MATE: Multi-Attribute Table Extraction

Mahdi Esmailoghli
Leibniz Universität Hannover & L3S Research Center
Hannover, Germany
esmailoghli@dbs.uni-hannover.de

Jorge-Arnulfo Quiané-Ruiz
TU Berlin
Berlin, Germany
jorge.quiane@tu-berlin.de

Ziawasch Abedjan
Leibniz Universität Hannover & L3S Research Center
Hannover, Germany
abedjan@dbs.uni-hannover.de

Darshan Patel

Nency Patel

Ajay Babu Popuri

College of Computing

Illinois Institute of Technology

1st Dec 2023

CS 520 & Data Integration, Warehousing, and Provenance

Dr. Boris Glavic

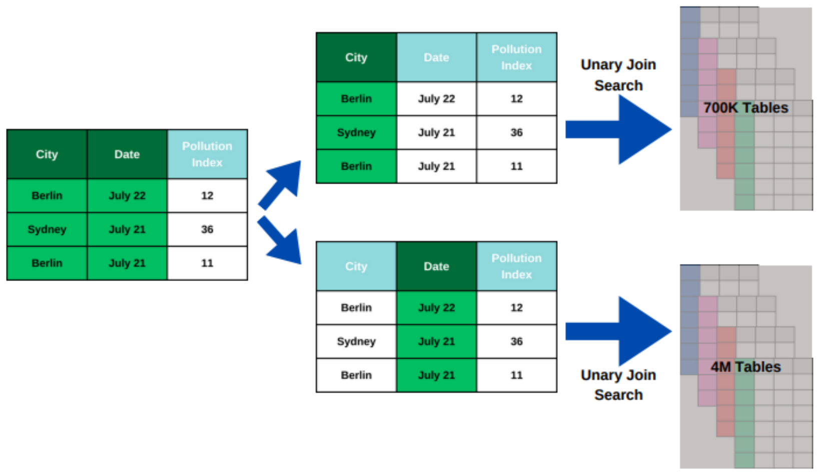
Contents

1. Abstract

"MATE: Multi-Attribute Table Extraction" introduces a table discovery system, which addresses the limitations of existing works on indexing and retrieving joinable tables from large corpora. Current systems are restricted to unary joins, limiting their application and efficiency for prevalent datasets with composite join keys. Mate as a solution to this challenge, leveraging a novel hash-based index to efficiently discover joinable tables, even in the presence of n-ary join keys. Filtering layer (XASH) using hash function which encode Mate can prune false positives up to 1000x more effectively, leading to over 60x faster table discovery compared to existing state-of-the-art systems. MATE presents a significant advancement in the field of table discovery systems, addressing the limitations of existing approaches and providing a more efficient solution for discovering joinable tables with n-ary join keys and multi-column combinations.

