/l2ap/

Angle/Suffix Filtering

Filter/prune object pairs not in final graph based on *similarity estimates*

$$\langle \mathbf{d}_q^{>p}, \mathbf{d}_c^{>p} \rangle \leq ||\mathbf{d}_q^{>p}||_2 \times ||\mathbf{d}_c^{>p}||_2$$

(Cauchy-Schwarz inequality)

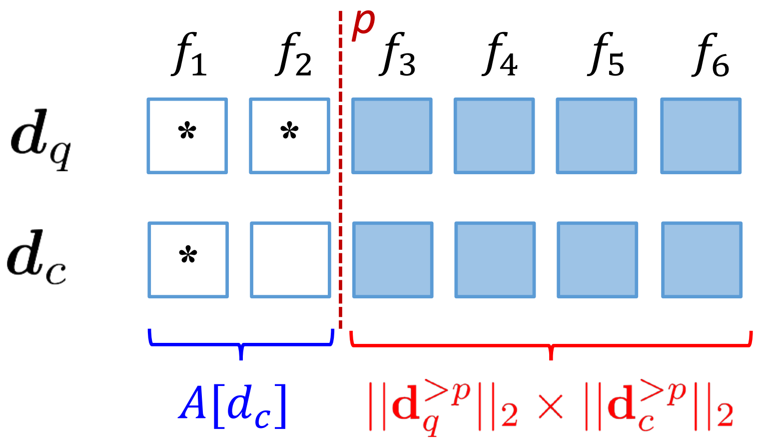


Filter $\mathrm{sim}(d_q, d_c)$ if $A[d_c] + ||\mathbf{d}_q^{>p}||_2 \times ||\mathbf{d}_c^{>p}||_2 < \epsilon$

Angle/Suffix Filtering

Filter/prune object pairs not in final graph based on *similarity estimates*

$$\langle \mathbf{d}_q^{>p}, \mathbf{d}_c^{>p} \rangle \leq \min(||\mathbf{d}_q^{>p}||_0, ||\mathbf{d}_c^{>p}||_0) \times ||\mathbf{d}_q^{>p}||_\infty \times ||\mathbf{d}_c^{>p}||_\infty$$



Filter $\text{sim}(d_q, d_c)$ if

$$A[d_c] + \min(||\mathbf{d}_q^{>p}||_0, ||\mathbf{d}_c^{>p}||_0) \times ||\mathbf{d}_q^{>p}||_\infty \times ||\mathbf{d}_c^{>p}||_\infty > \epsilon$$

But won't it take longer to compute those norms?

- We pre-compute all necessary norms and store them in a compressed sparse row (CSR) –like data structure
  - O(nnz) time and space for this step

	.75	.25	.25	.50	.25
.27			.72		.64
.72	.49			.49	
	.67		.33		.67
.65	.44		.44	.44	

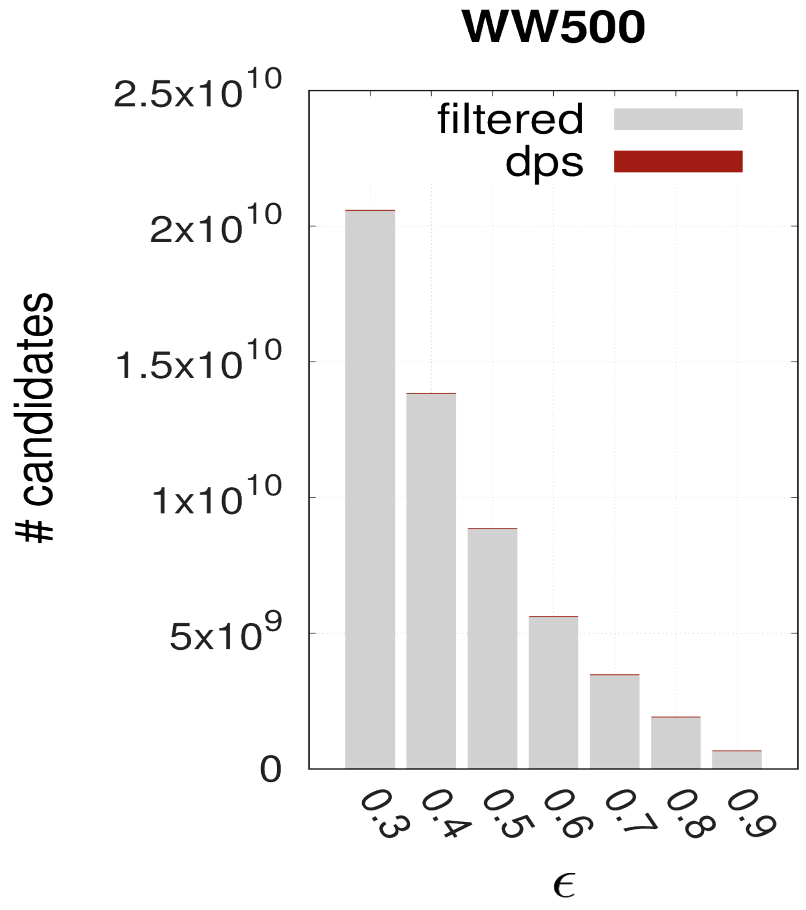
rowval	.75	.25	.25	.50	.25	.27	.72	.64	.72	.49	.49	.67	.33	.67	.65	.44	.44	.44
l2norm	.66	.61	.56	.25	.00	.96	.64	.00	.69	.49	.00	.75	.67	.00	.76	.62	.44	.00
rowind	1	2	3	4	5	0	3	5	0	1	4	1	3	5	0	1	3	4
rowptr	0	5	8	11	14	18												

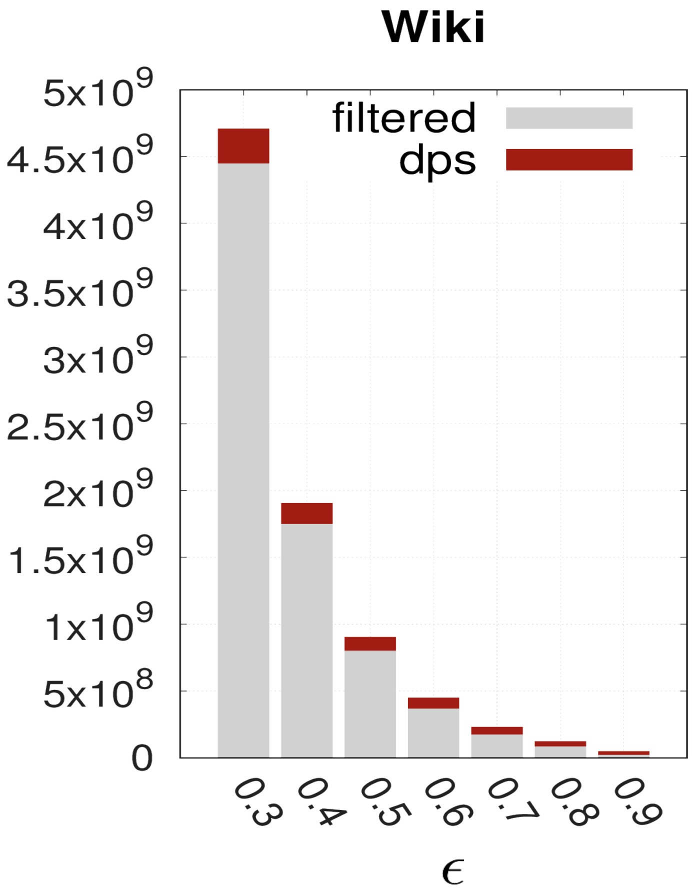
Same idea for

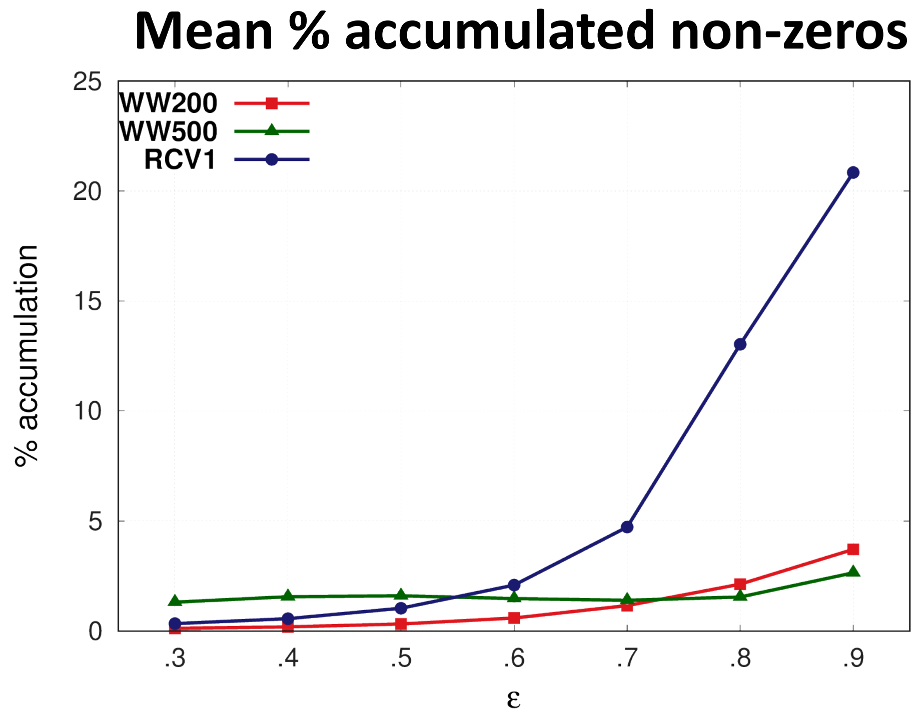
$$||\mathbf{d}^{>p}||_0, \text{ or } ||\mathbf{d}^{>p}||_\infty$$

Filtering in practice

- L2AP filters most objects without computing their similarity

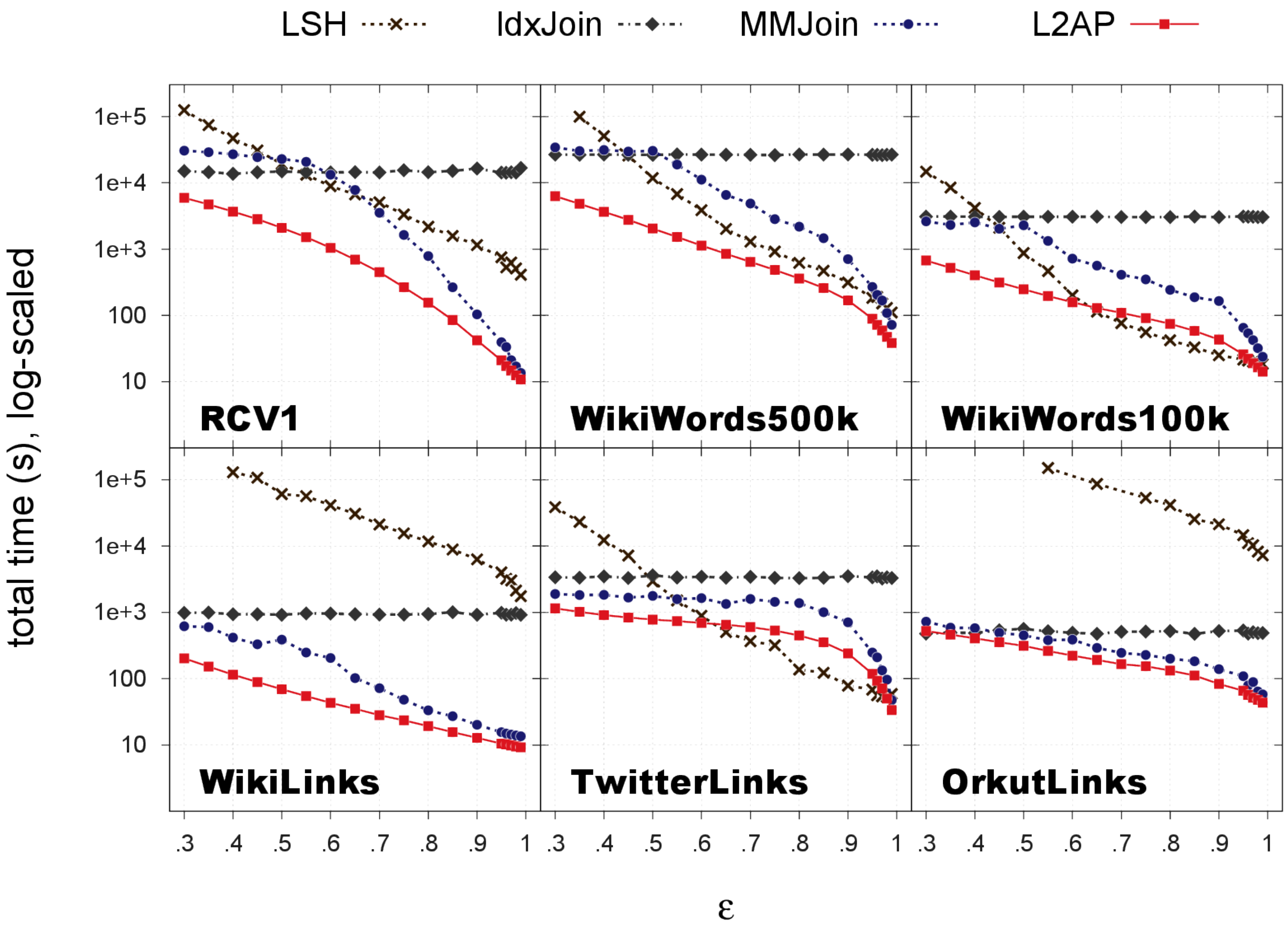






How fast is it?

