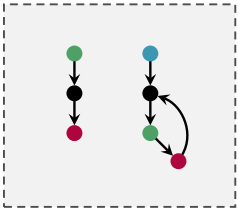
MetricJoin: Leveraging Metric Properties for Robust Exact Set Similarity Joins

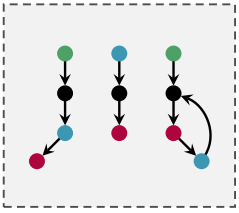
Manuel Widmoser, Daniel Kocher, Nikolaus Augsten, Willi Mann

University of Salzburg

Similarity Join

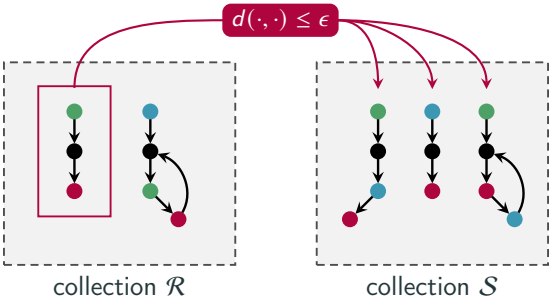


collection $\mathcal{R}$

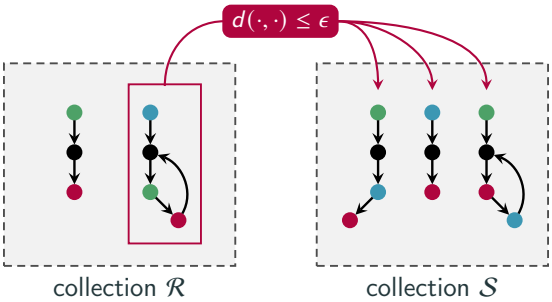


collection $\mathcal{S}$

Similarity Join



Similarity Join



$|\mathcal{R}| \cdot |\mathcal{S}|$ many comparisons

Set Similarity Join

- Model processes as (ordered) sets:

- Set similarity join: $\mathcal{R} \tilde{\bowtie} \mathcal{S} = \{(r, s) \in \mathcal{R} \times \mathcal{S} \mid d(r, s) \leq \epsilon\}$

e.g., Hamming:
$d(r, s) = |r \cup s| - |r \cap s|$

- Other applications: duplicate detection, recommendation systems, plagiarism detection, data cleaning, ...

Filter-Verification Framework

1. Candidate generation
   - Based on various filters
   - **Goal**: generate number of candidates close to $|R \tilde{\bowtie} S|$
2. Verification of candidate pairs

Filter-Verification Framework

1. Candidate generation
   - Based on various filters
   - **Goal**: generate number of candidates close to $|R \tilde{\bowtie} S|$
2. Verification of candidate pairs

Existing solutions are **not robust** to different data characteristics

Prefix Filter

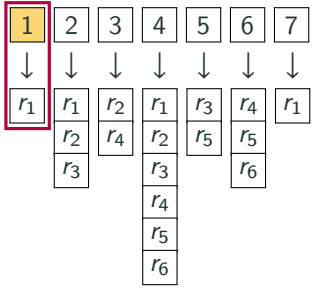
Idea: No common token in prefixes $\Rightarrow$ r and s cannot be similar

1. Globally order the tokens
2. Compute prefix length of r and s (based on $d(\cdot, \cdot)$ and ϵ)

Prefix-based Inverted List Index ($\epsilon = 2$)

ID	Set $r_i \in \mathcal{R}$					
r_1	1	2	4	10	11	12
r_2	2	3	4	5	9	11
r_3	2	4	5	7	9	10
r_4	3	4	6	7	9	11
r_5	4	5	6	7	8	10
r_6	4	6	7	8	9	10

1 create index



2 query the index

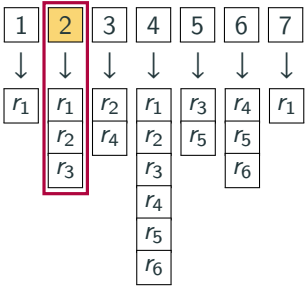
q

1	2	4	6	7	10

Prefix-based Inverted List Index ($\epsilon = 2$)