2. List of Figures-

- Unary join search



City	Date	Pollution Index
Berlin	July 22	12
Sydney	July 21	36
Berlin	July 21	11

City	Date	Pollution Index
Berlin	July 22	12
Sydney	July 21	36
Berlin	July 21	11

City	Date	Pollution Index
Berlin	July 22	12
Sydney	July 21	36
Berlin	July 21	11

- MultiAttribute Inverted Index

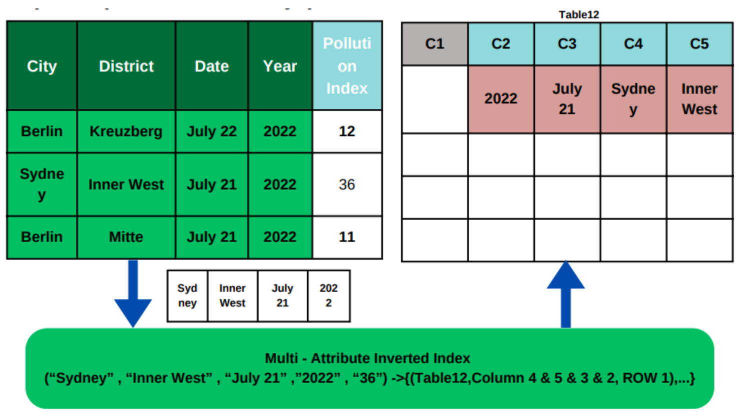


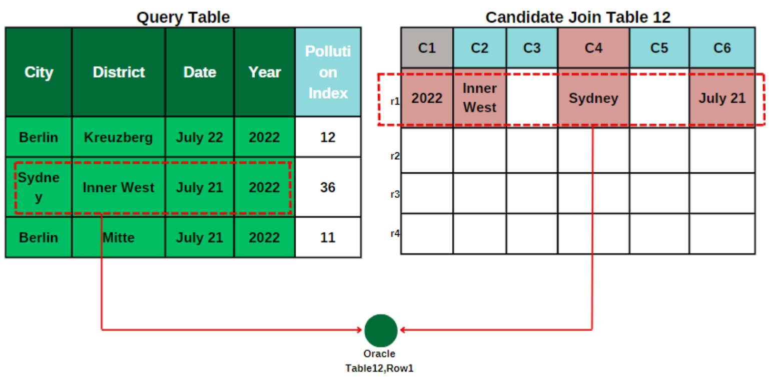
Table12
Multi - Attribute Inverted Index
("Sydney" , "Inner West" , "July 21" ,"2022" , "36") ->{(Table12,Column 4 & 5 & 3 & 2, ROW 1),...}

City	District	Date	Year	Pollution Index
Berlin	Kreuzberg	July 22	2022	12
Sydney	Inner West	July 21	2022	36
Berlin	Mitte	July 21	2022	11

Sydney	Inner West	July 21	2022

C1	C2	C3	C4	C5
	2022	July 21	Sydney	Inner West

- Superkey



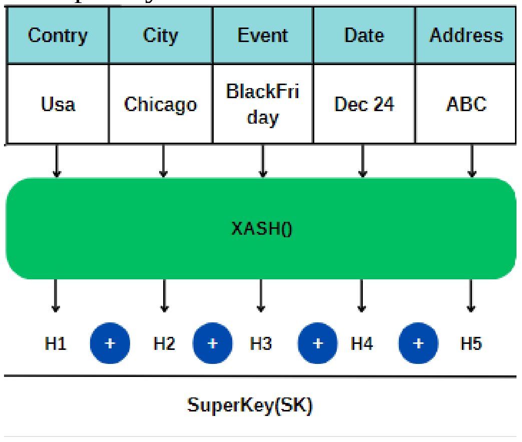
Query Table

City	District	Date	Year	Pollution Index
Berlin	Kreuzberg	July 22	2022	12
Sydney	Inner West	July 21	2022	36
Berlin	Mitte	July 21	2022	11