- L2AP outperforms all exact and most approximate baselines



The case of k-NNG construction

- In the k-NN problem, we don't have a nice global minimum similarity threshold we can use for filtering
  - It exists, but
    - (1) we don't know it
    - (2) it is likely too low to make a difference
- We can use local incomplete (approximate) neighborhood thresholds
- L2Knng strategy:
  - Build an initial approximate graph $\hat{G}$
    - Provides thresholds for filtering
  - Improve $\hat{G}$ until exact
    - Search for objects that can improve neighborhoods

http://davidanastasiu.net/software/l2knng/

Step 1: Approximate graph construction

a) Heuristically choose candidates likely to succeed

- Find an initial neighborhood for each object

f_1	d_3 **.72**	d_5 **.65**	d_2 **.27**	
f_2	d_1 **.75**	d_4 **.67**	d_5 **.44**	d_3 **.49**
f_3	d_1 **.25**			
f_4	d_2 **.72**	d_5 **.44**	d_4 **.34**	d_1 **.25**
f_5	d_1 **.50**	d_3 **.49**	d_5 **.44**	
f_6	d_4 **.67**	d_2 **.64**	d_1 **.25**	

d_3	f_1 **.72**	f_2 **.49**	f_5 **.49**

- Sort vectors and inverted lists in non-increasing weight order
- Traverse inverted lists and gather $\mu \geq k$ candidates

$$C_{\mu=3} = [d_5, d_1, d_4]$$

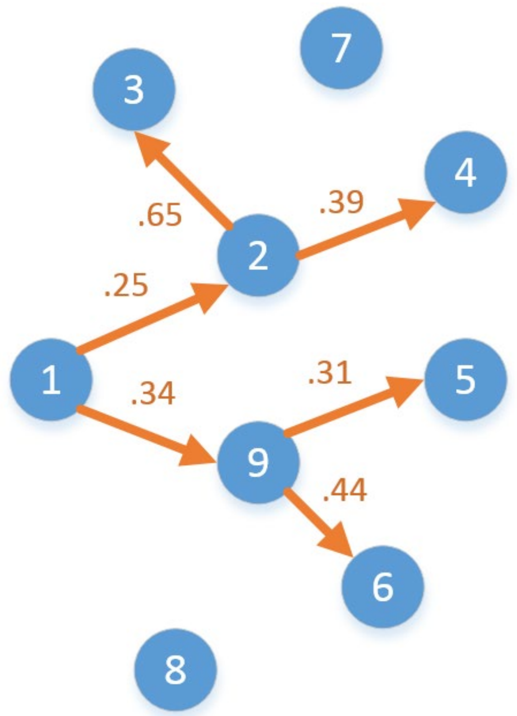
- Compute similarities with candidates and keep the k nearest neighbors

- This step results in an initial approximate graph

Step 1: Approximate graph construction

b) Improve initial approximate graph

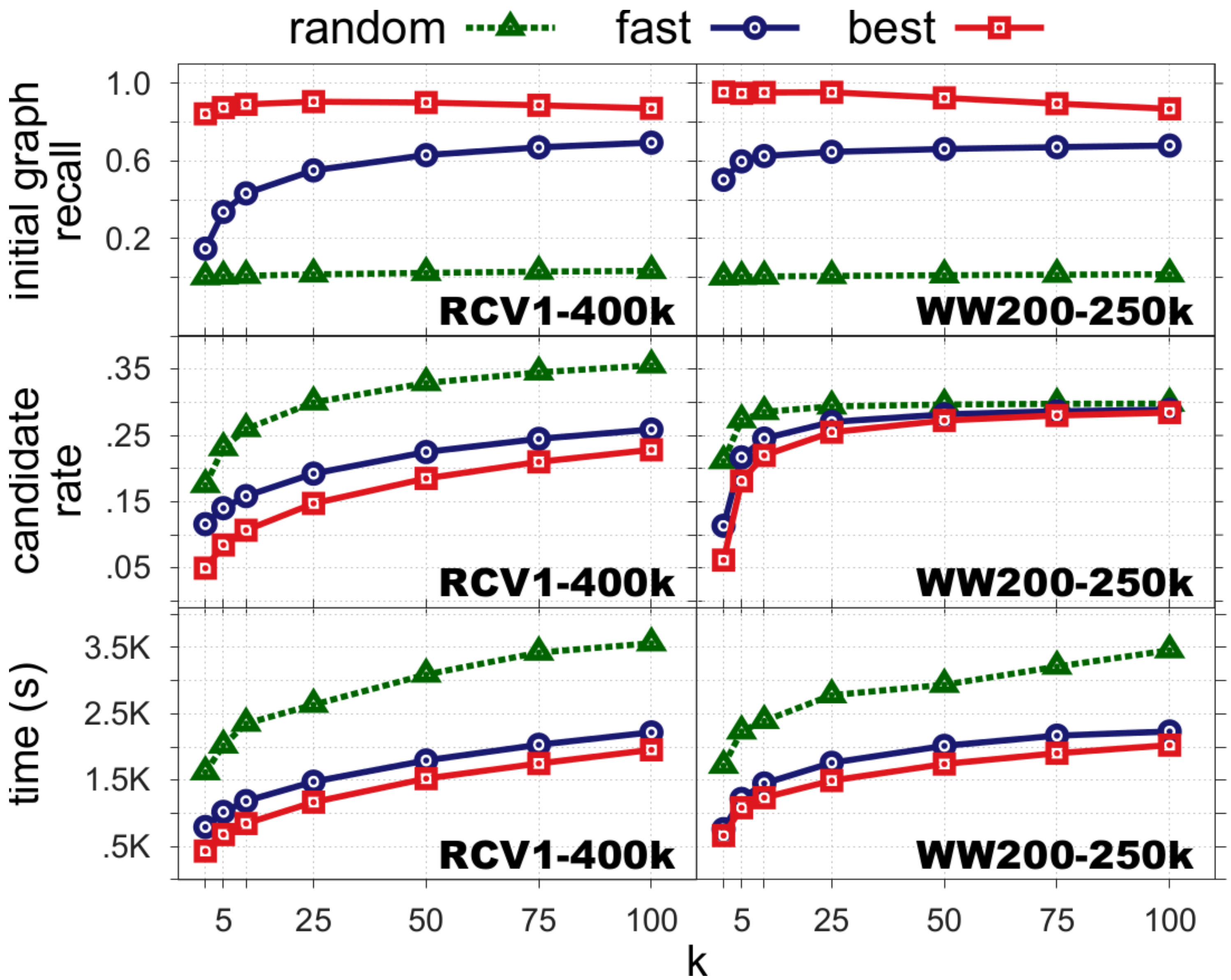
- Find potential better neighbors for each object (γ times)
  - Visit neighbors in non-increasing similarity order
  - Consider d_s, a neighbor of d_r, as a candidate if:
    - Have collected less than μ candidates
    - $sim(d_s, d_r) \geq sim(d_r, d_q)$
  - Compute similarities with candidates and update both neighborhoods of d_s and d_q



$$C_{\mu=3}(d_1) = [d_6, d_3, d_4]$$

How important is Step 1?

Influence of initial graph quality toward exact graph construction



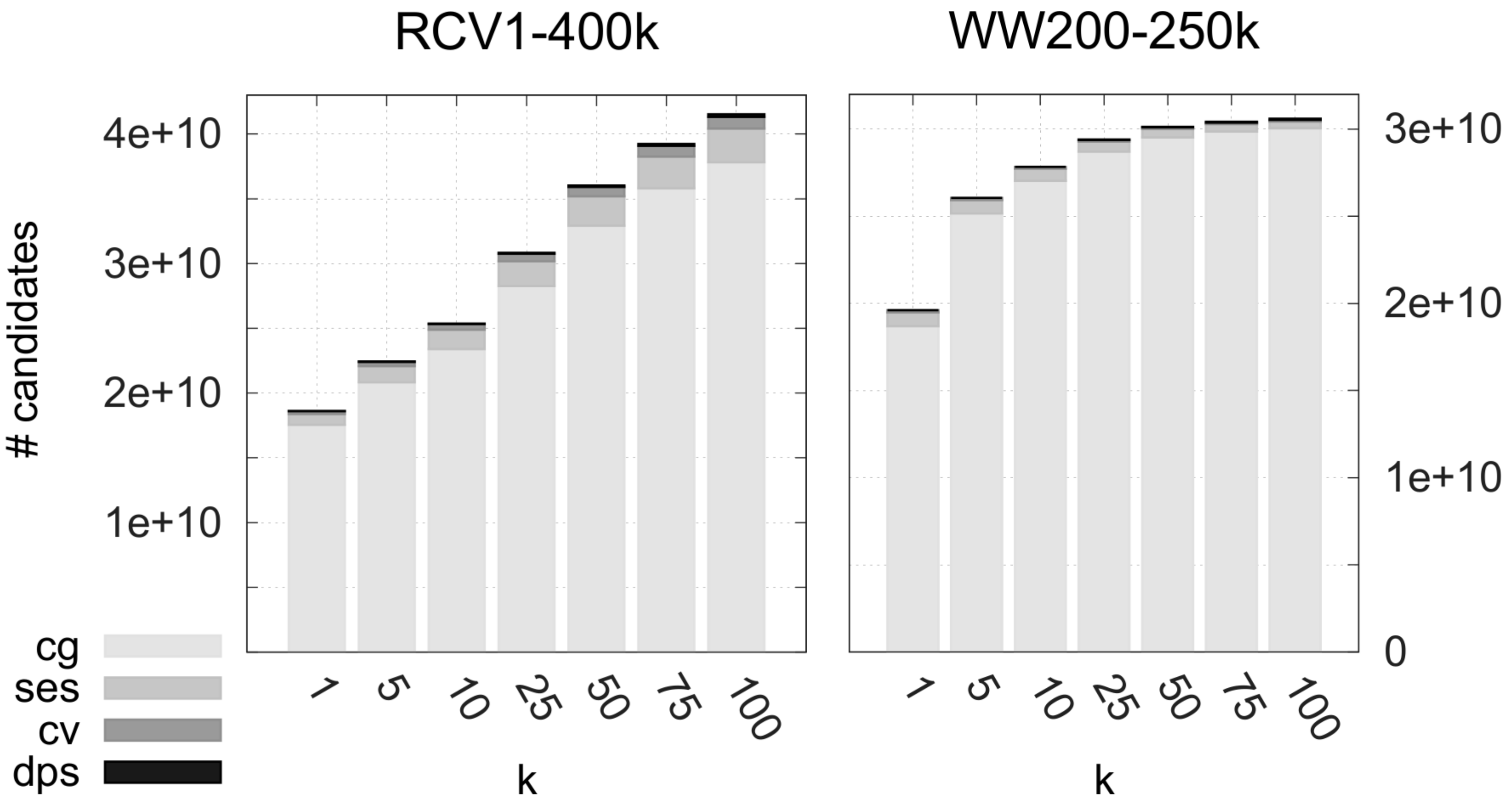
Step 2: Filtering

- For each *query object* d_q,
  - Find previously processed objects such that:
    - $sim(d_c, d_q) > \sigma_{d_q}$: can improve query obj. neighborhood
    - $sim(d_c, d_q) > \sigma_{d_c}$: can improve neighborhood of previously processed objects
  - Verify list of candidate objects:
    - prune object pair as soon as possible
    - else update neighborhoods of both objects
  - Index processed query object

- Caveats
  - Neighborhoods stored in max-heaps
  - Index tiling used to improve neighborhoods quicker