ID	Set $r_i \in \mathcal{R}$					
r_1	1	2	4	10	11	12
r_2	2	3	4	5	9	11
r_3	2	4	5	7	9	10
r_4	3	4	6	7	9	11
r_5	4	5	6	7	8	10
r_6	4	6	7	8	9	10

1 create index



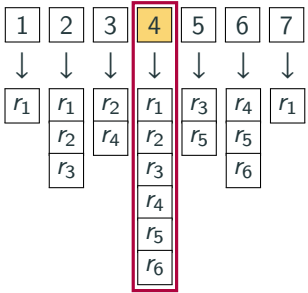
2 query the index

q	1	2	4	6	7	10

Prefix-based Inverted List Index ($\epsilon = 2$)

ID	Set $r_i \in \mathcal{R}$					
r_1	1	2	4	10	11	12
r_2	2	3	4	5	9	11
r_3	2	4	5	7	9	10
r_4	3	4	6	7	9	11
r_5	4	5	6	7	8	10
r_6	4	6	7	8	9	10

1 create index

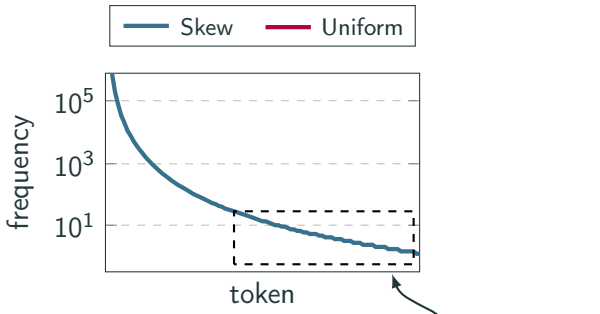


2 query the index

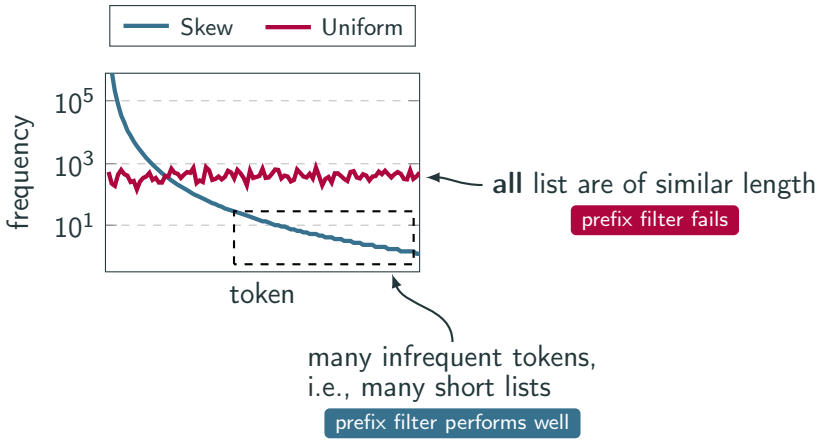
q	1	2	4	6	7	10

Scanning many large lists is slow

Token Frequency Distribution



Token Frequency Distribution



Outline of MetricJoin

Idea: spatially index long lists and visit only promising sub-regions

---

1. Transform sets into vectors
2. Create a prefix index
3. Extend long lists with spatial components (to organize the vectors)
4. Visit lists partially by leveraging metric properties

Data Transformation

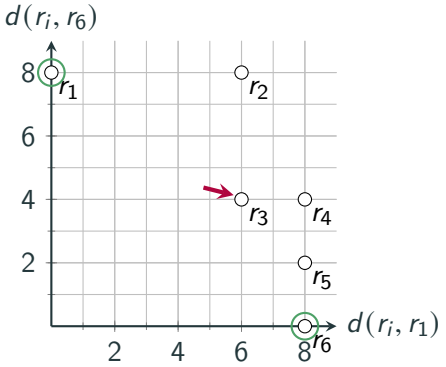
Transform the metric space of sets into an n-dimensional vector space:

1. Pick n pivot objects $\mathcal{P} \subseteq \mathcal{R}$
2. For each $r \in \mathcal{R}$ store a vector representation of r:

$$\phi(r) = \begin{pmatrix} d(r, p_1) \\ d(r, p_2) \\ \vdots \\ d(r, p_n) \end{pmatrix} \in \mathbb{R}^n \text{ and } p_i \in \mathcal{P}$$

