TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

LSH-Based Probabilistic Pruning of Inverted Indices for Sets and Ranked Lists

Koninika Pal and Sebastian Michel

pal@cs.uni-kl.de
smichel@cs.uni-kl.de

TU Kaiserslautern, Germany

Introduction

- Top-k Rankings, Preference lists

Bloomberg ▼ | Bloomberg Billionai

Rank	Name	Total net worth ▼
1	Bill Gates	$87.2B
2	Jeff Bezos	$83.3B
3	Amancio Ortega	$82.5B
4	Warren Buffett	$73.4B
5	Mark Zuckerberg	$64.5B
6	Carlos Slim	$58.9B
7	Bernard Arnault	$50.9B
8	Larry Ellison	$48.3B

IMDb | Find Movies, TV shows, Celebrities and | Movies, TV & Showtimes | Celebs, Events & Photos

Top 100 Greatest Actors of All Time (The Ultimate List)

1.	Jack Nicholson	Actor, The Shining
2.	Marlon Brando	Actor, The Godfather
3.	Robert De Niro	Actor, Goodfellas
4.	Al Pacino	Actor, The Godfather
5.	Daniel Day-Lewis	Actor, There Will Be Blood

- Top-k Rankings, Preference lists
- Some applications:
  - Finding similar queries by results,
  - mining relations between entities,
  - recommender system, e.g. business promotion, etc.
- Similarity search over ranked lists or sets of preferences

Inverted Index

- Inverted index handles set similarity efficiently.

$$\tau_1 = [2, 5, 4, 3, 1]$$
$$\tau_2 = [1, 4, 7, 5, 2]$$
$$\tau_3 = [0, 8, 7, 5, 6]$$

$$1 \rightarrow \langle \tau_1 \rangle, \langle \tau_2 \rangle$$
$$2 \rightarrow \langle \tau_1 \rangle, \langle \tau_2 \rangle$$
$$3 \rightarrow \langle \tau_1 \rangle$$
$$\ldots\ldots$$

- Filter: look up inverted index for each elements from query and collect candidates.
- Validate: calculate distance between the candidates and the query

Motivation

Higher similarity → more overlapping elements

↓

Using multiple elements as key → more precision

↓

Simple index

$1 \rightarrow \langle \tau_1 \rangle, \langle \tau_2 \rangle$

$2 \rightarrow \langle \tau_1 \rangle, \langle \tau_2 \rangle$

$3 \rightarrow \langle \tau_1 \rangle$

. . . . . .

Pairwise index

$(1,2) \rightarrow \langle \tau_1 \rangle, \langle \tau_2 \rangle$

$(1,3) \rightarrow \langle \tau_1 \rangle$

$(2,3) \rightarrow \langle \tau_1 \rangle$

. . . . . .

Number of Key Increases exponentially

» Increase size of index structure

» Increase look up at query time

query size 10 → 10 keys from query : 45 access keys from query

More similarity ⟶ more overlapping elements

How do we prune the index?
How do we measure the effect of pruning in similarity search?

Increase size of index structure
Increase look up at query time

More similarity ⟶ more overlapping elements

How do we prune the index?
How do we measure the effect of pruning in similarity search?

Key idea: Connecting index structure with locality sensitive hashing (LSH)

Problem Description

- Collection $\mathcal{T}$ of sets τ_i of size k

$$\tau_i = [2, 5, 4, 3]$$

- Input at query time:
  - A query q of size k
  - A distance threshold θ
- Set similarity: Compute

$$\mathcal{R} = \{\tau_i | \tau_i \in \mathcal{T} \text{ and } d(\tau_i, q) \leq \theta\}$$

$d(\tau_i, q)$ = dissimilarity measure between τ_i, q

$\mathcal{R}$ = Result set while using complete index structure

- Index pruning factor ϕ
- Query on pruned index return result set $\mathcal{R}_p$
  - $\mathcal{R}_p \subseteq \mathcal{R}$
- Additional Input to similarity search
  - Recall threshold ϱ

Objective:

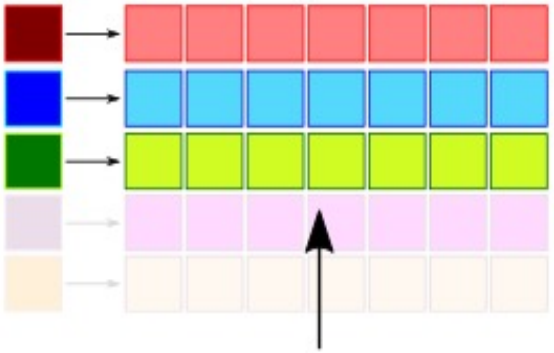
$$\begin{aligned} & maximize && \phi \\ & \text{subject to} && \mathcal{R}_p/\mathcal{R} \geq \varrho \end{aligned}$$

Content

- Motivation & Problem
- Pruning of Inverted Index
- Query Processing on Pruned index
- Experimental Results
- Conclusions

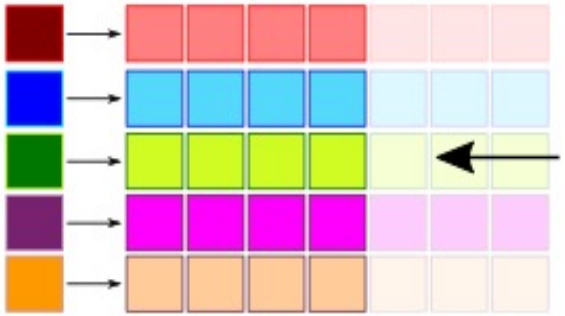
Pruning of Index Structure

Horizontal Pruning



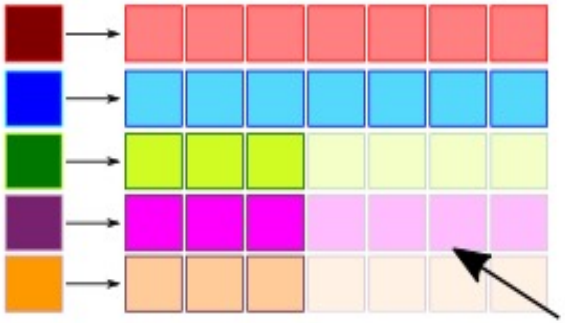
Randomly select factor ϕ of keys and delete the complete entry.

Vertical Pruning



Randomly delete factor ϕ of elements in each posting lists.

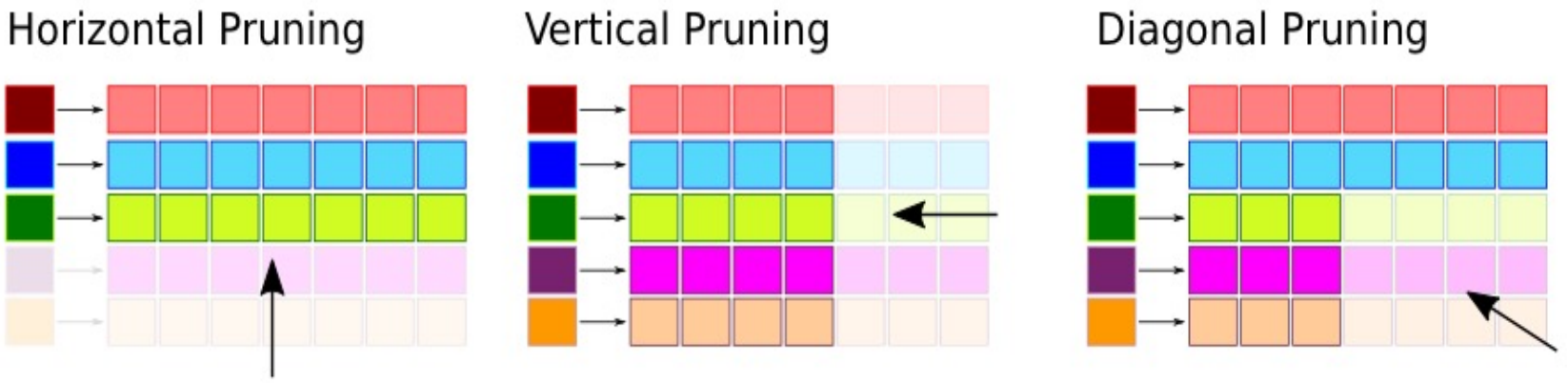
Diagonal Pruning



Randomly delete factor ϕ of elements from each sets and then build the index.