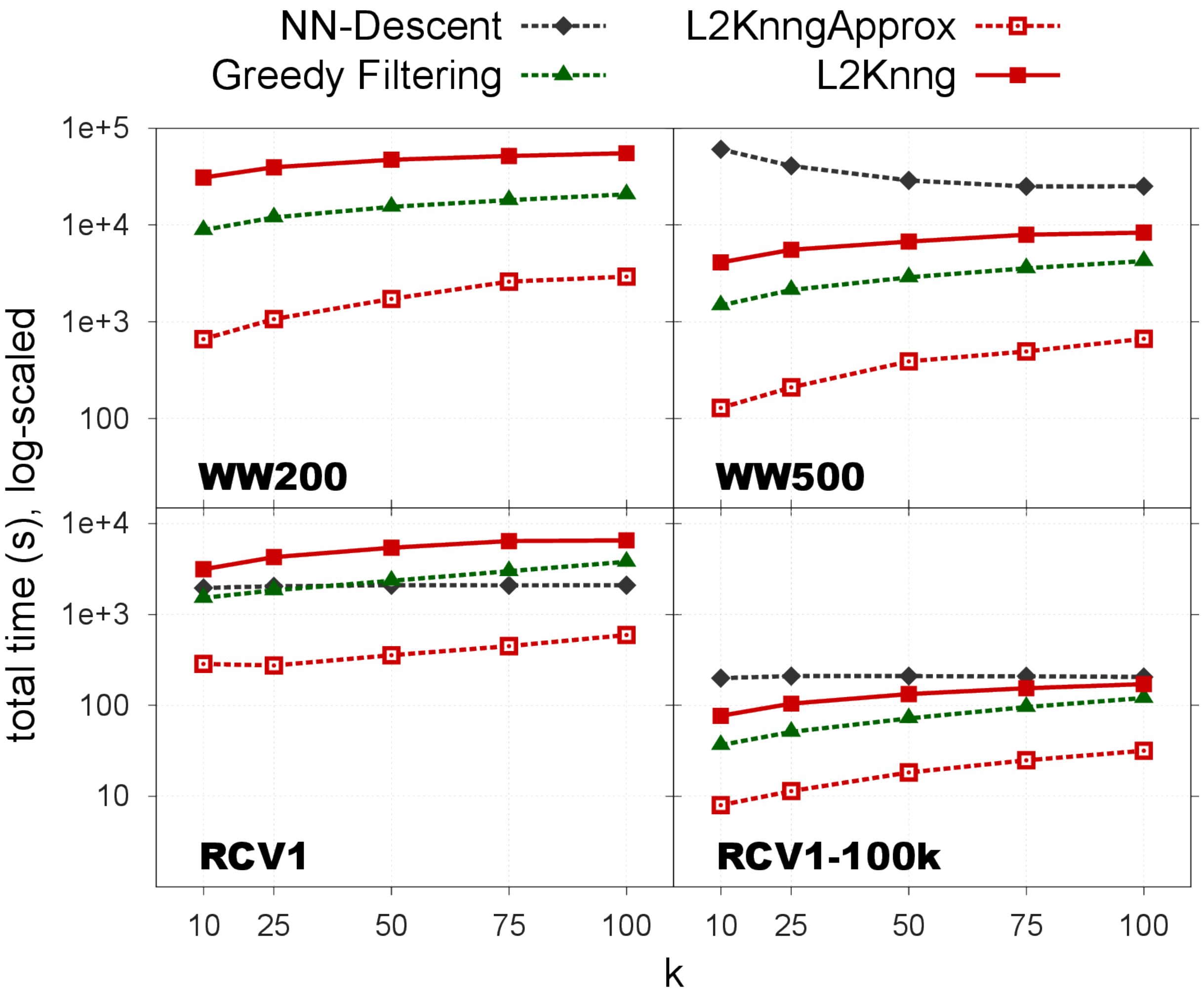
How well does the filtering work?

Number of candidates pruned in different stages of the filtering framework

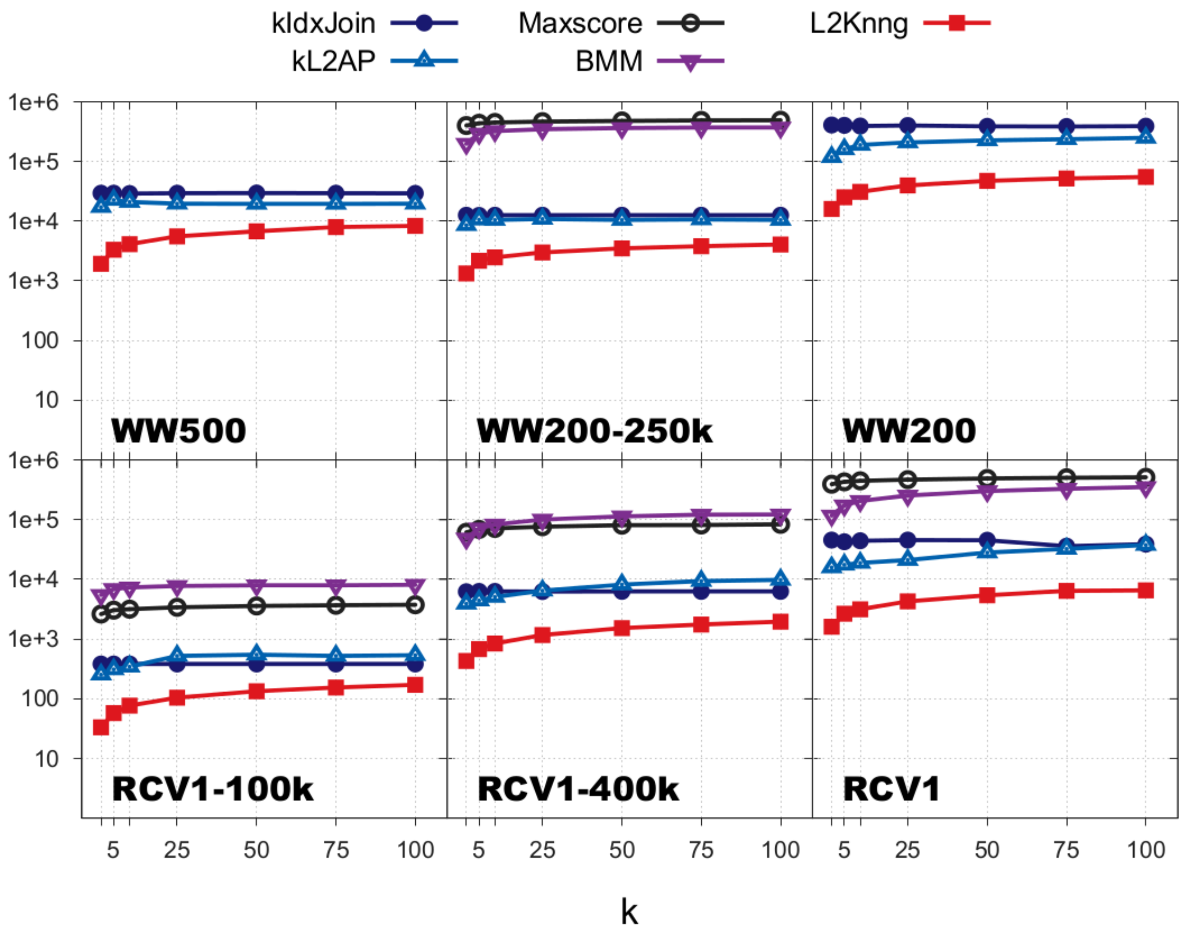


How fast is it?

Approximate Baselines



Exact Baselines



Approximate methods could use filtering too

- CANN applies the ideas in L2AP and L2Knng-a to the approximate min-ϵ graph construction problem

- Approximate solution, in 2 steps:
  1. Construct approximate min-ϵ k-NN graph $\mathcal{G}$
     1) Heuristically choose objects that area likely neighbors:
        a. Build partial inverted index for min-ϵ search
        b. Sort query vectors and partial inverted index in decreasing order
        c. Choose objects with high weights in common
  2. Use $\mathcal{G}$ to construct final min-ϵ NN graph
     1) Zero or more graph improvement steps
        a. Chose candidates among the neighbors of my neighbors that have higher similarity with my neighbor than me and my neighbor do

Filtering helps improve efficiency

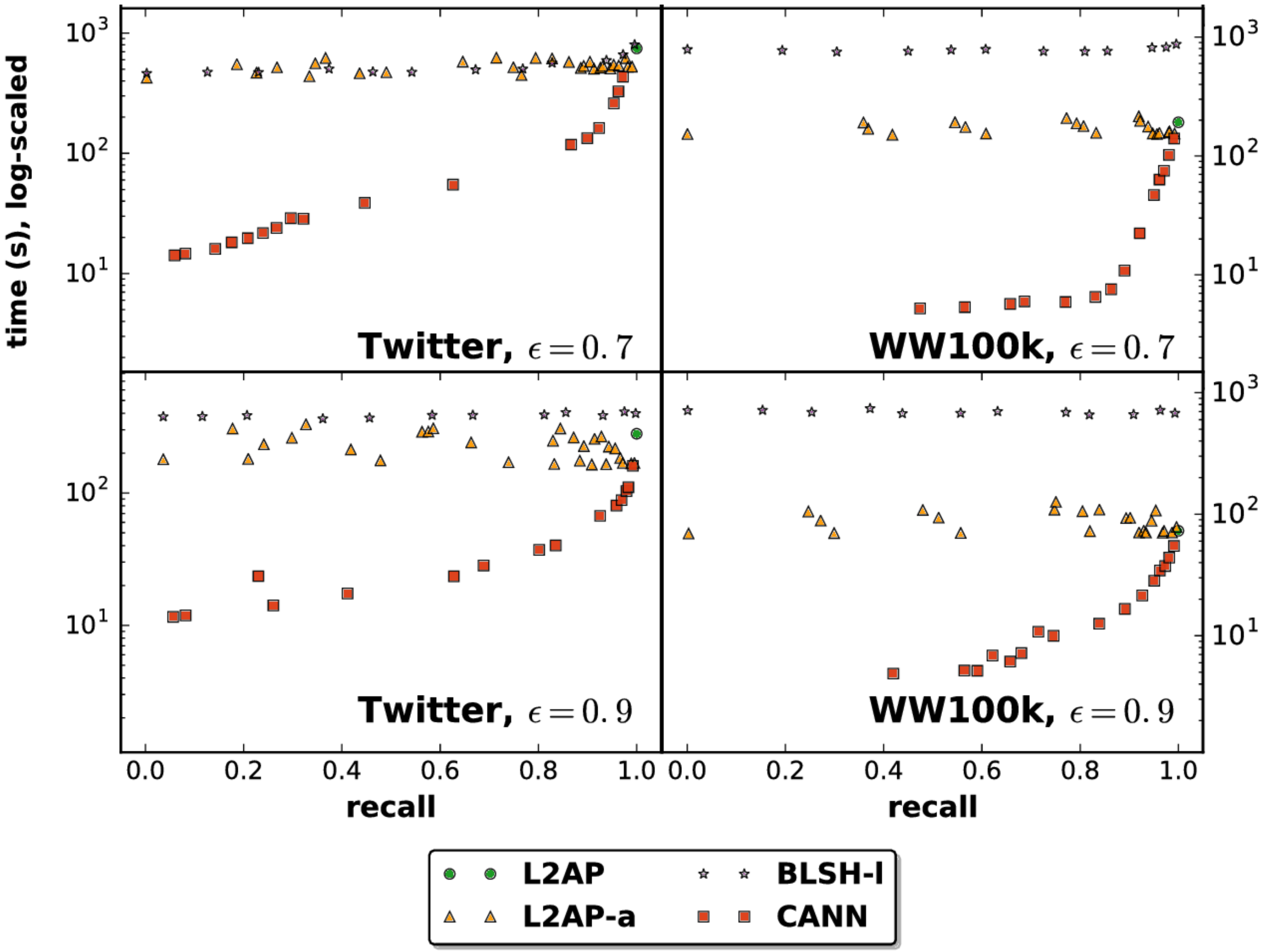
- L2-Norm bound is most useful (ignore others)
- Helpful to hash the query vector (e.g., make it dense)

