Scalable and Robust Set Similarity Join
using the Chosen Path Tree

Tobias Christiani, Rasmus Pagh and Johan Sivertsen
IT University of Copenhagen
{tobc, pagh, jovt}@itu.dk

European Research Council
Established by the European Commission

Introduction

Set similarity join is a fundamental and well-studied database operator. It is usually studied in the *exact* setting where the goal is to compute all pairs of sets that exceed a given level of similarity (measured e.g. as Jaccard similarity). But set similarity join is often used in settings where 100% recall may not be important — indeed, where the exact set similarity join is itself only an approximation of the desired result set.

We present a new randomized algorithm for set similarity join that can achieve any desired recall up to 100%, and show theoretically and empirically that it significantly outperforms state-of-the-art implementations of exact methods, and improves on existing approximate methods. Our experiments on benchmark data sets show the method is several times faster than comparable approximate methods. At 90% recall the algorithm is often more than 2 orders of magnitude faster than exact methods. Our algorithm makes use of recent theoretical advances in high-dimensional sketching and indexing that we believe to be of wider relevance to the data engineering community.

Approximate Set Similarity Join

In the regular Ser Similarity Join, given sets A and B and threshold λ, we must return

$$A \bowtie_\lambda B = \{(a,b) | a \in A, b \in B, sim(a,b) > \lambda\}.$$

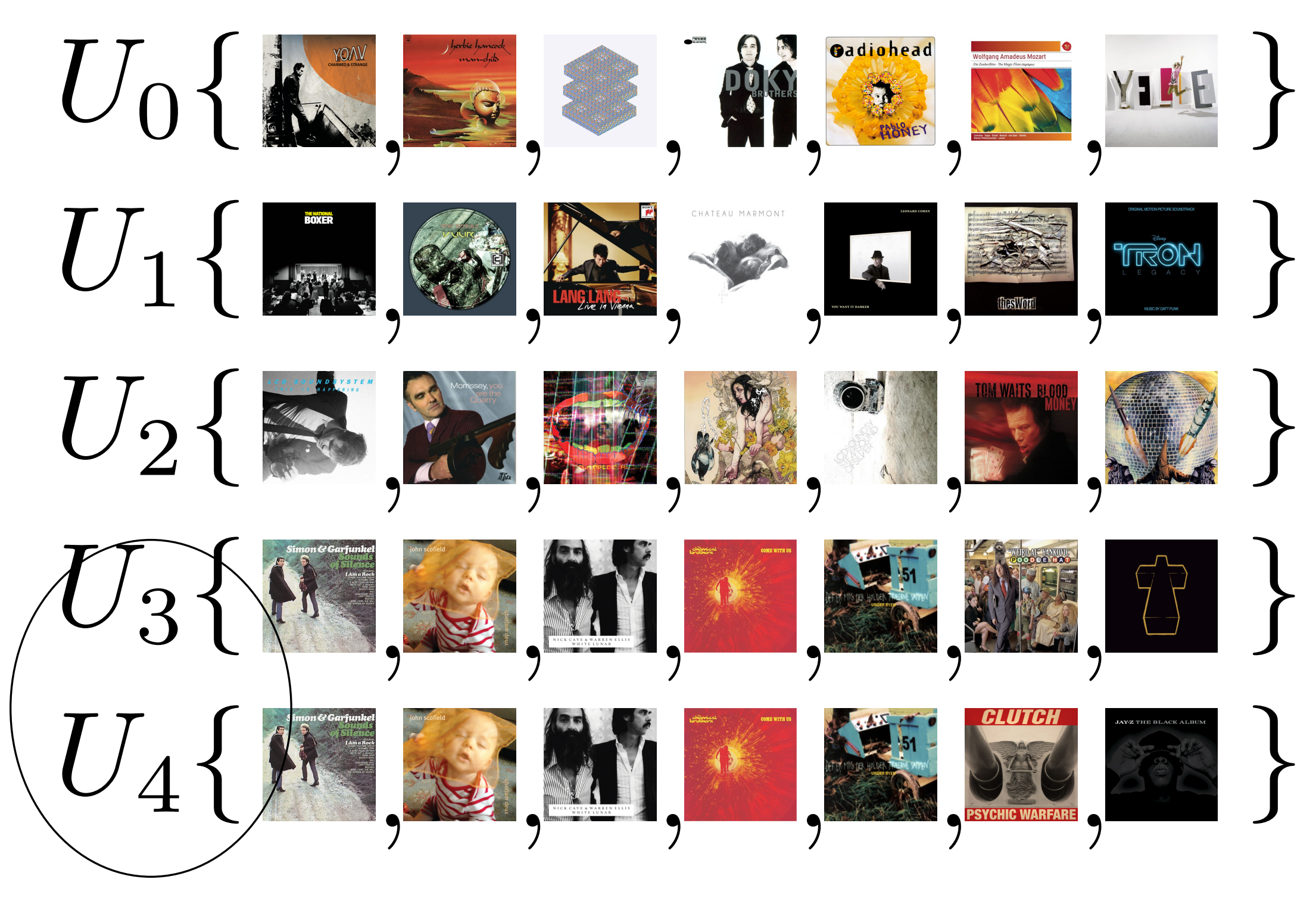