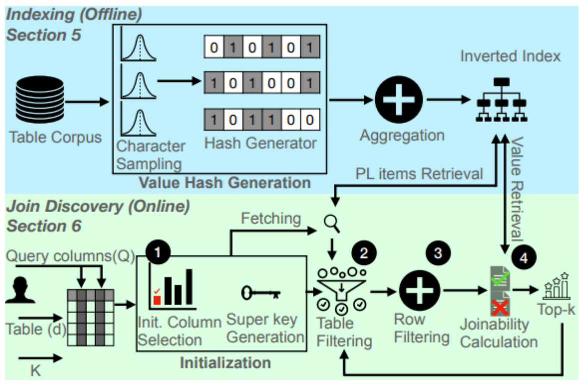
Candidate Join Table 12

	C1	C2	C3	C4	C5	C6
r1	2022	Inner West		Sydney		July 21
r2						
r3						
r4						

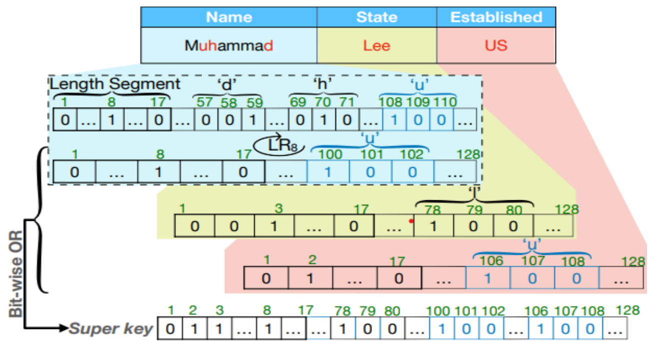
- Superkey Generation



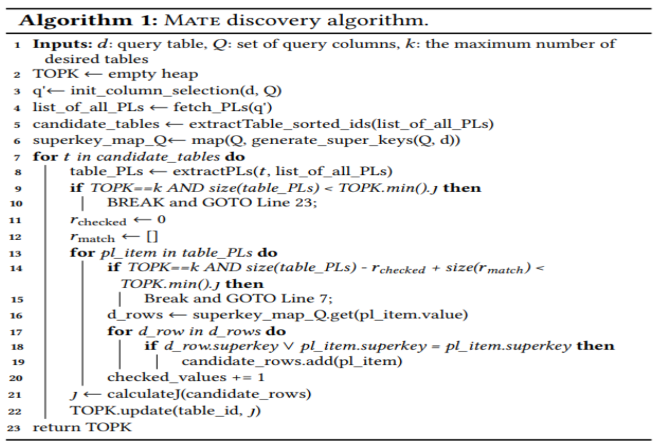
- Join Discovery phase And Indexing phase of the System:



- A diagram of how Xash functions and how the data is combined



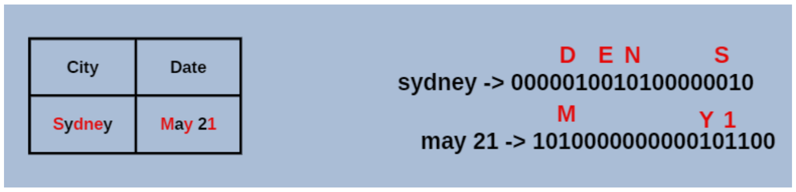
- Mate Discovery Algorithm



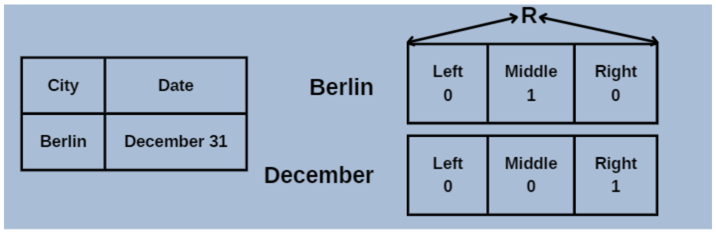
Algorithm 1: MATE discovery algorithm.

```
1  Inputs: $d$: query table, $Q$: set of query columns, $k$: the maximum number of
     desired tables
2  TOPK ← empty heap
3  q'← init_column_selection(d, Q)
4  list_of_all_PLs ← fetch_PLs(q')
5  candidate_tables ← extractTable_sorted_ids(list_of_all_PLs)
6  superkey_map_Q← map(Q, generate_super_keys(Q, d))
7  for t in candidate_tables do
8      table_PLs ← extractPLs(t, list_of_all_PLs)
9      if TOPK==k AND size(table_PLs) < TOPK.min().J then
10         BREAK and GOTO Line 23;
11     $r_{checked}$ ← 0
12     $r_{match}$ ← []
13     for pl_item in table_PLs do
14         if TOPK==k AND size(table_PLs) - $r_{checked}$ + size($r_{match}$) <
             TOPK.min().J then
15             Break and GOTO Line 7;
16         d_rows ← superkey_map_Q.get(pl_item.value)
17         for d_row in d_rows do
18             if d_row.superkey ∨ pl_item.superkey = pl_item.superkey then
19                 candidate_rows.add(pl_item)
20         checked_values += 1
21     J ← calculateJ(candidate_rows)
22     TOPK.update(table_id, J)
23 return TOPK
```

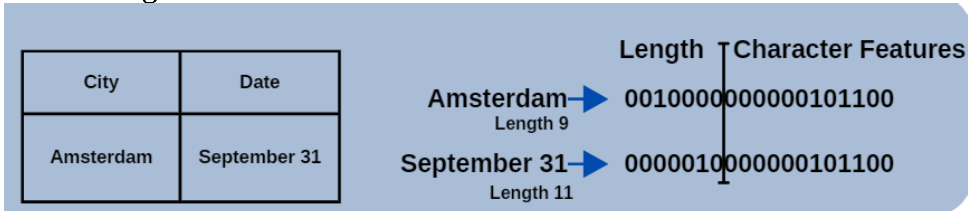
- Rare Characters


City
Date
Sydney
May 21
D E N S
sydney -> 00000100101000000010
M Y 1
may 21 -> 10100000000000101100