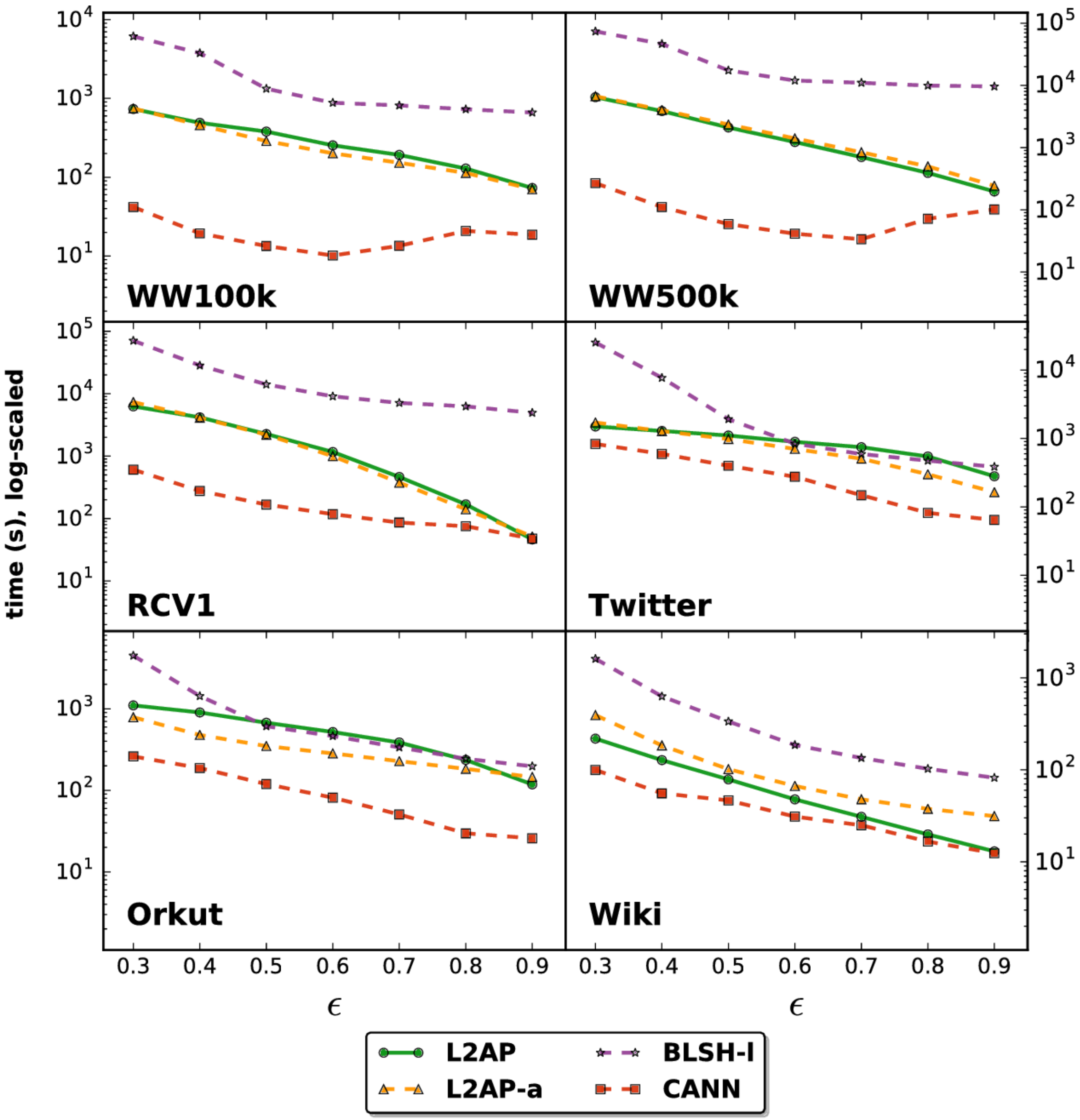
Bounded similarity computation with pruning:

```
function BOUNDEDSIM(d_q, d_c, ε)
    s ← 0
    for each j = 1, ..., m s.t. d_{c,j} > 0 do
        if d_{q,j} > 0 then
            s ← s + d_{q,j} × d_{c,j}
            if s + ||d_q^{>j}|| × ||d_c^{>j}|| < ε then
                return -1
return s
```

$$\langle \mathbf{d}_q, \mathbf{d}_c \rangle = \underbrace{\langle \mathbf{d}_q^{\le p}, \mathbf{d}_c^{\le p} \rangle}_{\text{compute}} + \underbrace{\langle \mathbf{d}_q^{>p}, \mathbf{d}_c^{>p} \rangle}_{\text{estimate}}$$

How fast is it?





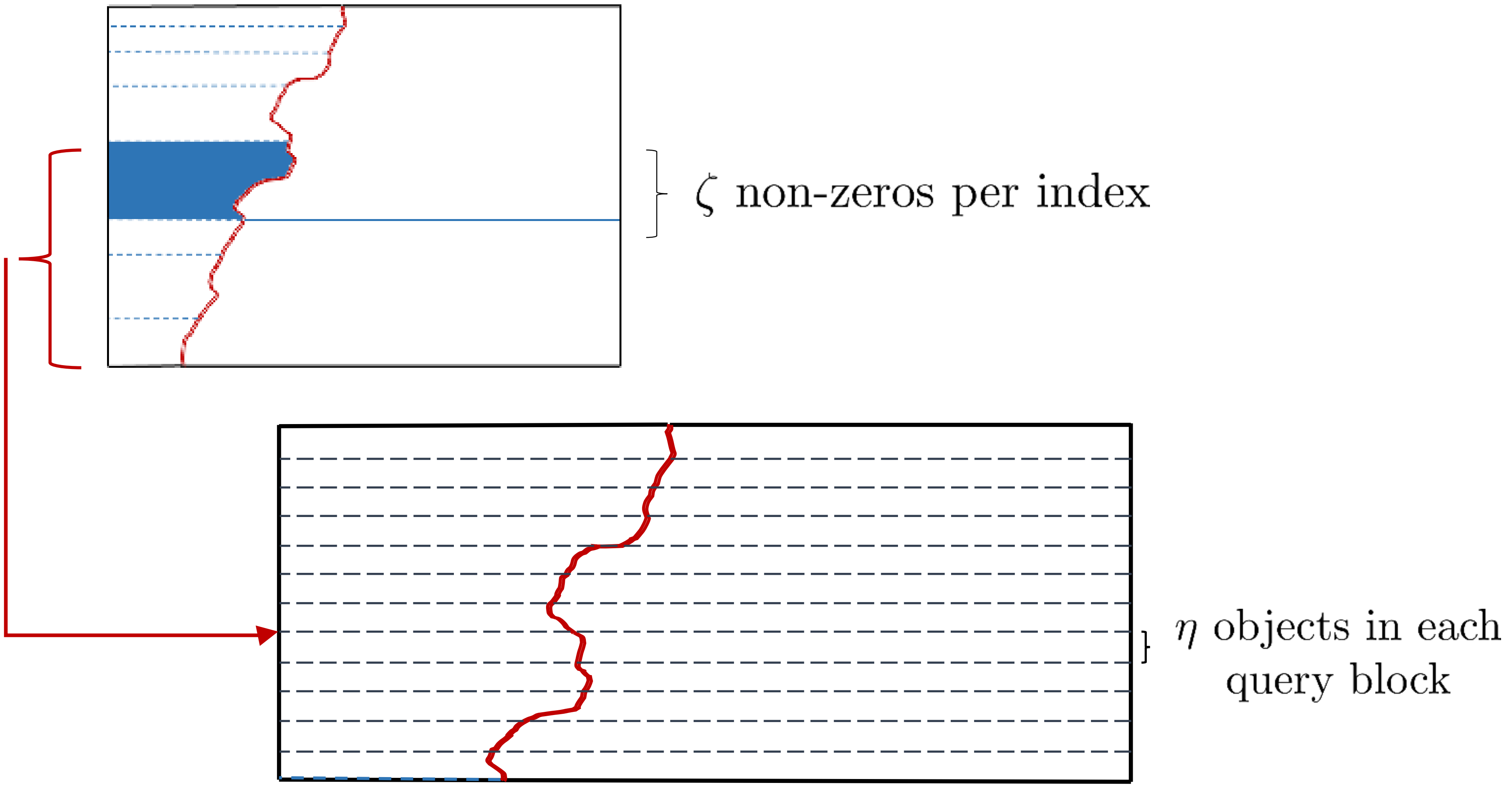
Recall = 0.9

What if we used parallelism?

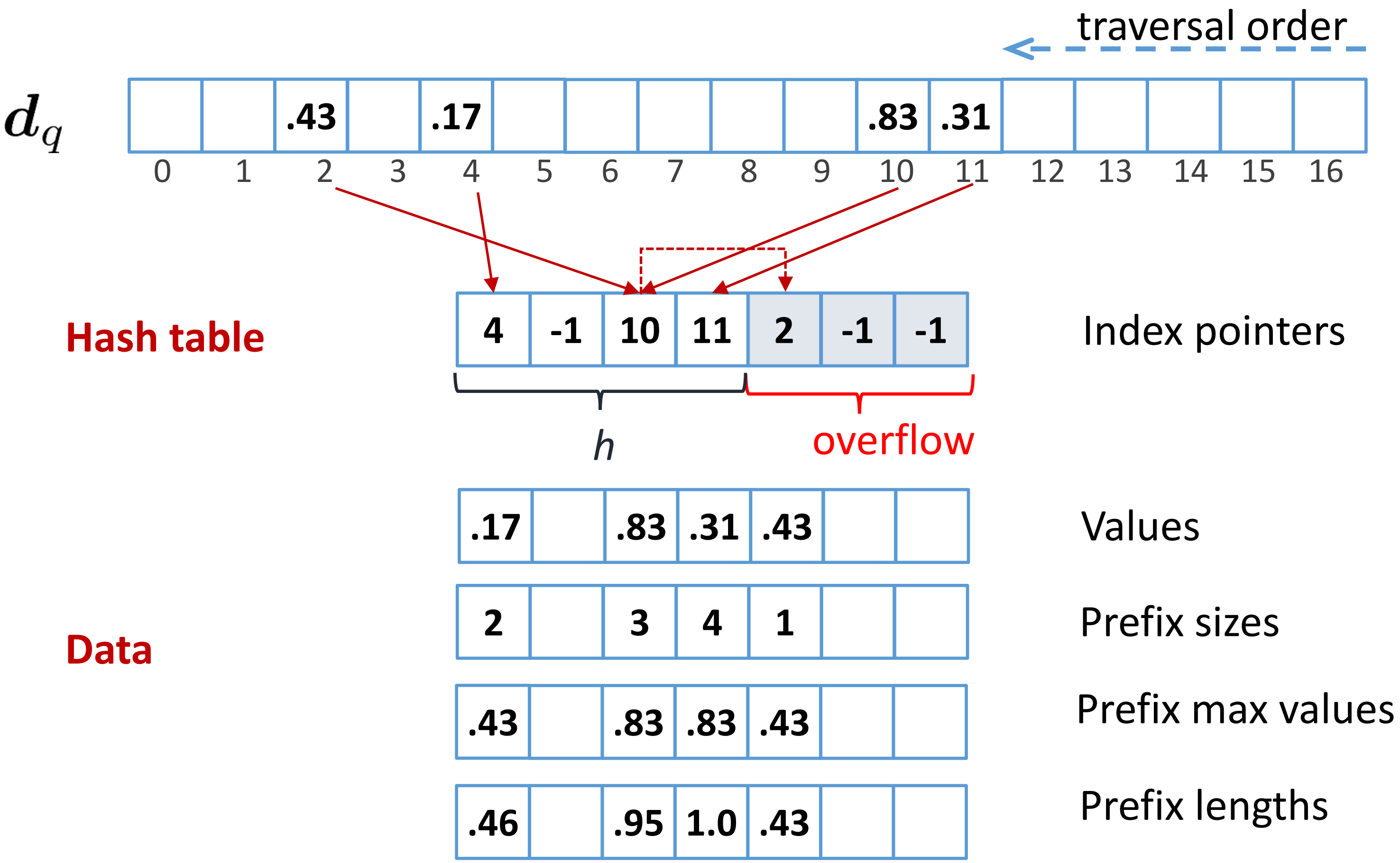
- Many processes/threads means potential contention over data structures or output space
  - Must carefully design methods that delineate independent work for the threads while ensuring load balance

- If all threads have working data, they may overwhelm the cache and cause delays due to cache misses
  - Cache tiling and memory-efficient data structures can help reduce the cache footprint of each thread