Figure 1: Example: We wish to find users with similar record collections.

(λ, ϕ)-Set similarity Join: Given two sets A and B, a threshold $\lambda \in (0,1)$ and a recall $\phi \in (0,1)$. Return

$$L \subseteq A \bowtie_\lambda B$$

such that $\Pr[(x,y) \in L] \geq \phi$ for any $(x,y) \in A \bowtie_\lambda B$.

Our Contributions

We present a new approximate set similarity join algorithm: Chosen Path Similarity Join (CPSJoin). We cover its theoretical underpinnings, and show experimentally that it achieves high recall with a substantial speedup compared to state-of-the-art exact techniques. The two key ideas behind CPSJoin are:

- A new recursive filtering technique inspired by the recently proposed ChosenPath index for set similarity search [1], adding new ideas to make the method parameter-free, near-linear space, and adaptive to a given data set.
- Apply efficient sketches for estimating set similarity [2] that take advantage of modern hardware.

Algorithm

Our algorithm follows a divide and conquer design based on the Chosen Path Tree. In the Chosen Path Tree a child denotes the subset of the parent set that shares some element. A path $p = (j_1, \ldots, j_k)$ of length k then identifies the subset sharing elements $j_{1,\ldots,k}$.

How to divide?

For a set $x \in S$, we sample a random hash function $r \colon [d] \to [0,1]$ and construct children for every element $j \in x$ such that $r(j) < \frac{1}{\lambda|x|}$. Note that this places every similar pair in one child on expectation. The subset of the data set S that survives to a node with path $p = (j_1, \ldots, j_k)$ is given by