Vector Space Transformation

ID	Set $r_i \in \mathcal{R}$						$\phi(r)$
r_1	1	2	4	10	11	12	$\langle 0, 8 \rangle$
r_2	2	3	4	5	9	11	$\langle 6, 8 \rangle$
r_3	2	4	5	7	9	10	$\langle 6, 4 \rangle$
r_4	3	4	6	7	9	11	$\langle 8, 4 \rangle$
r_5	4	5	6	7	8	10	$\langle 8, 2 \rangle$
r_6	4	6	7	8	9	10	$\langle 8, 0 \rangle$



$$\underbrace{\left|d(r,p) - d(p,s)\right|}_{= \left|\phi(r) - \phi(s)\right|} \overset{1}{\leq} \boxed{d(r,s) \leq \epsilon}$$

2 extend to n dimensions

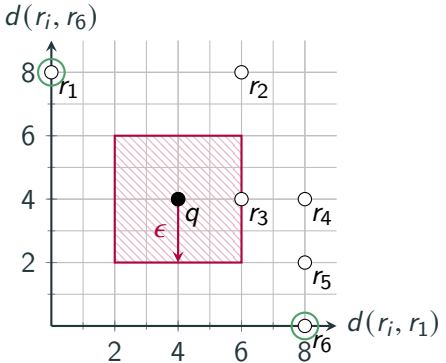
$$\max_{j \in [1,n]} \left|\phi_j(r) - \phi_j(s)\right| \leq \epsilon$$

$\Leftrightarrow$ (rearrangement) 3

$$\forall_j : \underbrace{\phi_j(r)}_{\text{in index}} \in \underbrace{\left[\phi_j(s) - \epsilon,\ \phi_j(s) + \epsilon\right]}_{\text{query rectangle}}$$

Vector Range Query ($\epsilon = 2$)

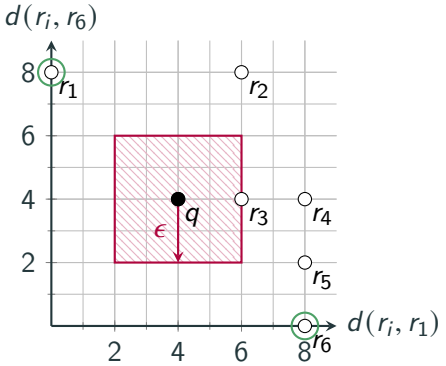
ID	Set $r_i \in \mathcal{R}$	$\phi(r)$
r_1	1 2 4 10 11 12	$\langle 0, 8 \rangle$
r_2	2 3 4 5 9 11	$\langle 6, 8 \rangle$
r_3	2 4 5 7 9 10	$\langle 6, 4 \rangle$
r_4	3 4 6 7 9 11	$\langle 8, 4 \rangle$
r_5	4 5 6 7 8 10	$\langle 8, 2 \rangle$
r_6	4 6 7 8 9 10	$\langle 8, 0 \rangle$
q	1 2 4 6 7 10	$\langle 4, 4 \rangle$



Vector Range Query ($\epsilon = 2$)

ID	Set $r_i \in \mathcal{R}$	$\phi(r)$
r_1	1 2 **4** 10 11 12	$\langle 0, 8 \rangle$
r_2	2 3 **4** 5 9 11	$\langle 6, 8 \rangle$
r_3	2 **4** 5 7 9 10	$\langle 6, 4 \rangle$
r_4	3 **4** 6 7 9 11	$\langle 8, 4 \rangle$
r_5	**4** 5 6 7 8 10	$\langle 8, 2 \rangle$
r_6	**4** 6 7 8 9 10	$\langle 8, 0 \rangle$
q	1 2 **4** 6 7 10	$\langle 4, 4 \rangle$

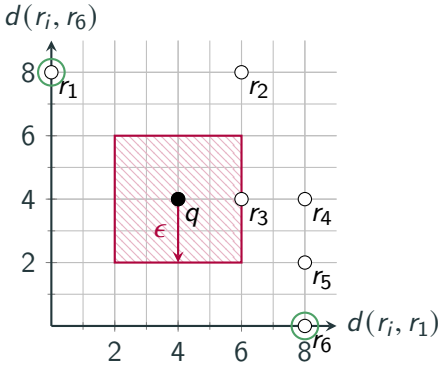
prefix length $= \epsilon + 1 = 3$



Vector Range Query ($\epsilon = 2$)