Pruning of Index Structure

- Similarity with document search:
  - Horizontal pruning: stop-word removal.
  - Vertical pruning: term-based index pruning (scoring model: tf-idf, KL-divergence etc.).
  - Diagonal pruning: document-centric pruning.



Pruning of Index Structure

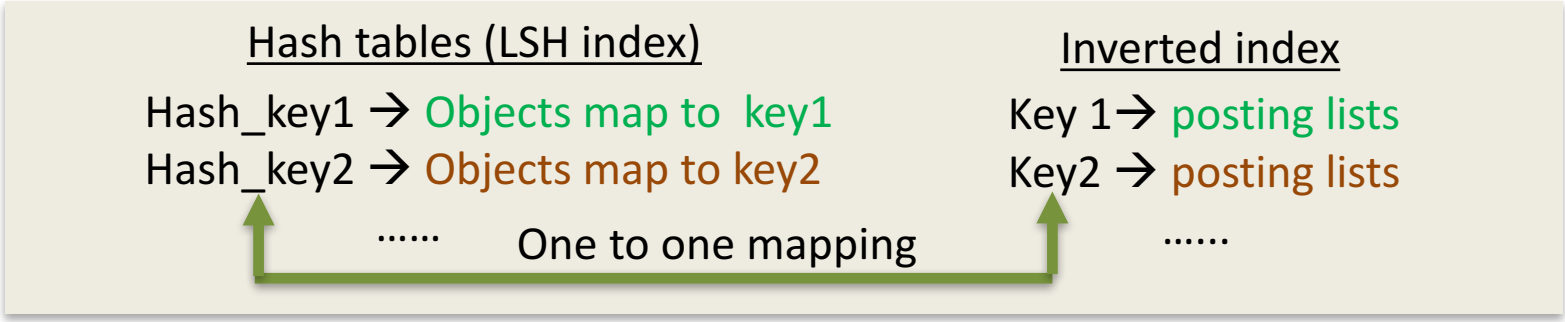
- Similarity with document search:
  - Horizontal pruning: stop-word removal.
  - Vertical pruning: term-based index pruning (scoring model: tf-idf, KL-divergence etc.).
  - Diagonal pruning: document-centric pruning.

- Contrast with common document retrieval:
  - Same size of query and documents.
  - Does not use score-based document search method.

Content

- Motivation & Problem
- Pruning of Inverted Index
- Query Processing on Pruned index
- Experimental Results
- Conclusions

Connecting Index with LSH Family



Example: projects sets on single elements

$$h_x : h_x(\tau_i) = x \ if \ x \in \tau_i \qquad h_7(\tau_2) = 7$$

$$\tau_2 = [1, 4, 7, 5, 2]$$
$$\tau_3 = [0, 8, 7, 5, 6]$$

$$h_7 : 7 \rightarrow \langle \tau_2 \rangle, \langle \tau_3 \rangle \iff 7 \rightarrow \langle \tau_2 \rangle, \langle \tau_3 \rangle$$

– Multiple hash functions are possible to use conjunctively in LSH

$$h_7, h_5 : (7, 5) \rightarrow \langle \tau_2 \rangle, \langle \tau_3 \rangle \iff (7, 5) \rightarrow \langle \tau_2 \rangle, \langle \tau_3 \rangle$$

Properties of LSH

- Why LSH?
  - Similar objects have higher probability to collide into same bucket
  - Tuning of number of index entries(l) are needed to look up to reach recall ϱ

$$\varrho = 1 - (1 - P_1^m)^l$$

- What we need?
  - Collision probability of hash function: P_1
  - Number of hash functions are used at query time.

Query Processing on Pruned Index

- Pruning of index -> dropping objects from LSH index
- Missing collision at query processing:
  - Objects and query are not similar
  - Objects are dropped due to pruning
- Access more entries (l) than the LSH method required.
- How many extra index look up are required?