$$S_p = \{x \in S \mid x_{j_1} = 1 \wedge \cdots \wedge x_{j_k} = 1\}.$$

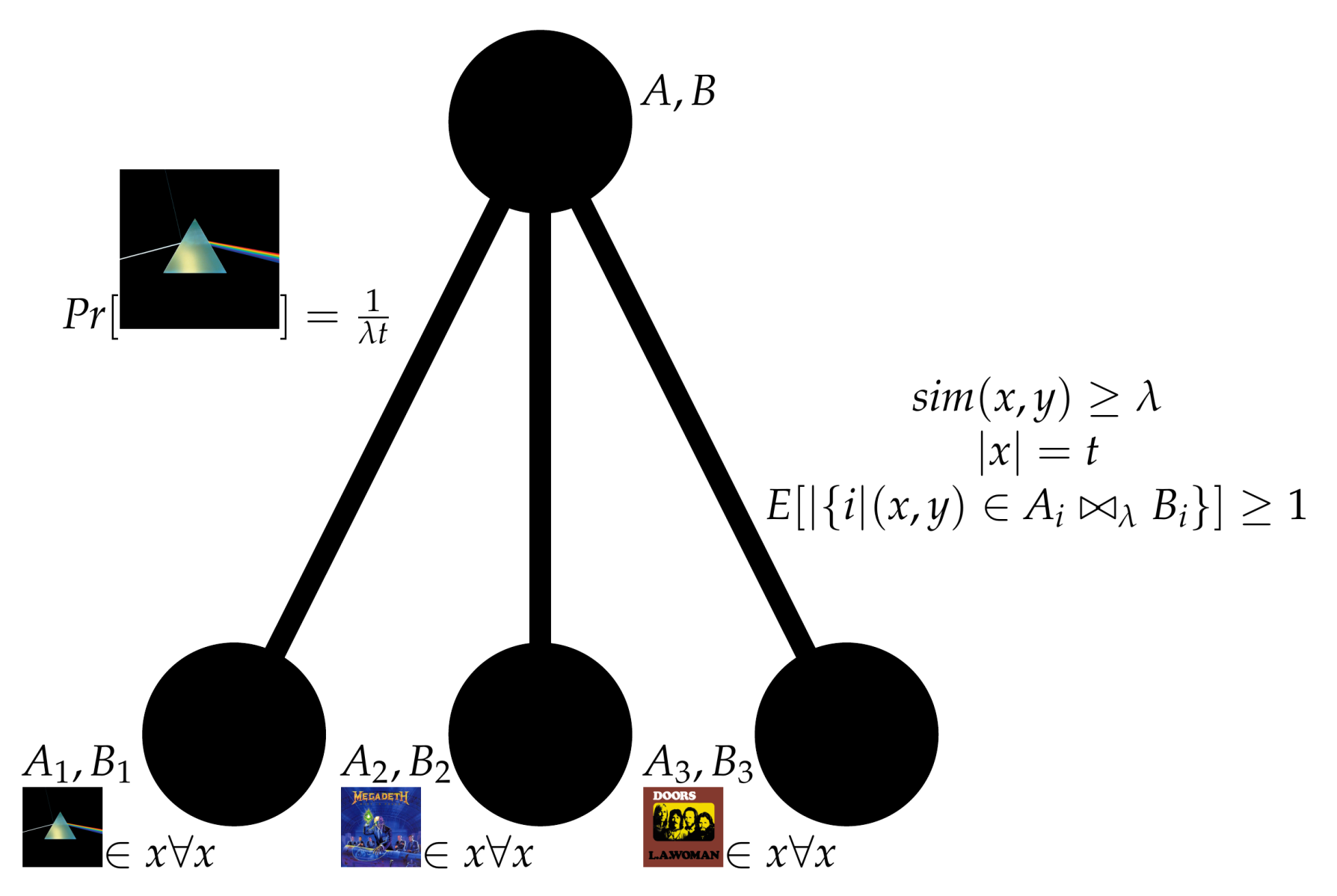

Figure 2: On expectation every similar pair is assigned to a subproblem.

When to conquer?

The BruteForce step removes a point x from the Chosen Path branching process, instead opting to compare it against every other point $y \in S$, if it satisfies the condition

$$\frac{1}{|S|-1} \sum_{y \in S \setminus \{x\}} |x \cap y|/t > (1-\varepsilon)\lambda. \tag{1}$$

We claim that this condition minimizes the expected number of comparisons performed by the algorithm: Consider a node in the Chosen Path Tree associated with a set of points S while running the CPSJoin algorithm. For a point $x \in S$, we can either remove it from S immediately at a cost of $|S|-1$ comparisons, or we can choose to let continue in the branching process (possibly into several nodes) and remove it later. The expected number of comparisons if we let it continue k levels before removing it from every node that it is contained in, is given by

$$\sum_{y \in S \setminus \{x\}} \left(\frac{1}{\lambda} \frac{|x \cap y|}{t} \right)^k.$$