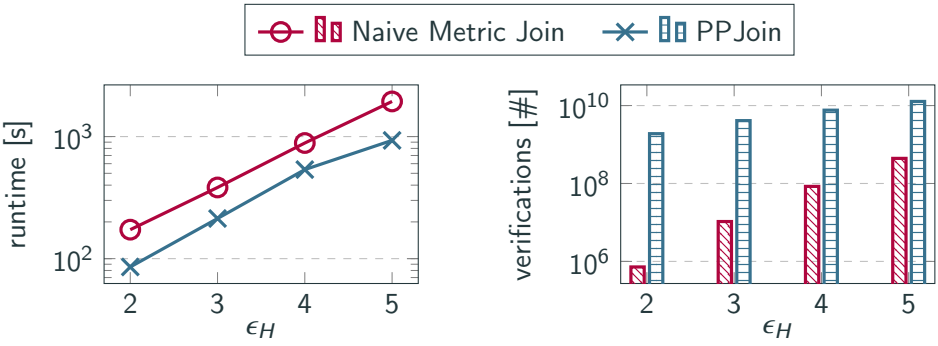
ID	Set $r_i \in \mathcal{R}$	$\phi(r)$
r_1	1 2 4 10 11 12	$\langle 0, 8 \rangle$
r_2	2 3 4 5 9 11	$\langle 6, 8 \rangle$
r_3	2 4 5 7 9 10	$\langle 6, 4 \rangle$
r_4	3 4 6 7 9 11	$\langle 8, 4 \rangle$
r_5	4 5 6 7 8 10	$\langle 8, 2 \rangle$
r_6	4 6 7 8 9 10	$\langle 8, 0 \rangle$
q	1 2 4 6 7 10	$\langle 4, 4 \rangle$

prefix length $= \epsilon + 1 = 3$



use a **spatial** index
(e.g., R-tree)

A Naive Metric Join Is Not Enough



Scanning lists is cheaper than accessing spatial indexes

DBLP-V12 dataset, $|\mathcal{P}| = 10$

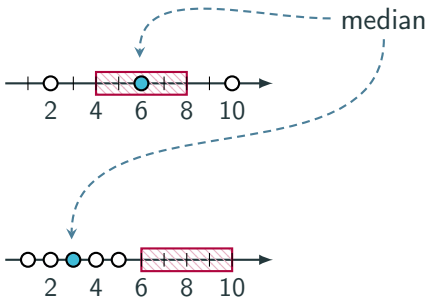
Use Spatial Components Carefully

Only query the spatial index if the objects are:

1. **well distributed**

or the query rectangle is

2. **off-aligned**



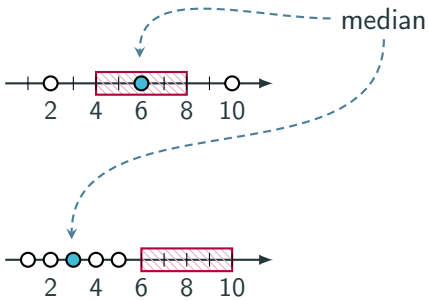
Use Spatial Components Carefully

Only query the spatial index if the objects are:

1. **well distributed**

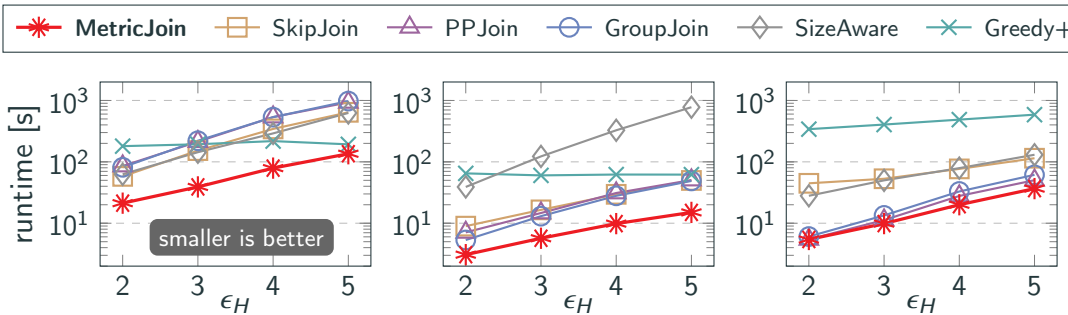
or the query rectangle is

2. **off-aligned**



Otherwise, perform a sequential list scan

Experiments: Runtime

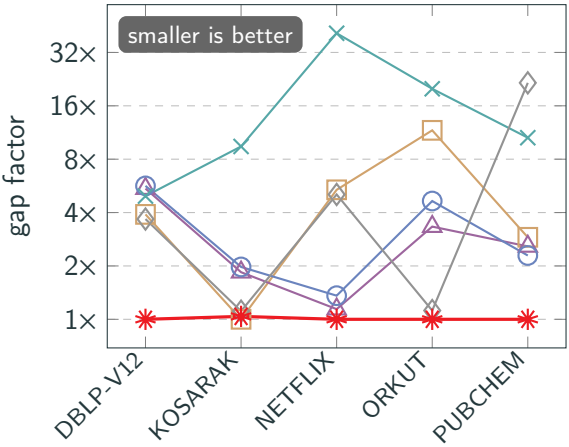


Experiments: Robustness

Gap Factor

$$= \frac{\text{runtime}}{\text{runtime of fastest}}$$

- Hamming distance
- fixed $\epsilon = 3$



Conclusion

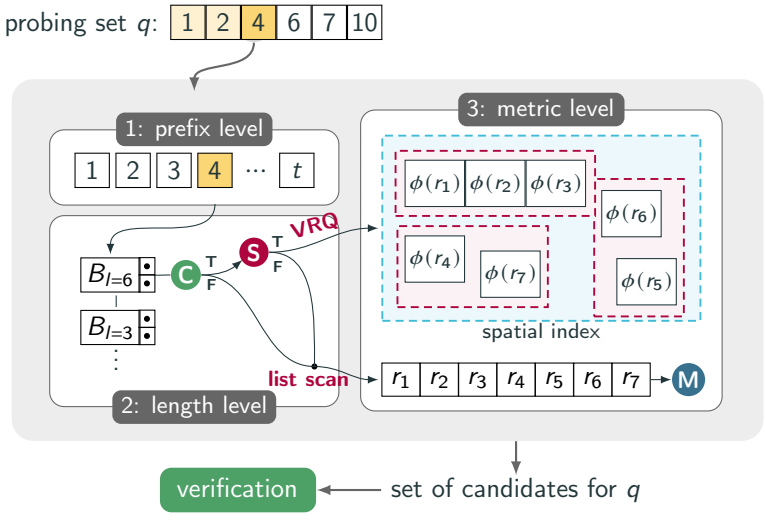
MetricJoin leverages metric properties of the set distance to speed up set similarity joins:

1. Transform sets from a general metric space into a multi-dimensional vector space
2. Construct a three-level metric index combining prefix, length, and a metric filter into a single linear-space index structure
3. Carefully decide whether to perform a vector range query or a list scan

MetricJoin is **robust** and performs well across various dataset characteristics

Appendix

Big Picture


probing set q: 1 2 4 6 7 10
1: prefix level
1 2 3 4 ··· t
$B_{l=6}$
$B_{l=3}$
C
T
F
S
T
F
VRQ
list scan
2: length level
3: metric level
$\phi(r_1)$ $\phi(r_2)$ $\phi(r_3)$
$\phi(r_6)$
$\phi(r_4)$
$\phi(r_7)$
$\phi(r_5)$
spatial index
r_1 r_2 r_3 r_4 r_5 r_6 r_7
M
verification
set of candidates for q