Cache tiling



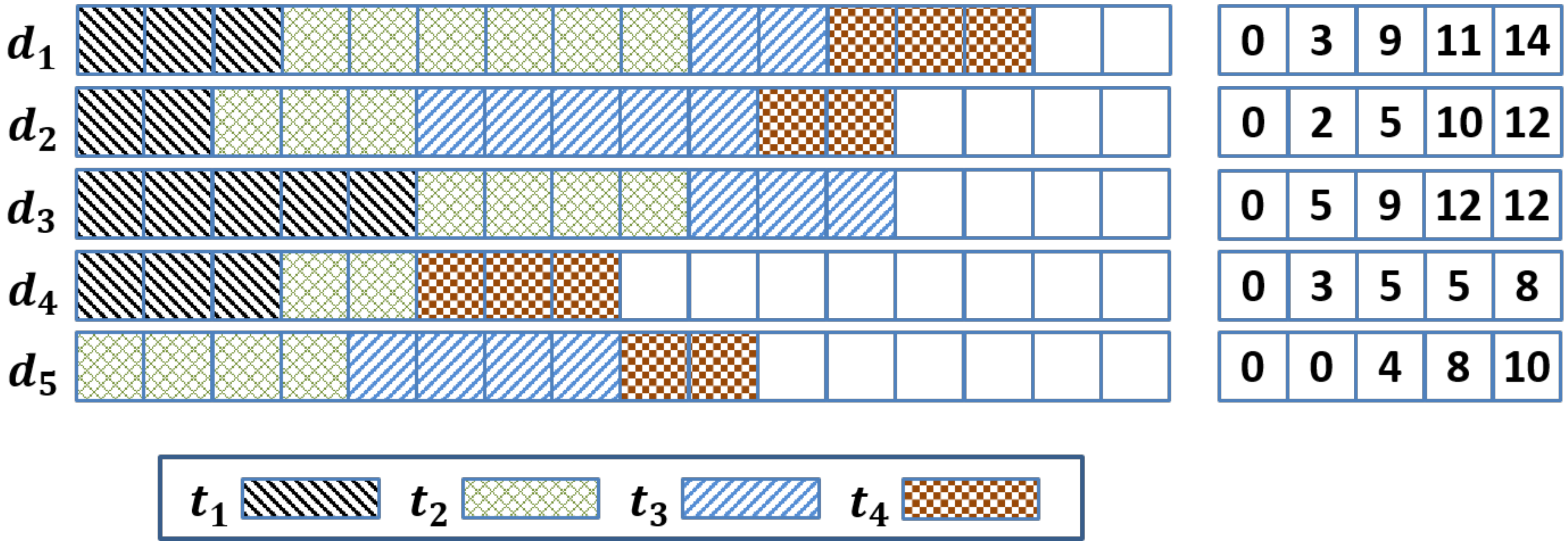
Masked hash table



Partial linear overflow scan during collision lookup.

Neighborhood updates

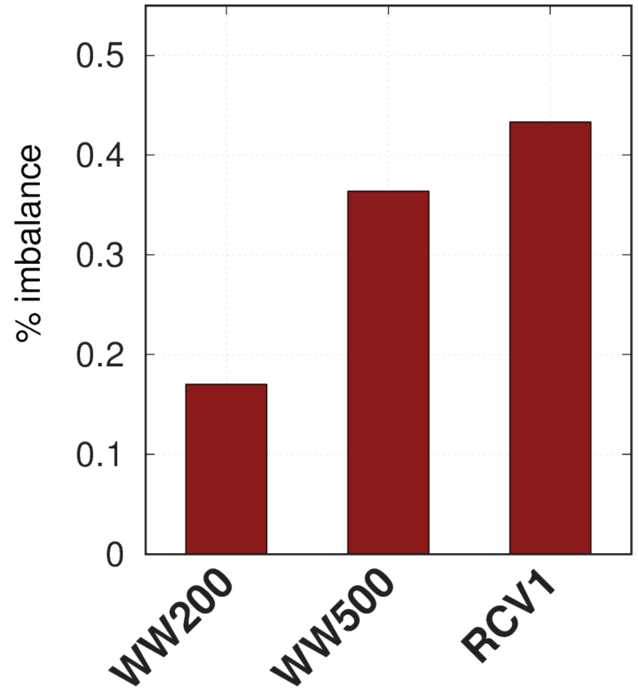
- Local neighborhood updated during search
- Candidate neighborhood updates staged for cooperative update at the end of query block



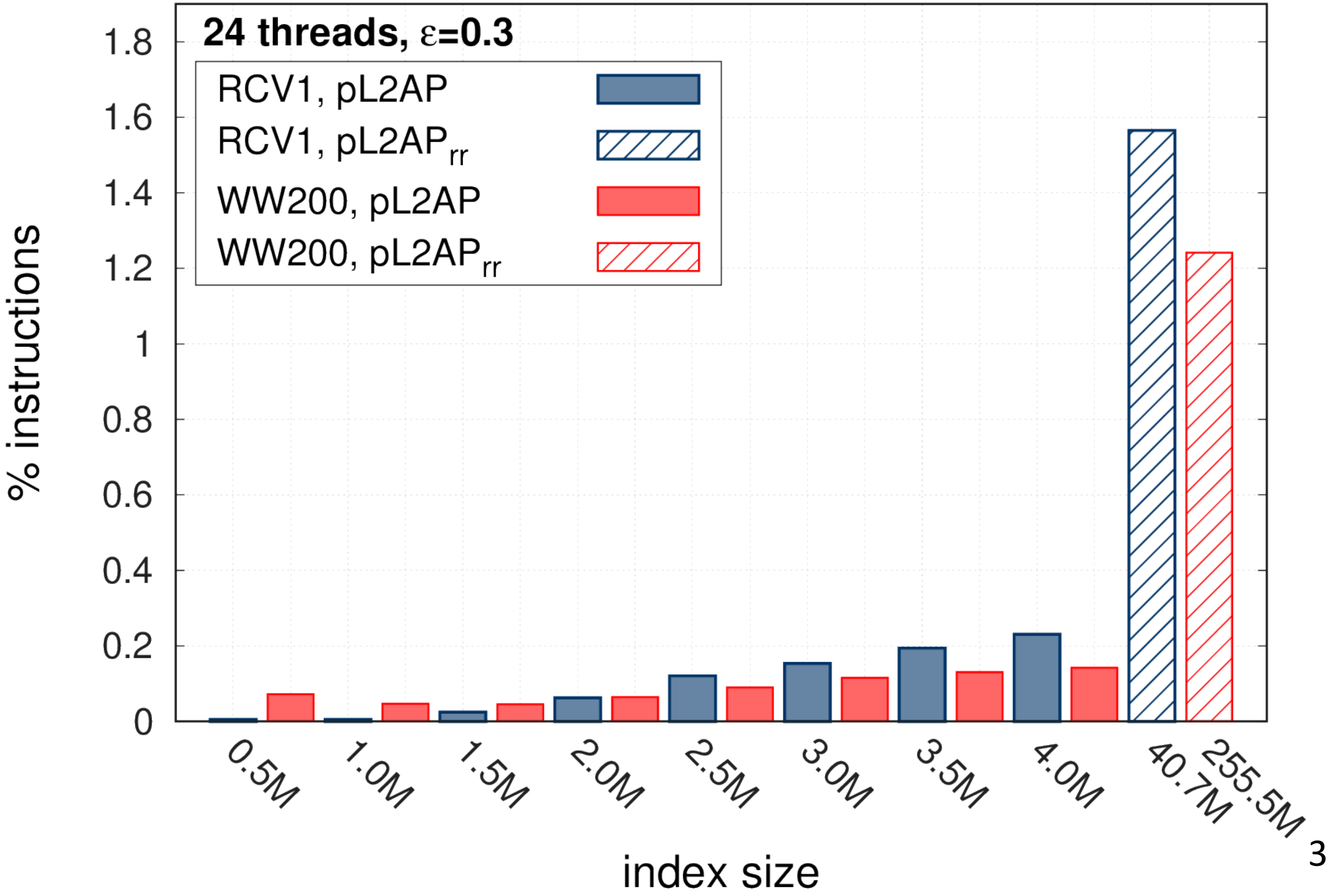
Tiling in practice

pL2AP – tiled index
pL2AP_rr – full inverted index

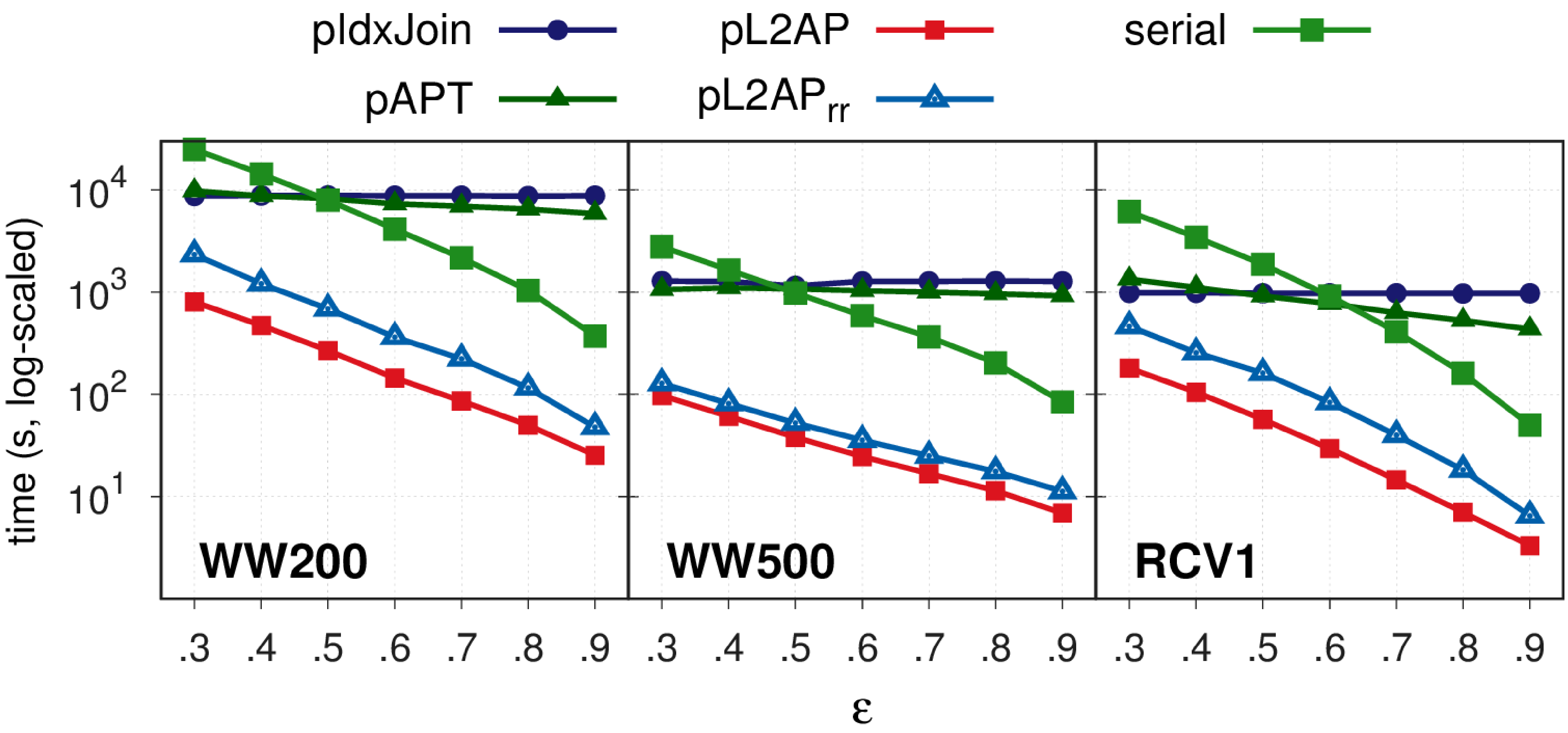
Load Balance



Percent Instructions Leading to Cache Misses (Collisions)



How fast is it?