This expression is convex and increasing in the similarity $|x \cap y|/t$ between x and other points $y \in S$, allowing us to state the following observation:

Observation 1 (Recursion). *Let $\varepsilon = 0$ and consider a set S containing a point $x \in S$ such that x satisfies the recursion condition in equation* (1). *Then the expected number of comparisons involving x if we continue branching exceeds $|S|-1$ at every depth $k \geq 1$. If x does not satisfy the condition, the opposite is observed.*

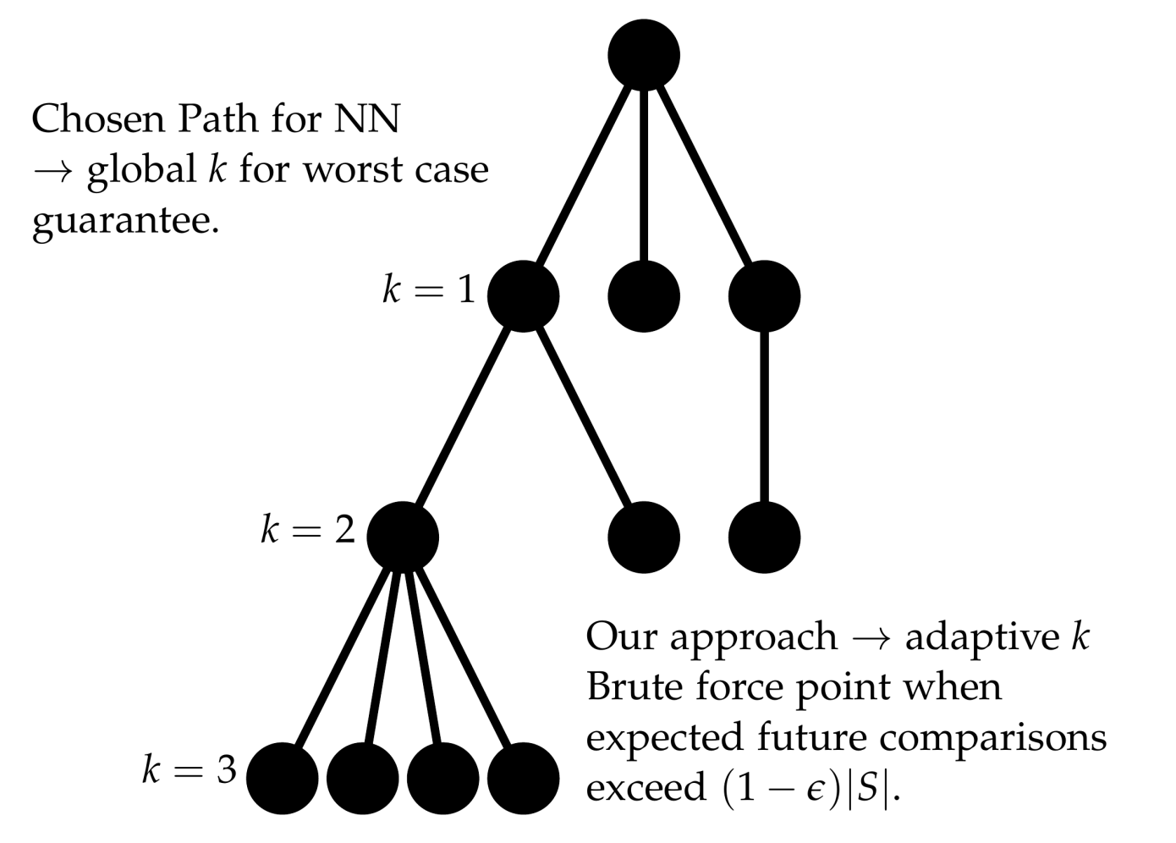
Chosen Path for NN
→ global k for worst case guarantee.
$k = 1$
$k = 2$
$k = 3$
Our approach → adaptive k
Brute force point when expected future comparisons exceed $(1-\epsilon)|S|$.