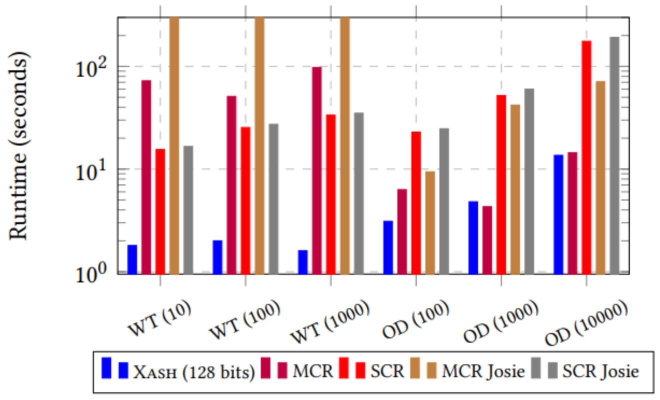
- Character Positions



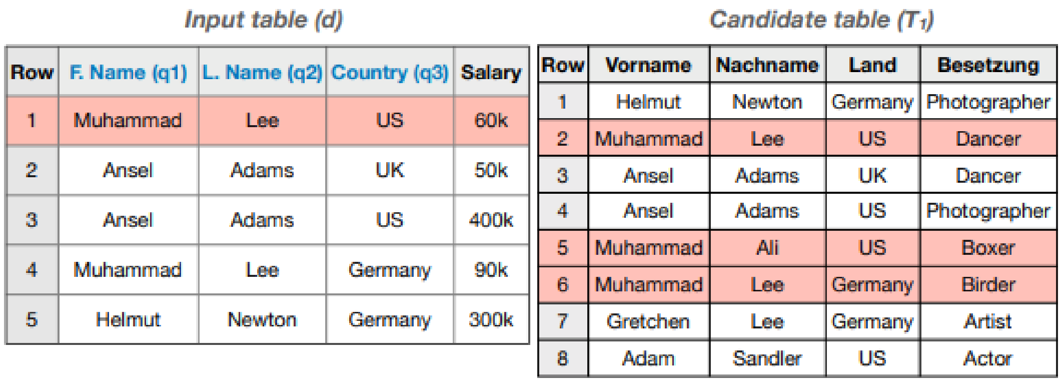
- Value Length



- Table for Experiment



3. List Of Tables-



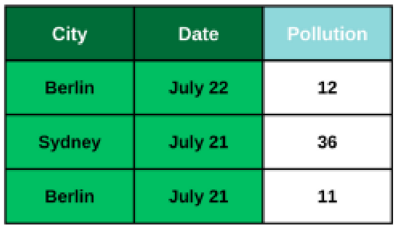
Input table (d)

Row	F. Name (q1)	L. Name (q2)	Country (q3)	Salary
1	Muhammad	Lee	US	60k
2	Ansel	Adams	UK	50k
3	Ansel	Adams	US	400k
4	Muhammad	Lee	Germany	90k
5	Helmut	Newton	Germany	300k

Candidate table (T_1)

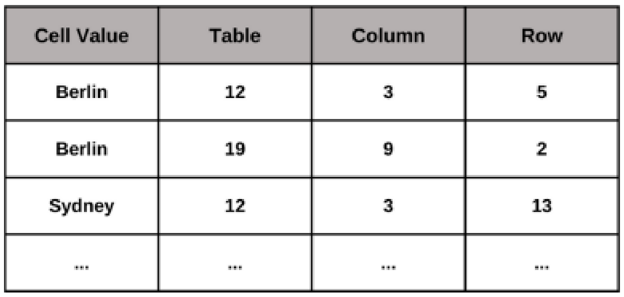
Row	Vorname	Nachname	Land	Besetzung
1	Helmut	Newton	Germany	Photographer
2	Muhammad	Lee	US	Dancer
3	Ansel	Adams	UK	Dancer
4	Ansel	Adams	US	Photographer
5	Muhammad	Ali	US	Boxer
6	Muhammad	Lee	Germany	Birder
7	Gretchen	Lee	Germany	Artist
8	Adam	Sandler	US	Actor