24 threads @ 2.5 GHz Intel Xeon E5-2680v3 \w 30 Mb Cache

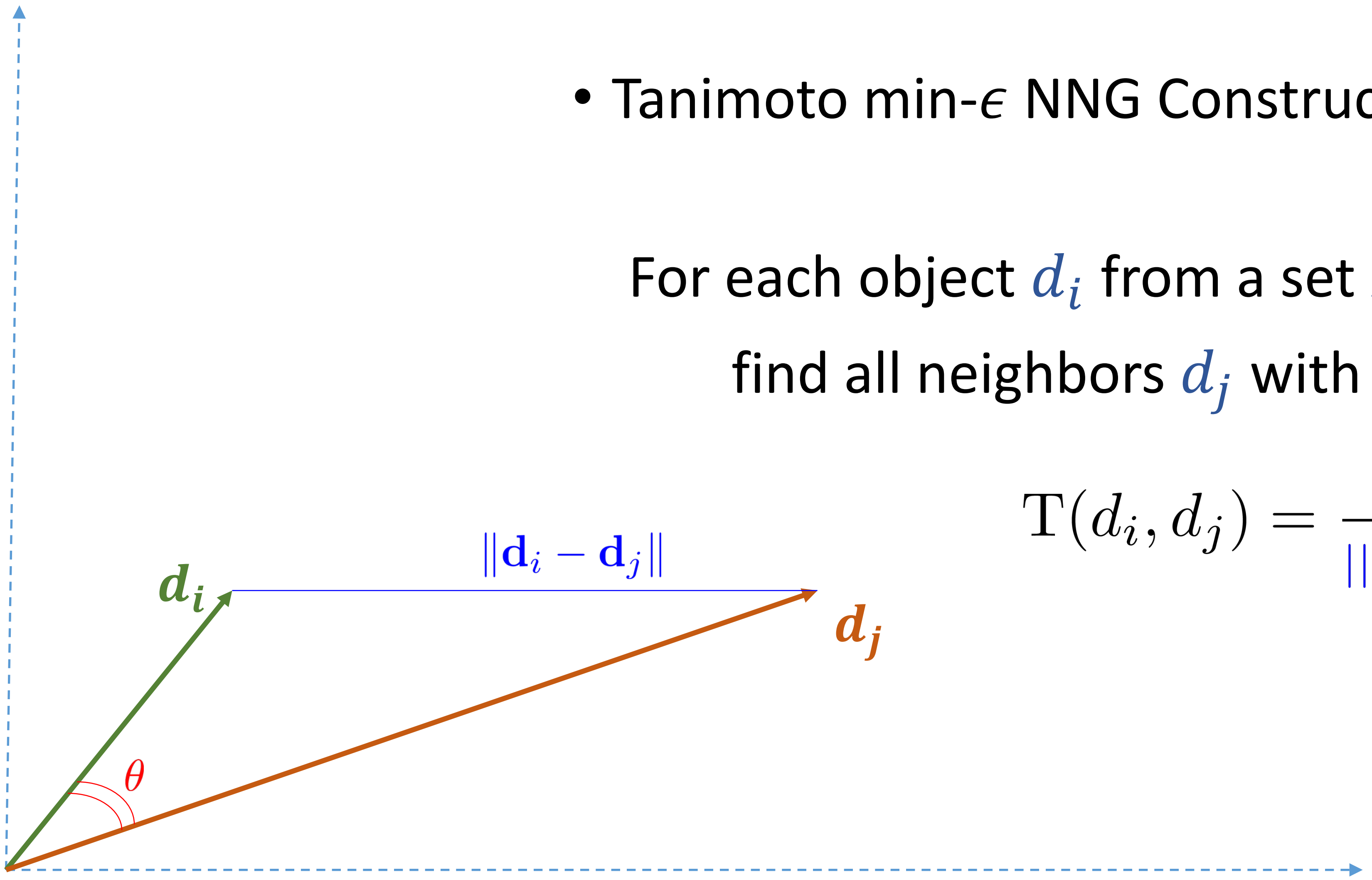
When both length and angles matter

- Tanimoto min-ϵ NNG Construction

For each object d_i from a set D,

find all neighbors d_j with $\mathrm{T}(d_i, d_j) \geq \epsilon$.

$$\mathrm{T}(d_i, d_j) = \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{||\mathbf{d}_i||_2^2 + ||\mathbf{d}_j||_2^2 - \langle \mathbf{d}_i, \mathbf{d}_j \rangle}$$



http://davidanastasiu.net/software/tapnn/

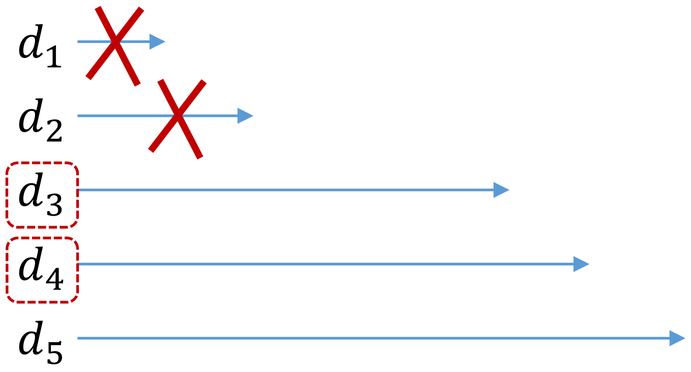
Length-based filtering

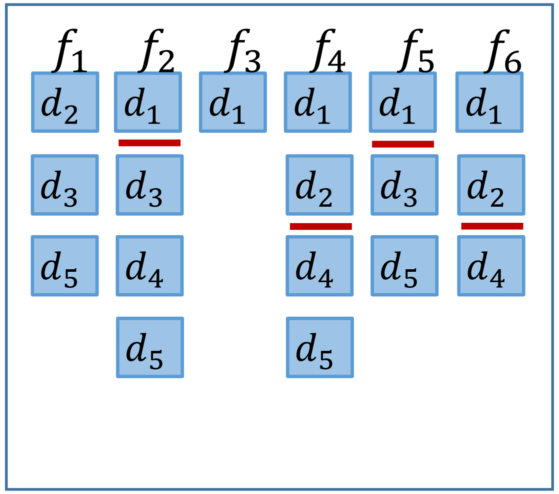
- d_q and d_c cannot be neighbors unless

$$\|\mathbf{d}_c\| \in \left[(1/\alpha)\|\mathbf{d}_q\|,\ \alpha\|\mathbf{d}_q\|\right],$$

$$\alpha = \frac{1}{2}\left(\left(1+\frac{1}{\epsilon}\right) + \sqrt{\left(1+\frac{1}{\epsilon}\right)^2 - 4}\right)$$

  - α bound due to Marzena Kryszkiewicz, IIDS 2013
- Relabel objects in non-decreasing length order





Inverted Index

Subset of cosine neighborhood

- The following inequalities hold for our domain:

$$\mathrm{T}(d_i, d_j) \leq \mathrm{C}(d_i, d_j)$$
$$\mathrm{T}(d_i, d_j) \geq \epsilon \Rightarrow \mathrm{C}(d_i, d_j) \geq \epsilon$$
$$\mathrm{C}(d_i, d_j) < \epsilon \Rightarrow \mathrm{T}(d_i, d_j) < \epsilon$$

- Potential solution
  - Store vector norms and normalize vectors
  - Find cosine neighbors (L2AP-like filtering here)
  - Transform $\mathrm{C}(d_i, d_j)$ to $\mathrm{T}(d_i, d_j)$
  - Remove non-Tanimoto neighbors
- Tighter bound due to Lee et al., DEXA 2010

$$\mathrm{T}(d_i, d_j) \geq \epsilon \Rightarrow \mathrm{C}(d_i, d_j) \geq \frac{2\epsilon}{1+\epsilon} = t$$

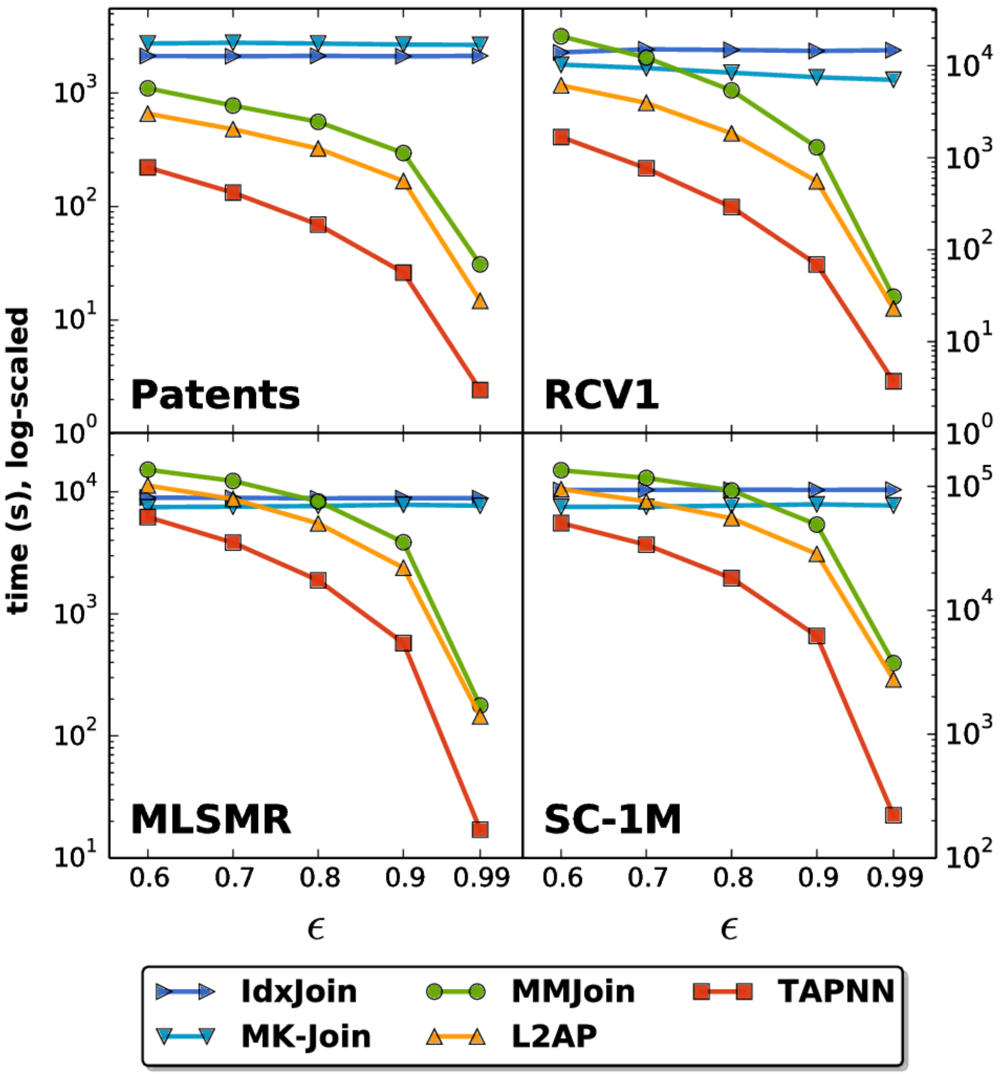
Length + Angle-Based Pruning

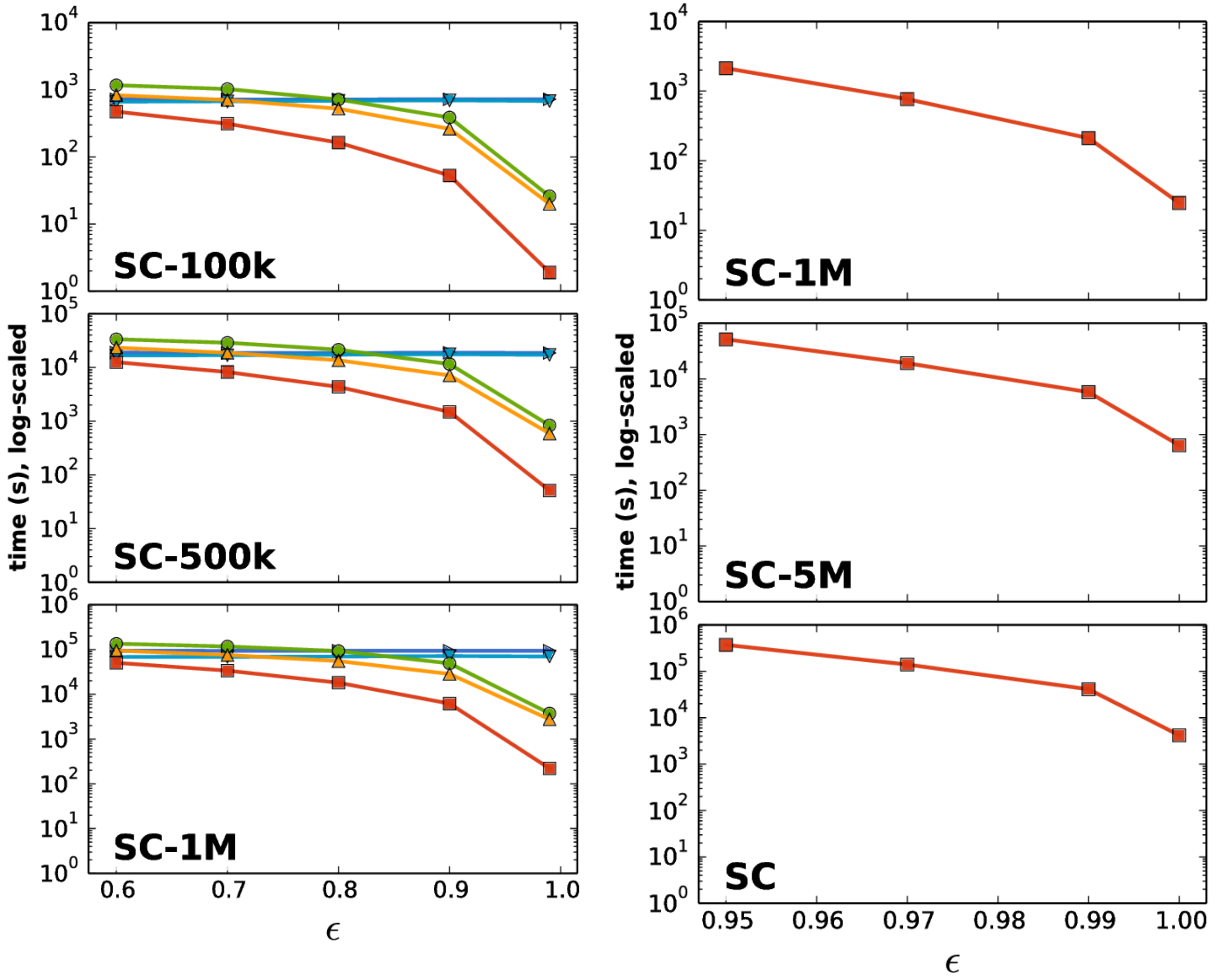
- d_q and d_c cannot be neighbors unless