Figure 3: We introduce an adaptive technique for stopping.

Algorithm 1: CPSJoin(S, λ)

```
1  For $j \in [d]$ initialize $S_j \leftarrow \varnothing$.
2  $S \leftarrow$ BruteForce$(S, \lambda)$
3  $r \leftarrow$ SeedHashFunction$()$
4  for $x \in S$ do
5      for $j \in x$ do
6          if $r(j) < \frac{1}{\lambda|x|}$ then $S_j \leftarrow S_j \cup \{x\}$
7  for $S_j \neq \varnothing$ do CPSJoin$(S_j, \lambda)$
```

Algorithm 2: BruteForce(S, λ)

```
   Global parameters: limit $\geq 1$, $\varepsilon \geq 0$.
1  Initialize empty map count[] with default value $0$.
2  if $|S| \leq$ limit then
3      BruteForcePairs$(S, \lambda)$
4      return $\varnothing$
5  for $x \in S$ do
6      for $j \in x$ do
7          count[$j$] $\leftarrow$ count[$j$] + 1
8  for $x \in S$ do
9      if $\frac{1}{|S|-1} \sum_{j \in x}($count[$j$]$-1)/t > (1-\varepsilon)\lambda$ then
10         BruteForcePoint$(S, x, \lambda)$
11         return BruteForce$(S \setminus \{x\}, \lambda)$
12 return $S$
```

Correctness and analysis

The random process underlying the Chosen Path Tree belongs to the well studied class of Galton-Watson branching processes.Originally these where devised to answer questions about the growth and decline of family names in a model of population growth assuming i.i.d. offspring for every member of the population across generations [4]. In order to make statements about the properties of the CPSJoin algorithm we study in turn the branching processes of the Chosen Path Tree associated with a point x, a pair of points (x, y), and a set of points S. Note that we use the same random hash functions for different points in S.

Theorem 2. *For every LSHable similarity measure and every choice of constant threshold $\lambda \in (0,1)$ and probability of recall $\varphi \in (0,1)$ we can solve the (λ, φ)-set similarity join problem on every set S of n points using working space $\tilde{O}(n)$ and with expected running time*

$$\tilde{O}\left(\sum_{x \in S} \min_{k_x} \left(\sum_{y \in S \setminus \{x\}} (\mathrm{sim}(x,y)/\lambda)^{k_x} + (1/\lambda)^{k_x} \right) \right).$$