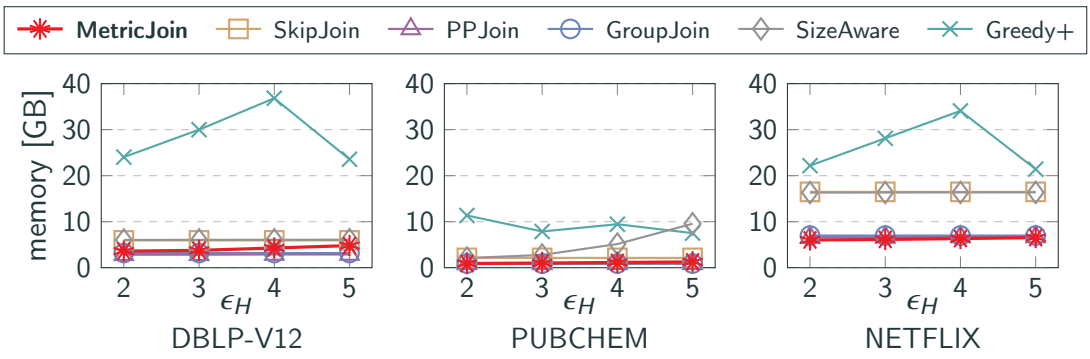
Idea: Pick pivots close to the hull of the dataset (Hull of Foci[1]).

- The first 2 pivots are the two most distance objects
- For the remaining $n-2$ pivots: pick a pivot o such that the distance to subsequently picked pivots $p \in \mathcal{P}$ is close to the distance of the initial pivots:

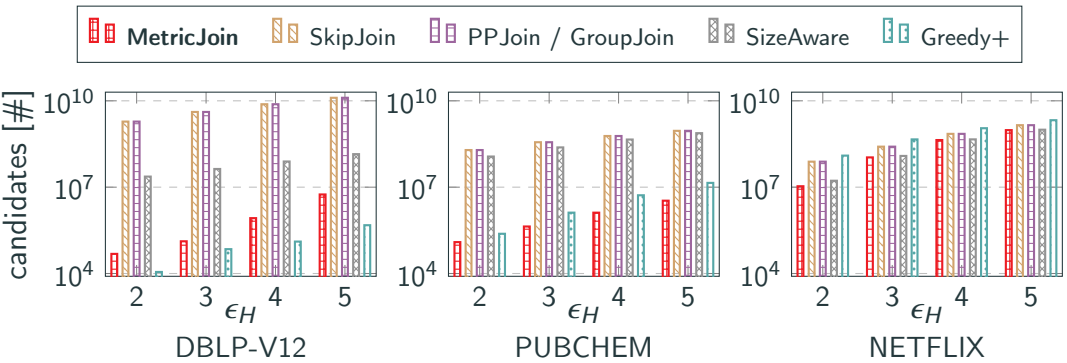
$$\min_{o \in \mathcal{R} \setminus \mathcal{P}} \sum_{p \in \mathcal{P}} |d(p_1, p_2) - d(o, p)|$$

1 Traina et al, *The Omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient*, VLDB Journal, 2007.

Experiments: Memory Consumption



Experiments: Candidates



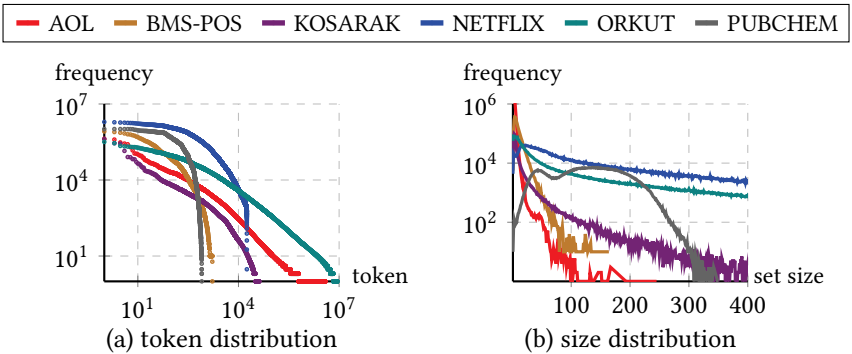
Spatial Query Statistics

Dataset	Indexes	Skips	Returns	
			avg	max
AOL	13.2%	6.3%	41.6%	99.1%
BMS-POS	85.9%	2.3%	0.5%	31.2%
CELONIS	75.0%	6.5%	2.8%	68.6%
DBLP-V12	81.1%	0.1%	0.1%	36.3%
KOSARAK	77.4%	68.4%	20.2%	87.5%
NETFLIX	4.0%	0.9%	3.9%	61.3%
ORKUT	14.6%	19.1%	19.7%	83.3%
PUBCHEM	62.0%	0.1%	0.3%	51.0%
TWITTER	65.6%	28.0%	14.1%	97.4%