Ad-hoc Query Processing

- Continue index look up until l successful accesses. $\varrho = 1 - (1 - P_1^m)^l$
- Max. lookup → look up all keys from query

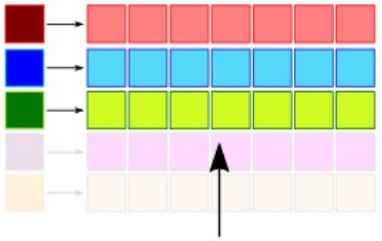
- Expected look ups: $E[l] = (1/f) \cdot l$
- Modifying factor in collision probability: f

Probabilistic Query Processing

- Find modified collision probability
- Find required modified number of accesses l_Y

$$\varrho = 1 - (1 - f_Y \cdot P_1^m)^{l_Y}$$

- Modifying factor f_Y = $function(\phi)$
  - Modifying factor of horizontal pruning f_h
    - ϕ faction of index pruning
    - → removing ϕ faction of keys
    - f_h = 1 – ϕ



Optimizing the Pruning Factor

- Max. lookup → look up all keys from query $\binom{k}{t}$
- Look up l_Y is bound by $\binom{k}{t}$
- Number of access (l_Y): $\varrho = 1 - (1 - f_Y \cdot P_1^m)^{l_Y}$
- Modifying factor f_Y = $function(\phi)$

$$\phi^* = argmax_\phi \left\{ \binom{k}{t} - l_Y = 0 \right\}$$

Case Studies

- Case 1: Jaccard Distance over sets
  - Use pairwise index
  - Relate LSH[1] index to pairwise index

$$P_1 = \frac{2\theta}{1+\theta}\ , m = 2$$

$$\varrho = 1 - (1 - P_1^m)^l$$

- Case 2: Kendall's Tau Distance over rankings

[1] Koninika Pal and Sebastian Michel. Efficient Similarity Search across Top-k Lists under the Kendall's Tau Distance. In SSDBM 2016.

Content

- Motivation & Problem
- Pruning of Inverted Index
- Query Processing on Pruned index
- Experimental Results
- Conclusions

Experimental Setup

- Datasets:
  - **LifeJ**: 100,000 profiles from Live Journal; truncated to set size = 20.
  - **Yago**: 25,000 top-20 rankings; Wikipedia based.
- 5 consecutive experimental runs over 1000 queries.
- Recall threshold ϱ = 99%
- Baseline approach: The plain LSH methods on the non-pruned index structures.
- Full scan and prefix filtering[2] method in simple index retrieve candidates > 5 times than the baseline approach.

[2] jiannan Wang, Guoliang Li, and Jianhua Feng. 2012. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In SIGMOD.

Theoretically Established Parameters for LiveJ

	not pruned	Horizontal pruning			Vertical pruning			Diagonal pruning		
θ	l	ϕ^*	l_h	$E[l_h]$	ϕ^*	l_v	$E[l_v]$	ϕ^*	l_d	$E[l_d]$
0.1	2	0.8	125	10	0.8	125	10	0.5	95	8.6
0.3	4	0.8	167	20	0.8	167	20	0.5	126	17.4
0.5	8	0.7	112	26.6	0.7	112	26.6	0.4	87	23.5

ϕ^* : Optimal pruning factor. Y : { h / v / d }

l_Y : Number of scan for probabilistic query processing.

$E[l_Y]$: Expected number of scan for l successful scan.