Experimental Results

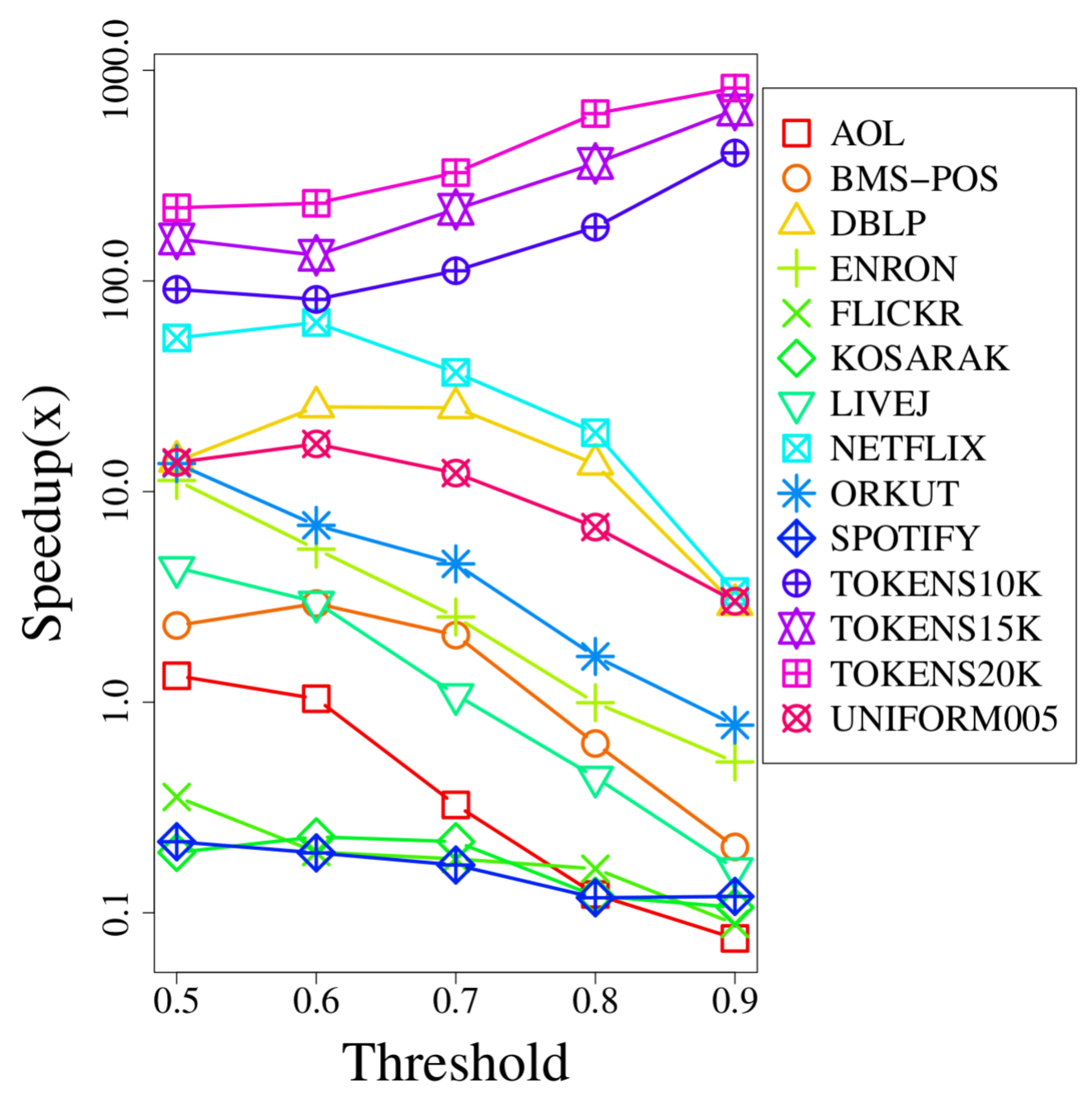

Figure 4: Join time speedup over AllPairs at 90% recall.

We compare CPSJoin to the exact set similarity join algorithms in the comprehensive empirical evaluation of Mann et al. [3], using the same data sets, and to other approximate set similarity join methods suggested in the literature. We find that CPSJoin outperforms other approximate methods and scales better than exact methods when the sets are relatively large (100 tokens or more) and the similarity threshold is low (e.g. Jaccard similarity 0.5) where we see speedups of more than an order of magnitude at 90% recall. We note that NETFLIX and FLICKR represents two different data set archetypes. On average a token in the NETFLIX dataset appears in more than 5000 sets while on average a token in the FLICKR dataset appears in less than 20 sets. Our experiment indicate that CPSJoin brings large speedups to the NETFLIX type datasets, while AllPairs exploits the presence of rare tokens to perform better on the FLICKR type.

References

[1] Tobias Christiani and Rasmus Pagh. Set similarity search beyond minhash. In *Proc. 49th Symp. on Theory of Computing (STOC)*, 2017.

[2] Ping Li and Arnd Christian König. Theory and applications of b-bit minwise hashing. *Comm. of the ACM*, 54(8):101–109, 2011.

[3] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. An empirical evaluation of set similarity join techniques. *Proc. VLDB Endowment*, 9(9):636–647, 2016.

[4] H. W. Watson and Francis Galton. On the probability of the extinction of families. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 4:138–144, 1875.

Acknowledgements

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.