Dataset Characteristics

Dataset	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert r\rvert_{\text{avg}}$	$\lvert\mathcal{P}\rvert$	Metric
AOL	$1.0 \cdot 10^7$	$3.9 \cdot 10^6$	3	10	Jaccard
BMS-POS	$3.1 \cdot 10^6$	$1.7 \cdot 10^3$	9	10	Jaccard
CELONIS	$2.6 \cdot 10^6$	$3.5 \cdot 10^3$	22	20	Jaccard
DBLP-V12	$4.6 \cdot 10^6$	$2.5 \cdot 10^4$	75	10	Hamming
KOSARAK	$6.1 \cdot 10^5$	$4.1 \cdot 10^4$	12	2	Hamming
NETFLIX	$4.8 \cdot 10^6$	$1.8 \cdot 10^4$	210	10	Hamming
ORKUT	$2.7 \cdot 10^6$	$8.7 \cdot 10^6$	120	2	Hamming
PUBCHEM	$1.0 \cdot 10^6$	$8.1 \cdot 10^2$	127	10	Hamming
TWITTER	$4.6 \cdot 10^6$	$6.1 \cdot 10^5$	12	5	Jaccard

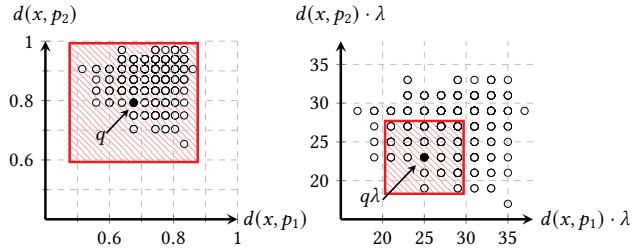
Frequency Distribution



(a) token distribution

(b) size distribution

Problem: Many normalized distances (e.g., Jaccard) are always 1 if the overlap is 0 (regardless of other properties).



(a) Without shifting (b) After shifting

Solution: Shift the objects by λ from a "bounded" to an "unbounded" space:

$$d_J(r,s) \cdot \lambda_J = \left(1 - \frac{|r \cap s|}{|r \cup s|}\right) \cdot \lambda_J = |r \cup s| - |r \cap s| = d_H(r,s)$$

Problem: Dice and Cosine violate the triangle inequality.

Solution: Project the non-metric distance to a metric (e.g., Jaccard):

1. We map sets to vectors using the Jaccard distance
2. For each vector range query, we adapt ϵ s.t. it returns a candidate superset, which includes all sets that are within the Dice or Cosine threshold, respectively.

$$\text{Idea: } d_{D|C}(r,s) \leq \epsilon \Leftrightarrow d_J(r,s) \leq \epsilon \cdot \alpha_{D|C}$$

3. Verification with the non-metric distance ensures the correct join result.

For overlap similarity: $|r \cap s| \geq t$ and $t = 4$

- $pre(r) = |r| - t + 1$
- r : 1 2 3 ? ? ? s : 4 5 ? ? ?
- Idea: with the remaining tokens (3 ?) we cannot reach the desired overlap of 4

For other distances:

- Replace t by an "equivalent overlap"
- E.g., for Hamming: $d_H(r, s) = |r \cup s| - |r \cap s| \Rightarrow |r \cap s| \geq \frac{|r|+|s|-\epsilon}{2} = t$

Length Filter

Idea: Two sets can only be similar if their size difference is not too large.

The overlap is trivially bounded by the set size, i.e., $t \leq |r \cap s| \leq |s|$.

- Replace t by an "equivalent overlap"
- E.g., for Hamming: $d_H(r,s) = |r \cup s| - |r \cap s| \Rightarrow |r \cap s| \geq \frac{|r|+|s|-\epsilon}{2} = t$
- $t \leq |s| \Rightarrow |r| - \epsilon \leq |s|$, i.e., $|s|$ must be at least of size $|r| - \epsilon$ to be similar