Experimental Results for Probabilistic Query Processing on LifeJ

Pruning method	θ	Time (ms)	#candidates	recall	#successful scan	l_Y	Baseline candidates
Horizontal	0.1	11.17	10031.3	100	24.6	125	5105.3
	0.3	11.54	13257.0	100	33.9	167	7360.4
	0.5	13.39	14452.2	100	33.6	112	9059.5
Vertical	0.1	14.0	11252.9	100	125	125	5105.3
	0.3	9.8	12208.7	100	167	167	7360.4
	0.5	11.0	14001.9	100	112	112	9059.5
Diagonal	0.1	10.38	10378.3	99.5	79.69	95	5105.3
	0.3	11.06	11512.7	100	104.58	126	7360.4
	0.5	11.32	13003.1	99.7	76.84	87	9059.5

Experimental Results for Ad-hoc Query Processing on LifeJ

Pruning method	θ	Time (ms)	#candidates	recall	#Total scan	$E[l_Y]$	Baseline candidates
Horizontal	0.1	3.4	3806.7	100	9.45	10	5105.3
	0.3	4.4	5163.3	100	19.39	20	7360.4
	0.5	7.4	7822.5	99.9	26.29	26.6	9059.5
Vertical	0.1	1.03	1142.2	51.3	2	10	5105.3
	0.3	1.67	1926.9	64.3	4	20	7360.4
	0.5	4.08	3998.9	93.6	8	26.6	9059.5
Diagonal	0.1	1.24	1309.1	37.3	2.63	8.6	5105.3
	0.3	1.99	2098.4	47.3	4.94	17.4	7360.4
	0.5	3.52	3938.9	61.0	9.61	23.5	9059.5

Content

- Motivation & Problem
- Pruning of Inverted Index
- Query Processing on Pruned index
- Experimental Results
- Conclusions

Conclusions

- Saving upto 80% index pruning for small distance threshold, $\theta \leqslant 0.3$
- Probabilistic query processing ensures recall requirement.
- Ad-hoc query processing in horizontal pruning outperforms.

- Future directions:
  - Combining the proposed approach with compression techniques.
  - Analysis over non-randomized, application driven pruning.

Thank you

Theoretically Established Parameters for Yago

	not pruned	Horizontal pruning			Vertical pruning			Diagonal pruning		
θ	l	ϕ^*	l_h	$E[l_h]$	ϕ^*	l_v	$E[l_v]$	ϕ^*	l_d	$E[l_d]$
0.1	2	0.8	125	10	0.8	125	10	0.5	95	8.6
0.3	4	0.8	167	20	0.8	167	20	0.5	126	17.4
0.5	8	0.7	112	26.6	0.7	112	26.6	0.4	87	23.5

ϕ^* : Optimal pruning factor. Y : { h / v / d }

l_Y : Number of scan for probabilistic query processing.

$E[l_Y]$: Expected number of scan for l successful scan.

Extra slides for more experimental data

Experimental Results for Probabilistic Query Processing on Yago

Pruning method	θ	Time (ms)	#candidates	recall	#successful scan	l_Y	Baseline candidates
Horizontal	0.1	11.17	10031.3	100	24.6	125	5105.3
	0.3	11.54	13257.0	100	33.9	167	7360.4
	0.5	13.39	14452.2	100	33.6	112	9059.5
Vertical	0.1	14.0	11252.9	100	125	125	5105.3
	0.3	9.8	12208.7	100	167	167	7360.4
	0.5	11.0	14001.9	100	112	112	9059.5
Diagonal	0.1	10.38	10378.3	99.5	79.69	95	5105.3
	0.3	11.06	11512.7	100	104.58	126	7360.4
	0.5	11.32	13003.1	99.7	76.84	87	9059.5

Extra slides for more experimental data

Experimental Results for Ad-hoc Query Processing on Yago

Pruning method	θ	Time (ms)	#candidates	recall	#Total scan	$E[l_Y]$	Baseline candidates
Horizontal	0.1	3.4	3806.7	100	9.45	10	5105.3
	0.3	4.4	5163.3	100	19.39	20	7360.4
	0.5	7.4	7822.5	99.9	26.29	26.6	9059.5
Vertical	0.1	1.03	1142.2	51.3	2	10	5105.3
	0.3	1.67	1926.9	64.3	4	20	7360.4
	0.5	4.08	3998.9	93.6	8	26.6	9059.5
Diagonal	0.1	1.24	1309.1	37.3	2.63	8.6	5105.3
	0.3	1.99	2098.4	47.3	4.94	17.4	7360.4
	0.5	3.52	3938.9	61.0	9.61	23.5	9059.5

Extra slides for more experimental data