$$\|\mathbf{d}_c\| \in \left[(1/\beta)\left\|\mathbf{d}_q\right\|,\ \beta\left\|\mathbf{d}_q\right\|\right], \quad \beta = \frac{s}{t} + \sqrt{\left(\frac{s}{t}\right)^2 - 1}$$

where s is any cosine similarity upper bound
such as the ones we compute during filtering.

How fast is it?





Scales well as data set size increases.

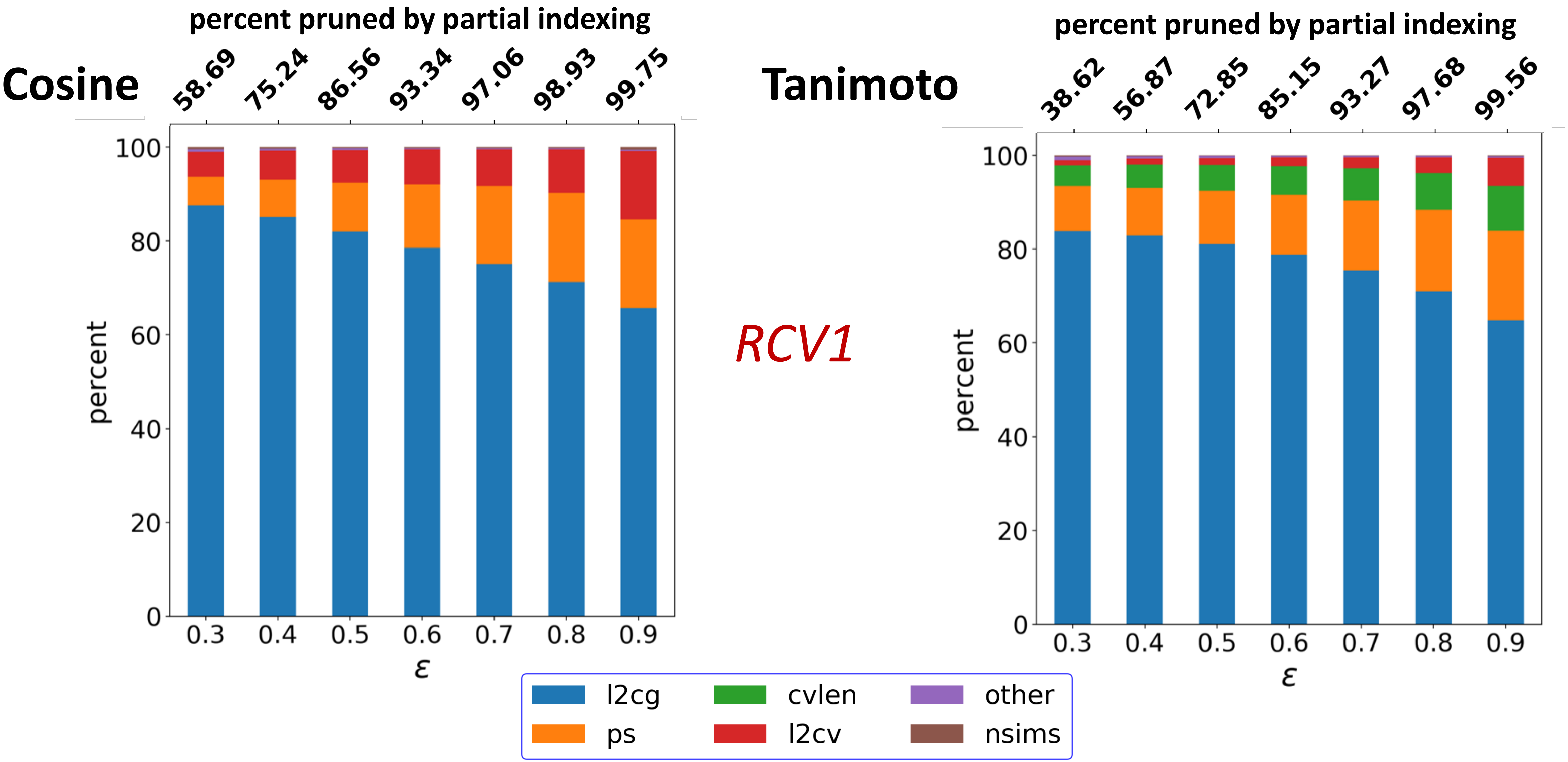
Effective proximity bounds and when they are most useful

What will the output look like?

And, a related question, how do I choose ϵ/k?

- Output of similarity search/graph construction is data-dependent
  - For some data sets, you get no neighbors at $\epsilon = 0.95$ cosine similarity; for others, you get many neighbors
  - A given ϵ threshold means different things in different contexts
- Number of neighbors is dependent on dimensionality
  - By the pigeon hole principle, when $n \gg m$, more likely to see collisions (features in common)
- Filtering effectiveness is dependent on stdev of feature weights
  - If all features weight the same, it will take longer to accumulate similarity
- Parameter choices are often dependent on subsequent analysis that the neighbors are sought for

Pruning Effectiveness Comparison



Neighborhood Graph Statistics

ϵ	μ	ρ	μ	ρ	μ	ρ
	WW500k		**RCV1**		**Orkut**	
0.1	1,749	3.5e-03	10,986	1.4e-02	76	2.5e-05
0.2	233	4.7e-04	2,011	2.5e-03	21	6.9e-06
0.3	64	1.3e-04	821	1.0e-03	7.2	2.4e-06
0.4	25	5.1e-05	355	4.4e-04	2.3	7.6e-07
0.5	10	2.2e-05	146	1.8e-04	0.69	2.3e-07
0.6	4.7	9.5e-06	57	7.2e-05	0.22	7.2e-08
0.7	2.1	4.2e-06	25	3.2e-05	0.09	3.1e-08
0.8	0.93	1.9e-06	14	1.8e-05	0.07	2.1e-08
0.9	0.28	5.7e-07	8.1	1.0e-05	0.06	2.0e-08

μ: Average neighborhood size

ρ: Output graph density

Not all pruning is created equal

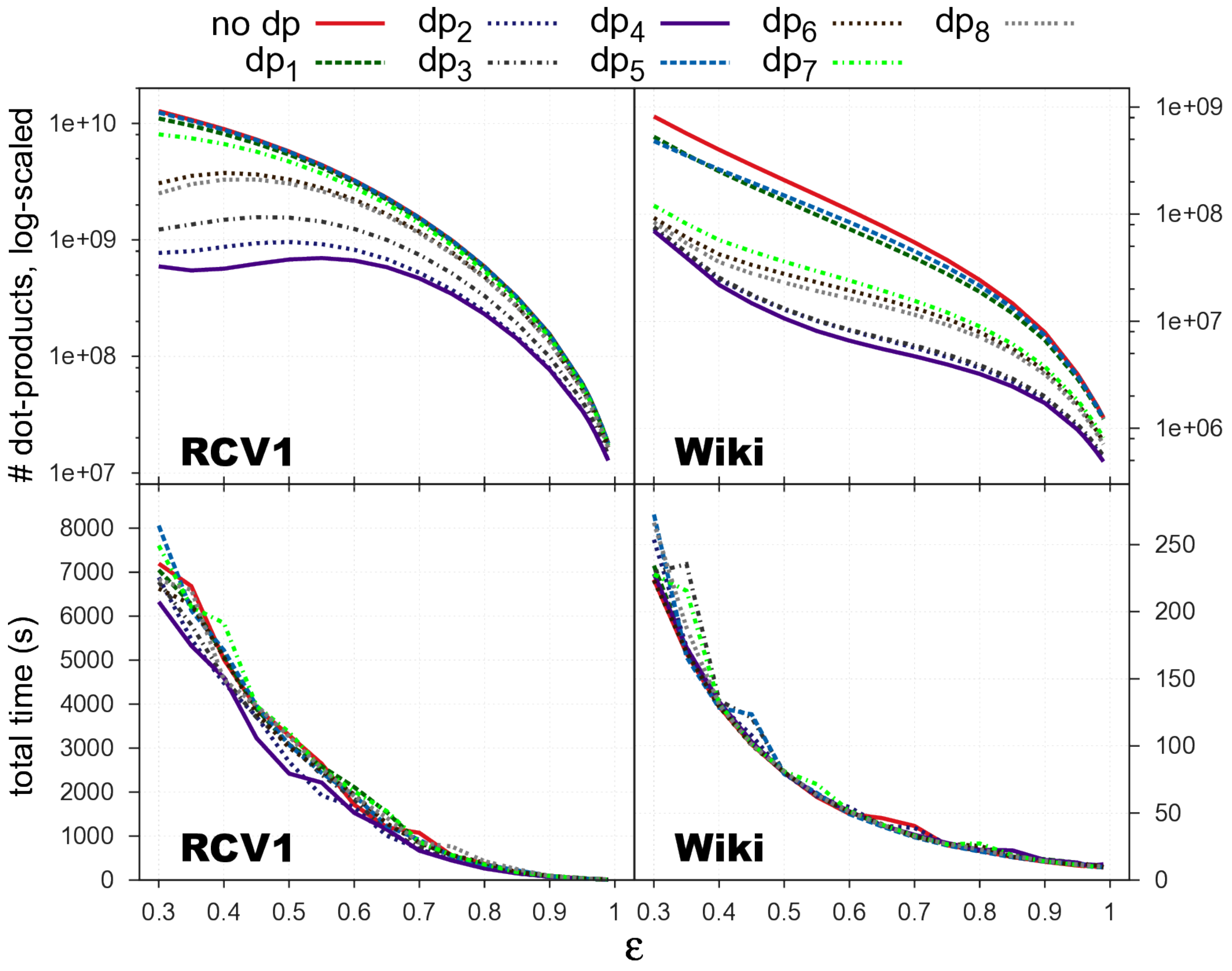
- Designed many filtering criteria for the min-ϵ and k-NN problems. E.g.,

bound	stage	target	estimate
idx	idx	$\text{sim}(\boldsymbol{d}_q^{\leq j}, \boldsymbol{d}_{>q})$	$\min(\langle \boldsymbol{d}_{\mathbf{q}}^{\leq \mathbf{j}}, \boldsymbol{mx}_{\geq \boldsymbol{q}} \rangle, \lVert \boldsymbol{d}_q^{\leq j} \rVert_2)$
sz	c.g.	$\min(\lVert \boldsymbol{d}_c \rVert_0)$	$(\epsilon / \lVert \boldsymbol{d}_q \rVert_\infty)^2$
rs		$\text{sim}(\boldsymbol{d}_q^{\leq j}, \boldsymbol{d}_{<q})$	$\min(\langle \boldsymbol{d}_{\mathbf{q}}^{\leq \mathbf{j}}, \boldsymbol{mx} \rangle, \lVert \boldsymbol{d}_q^{\leq j} \rVert_2)$
$l2cg$		$\text{sim}(\boldsymbol{d}_q^{<j}, \boldsymbol{d}_c^{<j})$	$\lVert \boldsymbol{d}_q^{<j} \rVert_2 \times \lVert \boldsymbol{d}_c^{<j} \rVert_2$
ps	c.v.	$\text{sim}(\boldsymbol{d}_q, \boldsymbol{d}_c^{\leq})$	$\min(\langle \boldsymbol{d}_{\mathbf{c}}^{\leq}, \boldsymbol{mx}_{\geq \boldsymbol{c}} \rangle, \lVert \boldsymbol{d}_c^{\leq} \rVert_2)$
dps_1		$\text{sim}(\boldsymbol{d}_q, \boldsymbol{d}_c^{\leq})$	$\min(\lVert \boldsymbol{d}_q \rVert_0, \lVert \boldsymbol{d}_c^{\leq} \rVert_0) \times \lVert \boldsymbol{d}_q \rVert_\infty \times \lVert \boldsymbol{d}_c^{\leq} \rVert_\infty$
dps_2		$\text{sim}(\boldsymbol{d}_q, \boldsymbol{d}_c^{\leq})$	$\min(\lVert \boldsymbol{d}_q \rVert_0, \lVert \boldsymbol{d}_c^{\leq l} \rVert_0) \times \lVert \boldsymbol{d}_q^{\leq l} \rVert_\infty \times \lVert \boldsymbol{d}_c^{\leq l} \rVert_\infty$
$l2cv$		$\text{sim}(\boldsymbol{d}_q^{<j}, \boldsymbol{d}_c^{<j})$	$\lVert \boldsymbol{d}_q^{<j} \rVert_2 \times \lVert \boldsymbol{d}_c^{<j} \rVert_2$