Running Example



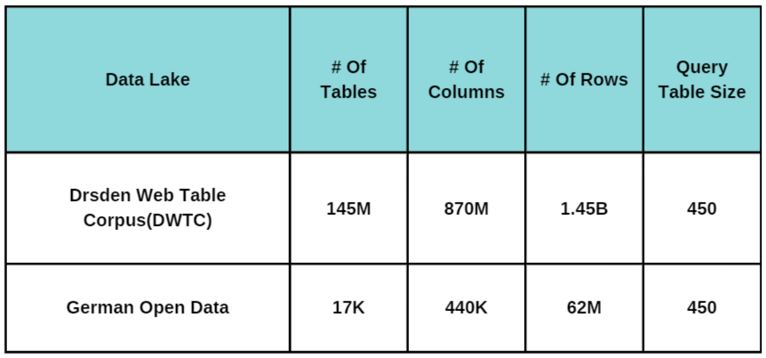
City	Date	Pollution
Berlin	July 22	12
Sydney	July 21	36
Berlin	July 21	11

sample query table



Cell Value	Table	Column	Row
Berlin	12	3	5
Berlin	19	9	2
Sydney	12	3	13
...	...	...	...

sample inverted index

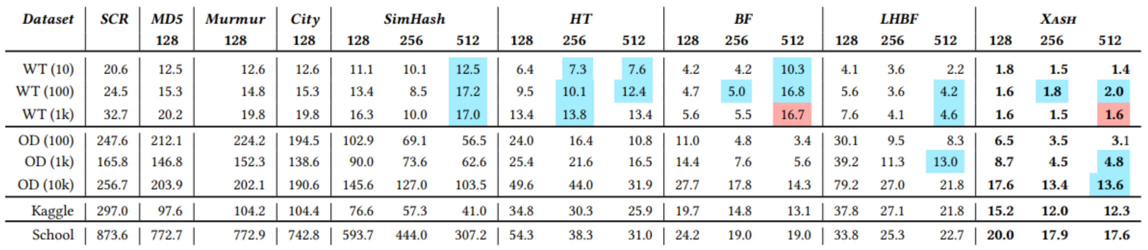


Data Lake	# Of Tables	# Of Columns	# Of Rows	Query Table Size
Drsden Web Table Corpus(DWTC)	145M	870M	1.45B	450
German Open Data	17K	440K	62M	450

Tables for Experiments

- A comparison of Mate's runtime using various hash functions, including Xash

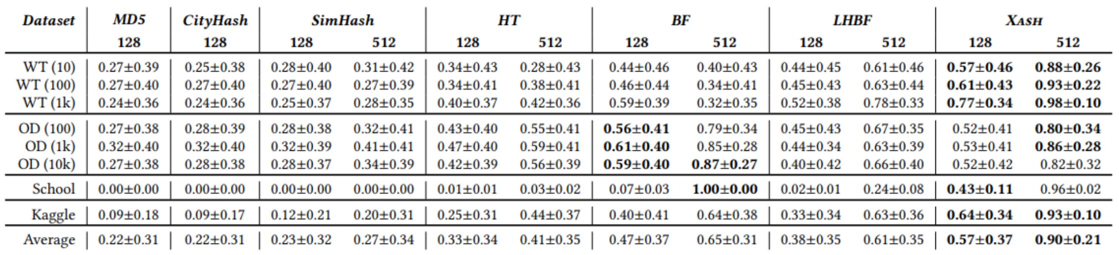
Table 2: Runtime experiment (seconds). blue cells represent the experiments where the larger hash performs worse. Red cells show the maximum performance gain of MATE over BF.



Dataset	*SCR*	*MD5*	*Murmur*	*City*	*SimHash*			*HT*			*BF*			*LHBF*			*XASH*		
		128	**128**	**128**	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**
WT (10)	20.6	12.5	12.6	12.6	11.1	10.1	12.5	6.4	7.3	7.6	4.2	4.2	10.3	4.1	3.6	2.2	**1.8**	**1.5**	**1.4**
WT (100)	24.5	15.3	14.8	15.3	13.4	8.5	17.2	9.5	10.1	12.4	4.7	5.0	16.8	5.6	3.6	4.2	**1.6**	**1.8**	**2.0**
WT (1k)	32.7	20.2	19.8	19.8	16.3	10.0	17.0	13.4	13.8	13.4	5.6	5.5	16.7	7.6	4.1	4.6	**1.6**	**1.5**	**1.6**
OD (100)	247.6	212.1	224.2	194.5	102.9	69.1	56.5	24.0	16.4	10.8	11.0	4.8	3.4	30.1	9.5	8.3	**6.5**	**3.5**	**3.1**
OD (1k)	165.8	146.8	152.3	138.6	90.0	73.6	62.6	25.4	21.6	16.5	14.4	7.6	5.6	39.2	11.3	13.0	**8.7**	**4.5**	**4.8**
OD (10k)	256.7	203.9	202.1	190.6	145.6	127.0	103.5	49.6	44.0	31.9	27.7	17.8	14.3	79.2	27.0	21.8	**17.6**	**13.4**	**13.6**
Kaggle	297.0	97.6	104.2	104.4	76.6	57.3	41.0	34.8	30.3	25.9	19.7	14.8	13.1	37.8	27.1	21.8	**15.2**	**12.0**	**12.3**
School	873.6	772.7	772.9	742.8	593.7	444.0	307.2	54.3	38.3	31.0	24.2	19.0	19.0	33.8	25.3	22.7	**20.0**	**17.9**	**17.6**