- Some of the criteria are problem-specific (L2-Norm-bound is not)
- Of all criterial, the L2-Norm bound is the most productive (by far)
- Some have suggested less pruning may be more efficient
  - E.g., De Francisci Morales & Gionis, VLDB'16 (extended L2AP to streaming case)
  - May be data specific, but has not been my finding so far

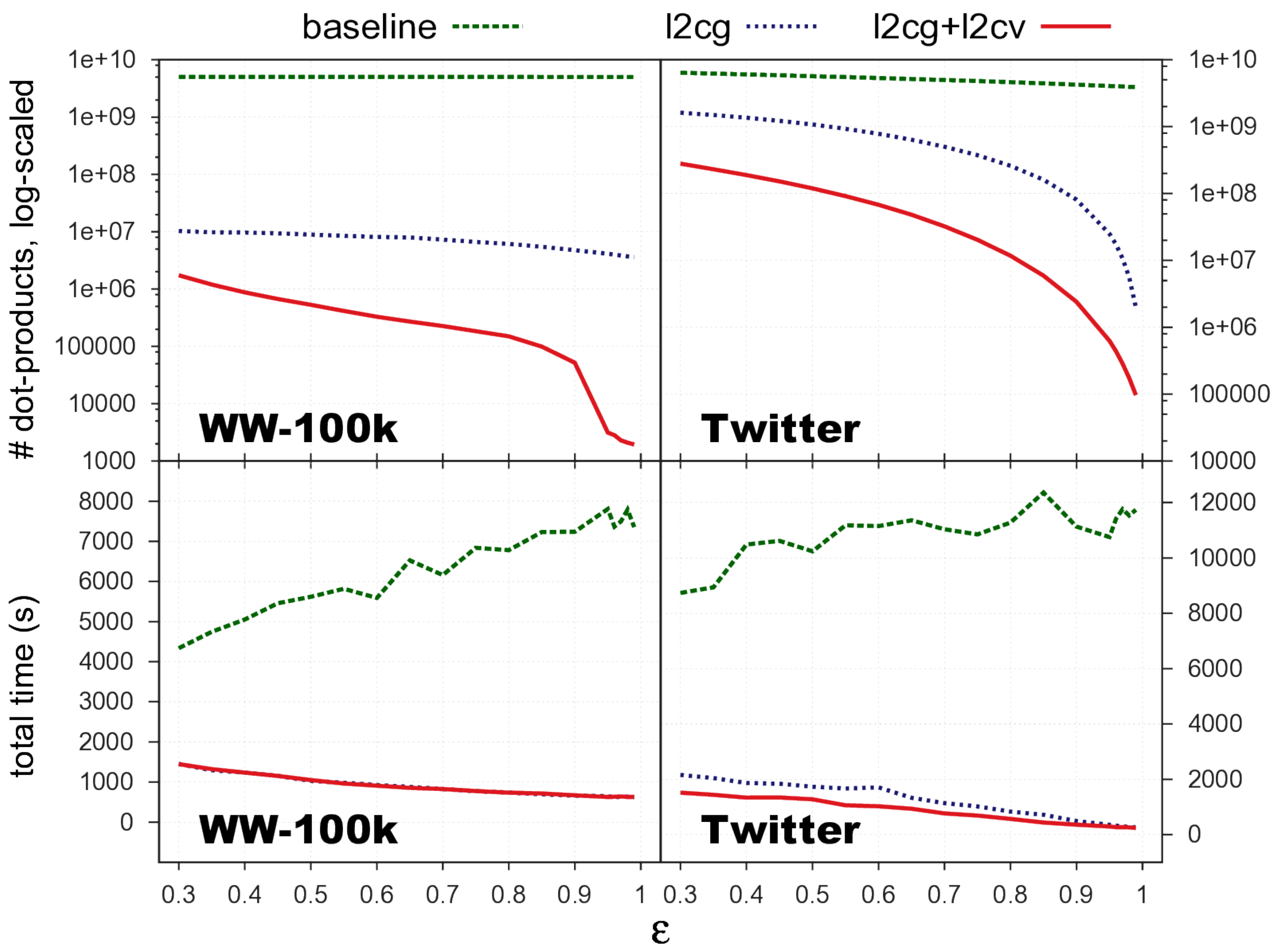
When less is more

- The amount of pruning is not directly proportional to efficiency



bound	estimate
dps_1	$\min(\|\|\boldsymbol{d}_q\|\|_\infty \times \|\|\boldsymbol{d}_c^{\leq}\|\|_1,\ \|\|\boldsymbol{d}_c^{\leq}\|\|_\infty \times \|\|\boldsymbol{d}_q\|\|_1)$
dps_2	$\min(\|\|\boldsymbol{d}_q^{\leq p}\|\|_\infty \times \|\|\boldsymbol{d}_c^{\leq}\|\|_1,\ \|\|\boldsymbol{d}_c^{\leq}\|\|_\infty \times \|\|\boldsymbol{d}_q^{\leq p}\|\|_1)$
dps_3	$\min(\|\|\boldsymbol{d}_q\|\|_\infty \times \|\|\boldsymbol{d}_c^{\leq p}\|\|_1,\ \|\|\boldsymbol{d}_c\|\|_\infty \times \|\|\boldsymbol{d}_q^{\leq p}\|\|_1)$
dps_4	$\min(\|\|\boldsymbol{d}_q^{\leq p}\|\|_\infty \times \|\|\boldsymbol{d}_c^{\leq p}\|\|_1,\ \|\|\boldsymbol{d}_c^{\leq p}\|\|_\infty \times \|\|\boldsymbol{d}_q^{\leq p}\|\|_1)$
dps_5	$\min(\|\|\boldsymbol{d}_q\|\|_0, \|\|\boldsymbol{d}_c^{\leq}\|\|_0) \times \|\|\boldsymbol{d}_q\|\|_\infty \times \|\|\boldsymbol{d}_c^{\leq}\|\|_\infty$
dps_6	$\min(\|\|\boldsymbol{d}_q\|\|_0, \|\|\boldsymbol{d}_c^{\leq p}\|\|_0) \times \|\|\boldsymbol{d}_q^{\leq p}\|\|_\infty \times \|\|\boldsymbol{d}_c^{\leq p}\|\|_\infty$
dps_7	$\min(\|\|\boldsymbol{d}_q^{\leq p}\|\|_0, \|\|\boldsymbol{d}_c^{\leq p}\|\|_0) \times \|\|\boldsymbol{d}_q\|\|_\infty \times \|\|\boldsymbol{d}_c^{\leq p}\|\|_\infty$
dps_8	$\min(\|\|\boldsymbol{d}_q\|\|_0, \|\|\boldsymbol{d}_c^{\leq p}\|\|_0) \times \|\|\boldsymbol{d}_q^{\leq p}\|\|_\infty \times \|\|\boldsymbol{d}_c^{\leq p}\|\|_\infty$

Some filtering may cover other filtering



Summary and Open Questions

In summary

- Creating sparsity-aware algorithms goes a long way towards efficient solutions to hard problems
- Filtering is a very effective technique for similarity search, especially for sparse data and asymmetric proximity measures
- L2-norm filtering is extremely effective for cosine similarity and Tanimoto coefficient – may also be beneficial in other proximity measures (e.g., Euclidean distance)
- More research is needed to:
  - Derive new even tighter filtering bounds
  - Identify optimum balance between checking and not checking bounds
  - Characterize NNS pruning and output based on feature statistics
    - Yuliang Li et al. (ICDT2019) prove optimality guarantees for L2-norm filtering of skewed data
    - They also propose alternate and partial inverted list traversal + lower-bound filters

Questions?

References

[1] Yuliang Li, Jianguo Wang, Benjamin Pullman, Nuno Bandeira and Yannis Papakonstantinou Index-based, High-dimensional, Cosine Threshold Querying with Optimality Guarantees. ICDT 2019.

[2] David C. Anastasiu & George Karypis. Parallel cosine nearest neighbor graph construction. Elsevier Journal of Parallel and Distributed Computing, 2017. Impact factor: 1.815.

[3] David C. Anastasiu & George Karypis. Efficient identification of Tanimoto nearest neighbors; All Pairs Similarity Search Using the Extended Jaccard Coefficient. Springer International Journal of Data Science and Analytics, 4(3):153-172, 2017.

[4] Gianmarco De Francisci Morales and Aristides Gionis. 2016. Streaming similarity self-join. Proc. VLDB Endow. 9, 10 (June 2016), 792-803. DOI: http://dx.doi.org/10.14778/2977797.2977805

[5] David C. Anastasiu and George Karypis. Efficient Identification of Tanimoto Nearest Neighbors. Proceedings of the 3rd IEEE International Conference on Data Science and Advanced Analytics (DSAA 2016).

[6] David C. Anastasiu & George Karypis. Fast Parallel Cosine K-Nearest Neighbor Graph Construction. In 2016 6th Workshop on Irregular Applications: Architecture and Algorithms (IA3) (IA3 2016), pages 50-53, 2016.

[7] David C. Anastasiu & George Karypis. PL2AP: Fast Parallel Cosine Similarity Search. In Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms, in conjunction with SC'15 (IA3 2015), pages 1-8, ACM, 2015.

[8] David C. Anastasiu and George Karypis. L2Knng: Fast Exact K-Nearest Neighbor Graph Construction with L2-Norm Pruning. In 24th ACM International Conference on Information and Knowledge Management, CIKM '15, 2015.

References

[9] David C. Anastasiu and George Karypis. L2AP: Fast Cosine Similarity Search With Prefix L-2 Norm Bounds. Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE 2014).

[10] M. Kryszkiewicz. Using non-zero dimensions and lengths of vectors for the tanimoto similarity search among real valued vectors. Intelligent Information and Database Systems. Springer International Publishing, 2014, pp. 173-182.

[11] Youngki Park, Sungchan Park, Sang-goo Lee, and Woosung Jung. Greedy filtering: A scalable algorithm for k-nearest neighbor graph construction. In Database Systems for Advanced Applications, volume 8421 of Lecture Notes in Computer Science, pages 327-341. Springer-Verlag, 2014.

[12] Venu Satuluri and Srinivasan Parthasarathy. 2012. Bayesian locality sensitive hashing for fast similarity search. Proc. VLDB Endow. 5, 5 (January 2012), 430-441.

[13] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In Proceedings of the 20th International Conference on World Wide Web, WWW ’11, pages 577–586, New York, NY, USA, 2011. ACM.

[14] Dongjoo Lee, Jaehui Park, Junho Shim, and Sang-goo Lee. 2010. An efficient similarity join algorithm with cosine similarity predicate. In Proceedings of the 21st international conference on Database and expert systems applications: Part II (DEXA'10), Pablo Garcia Bringas, Abdelkader Hameurlain, and Gerald Quirchmayr (Eds.). Springer-Verlag, Berlin, Heidelberg, 422-436

[15] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In Proceedings of the 16th international conference on World Wide Web (WWW '07). ACM, New York, NY, USA, 131-140.