- Precision Experiment

Table 3: Precision experiment.



Dataset	*MD5*	*CityHash*	*SimHash*		*HT*		*BF*		*LHBF*		*XASH*	
	128	**128**	**128**	**512**	**128**	**512**	**128**	**512**	**128**	**512**	**128**	**512**
WT (10)	0.27±0.39	0.25±0.38	0.28±0.40	0.31±0.42	0.34±0.43	0.28±0.43	0.44±0.46	0.40±0.43	0.44±0.45	0.61±0.46	**0.57±0.46**	**0.88±0.26**
WT (100)	0.27±0.40	0.27±0.40	0.27±0.40	0.27±0.39	0.34±0.41	0.38±0.41	0.46±0.44	0.34±0.41	0.45±0.43	0.63±0.44	**0.61±0.43**	**0.93±0.22**
WT (1k)	0.24±0.36	0.24±0.36	0.25±0.37	0.28±0.35	0.40±0.37	0.42±0.36	0.59±0.39	0.32±0.35	0.52±0.38	0.78±0.33	**0.77±0.34**	**0.98±0.10**
OD (100)	0.27±0.38	0.28±0.39	0.28±0.38	0.32±0.41	0.43±0.40	0.55±0.41	**0.56±0.41**	0.79±0.34	0.45±0.43	0.67±0.35	0.52±0.41	**0.80±0.34**
OD (1k)	0.32±0.40	0.32±0.40	0.32±0.39	0.41±0.41	0.47±0.40	0.59±0.41	**0.61±0.40**	0.85±0.28	0.44±0.34	0.63±0.39	0.53±0.41	**0.86±0.28**
OD (10k)	0.27±0.38	0.28±0.38	0.28±0.37	0.34±0.39	0.42±0.39	0.56±0.39	**0.59±0.40**	**0.87±0.27**	0.40±0.42	0.66±0.40	0.52±0.42	0.82±0.32
School	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.01±0.01	0.03±0.02	0.07±0.03	**1.00±0.00**	0.02±0.01	0.24±0.08	**0.43±0.11**	0.96±0.02
Kaggle	0.09±0.18	0.09±0.17	0.12±0.21	0.20±0.31	0.25±0.31	0.44±0.37	0.40±0.41	0.64±0.38	0.33±0.34	0.63±0.36	**0.64±0.34**	**0.93±0.10**
Average	0.22±0.31	0.22±0.31	0.23±0.32	0.27±0.34	0.33±0.34	0.41±0.35	0.47±0.37	0.65±0.31	0.38±0.35	0.61±0.35	**0.57±0.37**	**0.90±0.21**

4. Introduction-

The new table discovery system, MATE, which can effectively handle n-ary join keys, is introduced by the authors. MATE efficiently prune non-joinable rows using a new hash function called Xash. The Xash hash function is based on the concept of hashing each attribute value independently and then combining the hash values to form a composite hash value for the entire row. It is intended to operate with n-ary join keys. Because of this, MATE can effectively remove non-joinable rows without requiring the table to be materialized in its entirety. MATE: Multi-Attribute Table Extraction paper highlights three major contributions of the proposed system.

Firstly, current systems are limited for unary joins which can not be efficient and this restricts makes application for accuracy for dataset with composite keys. Therefore filter -based method has been introduced in this MATE.

Secondly, the paper introduces Xash, which encodes the syntactic attributes of the key values to obtain hash results that optimally use a fixed bit space to avoid overlapping bits of similar join values. The hash function leads effectively filters non-joinable rows and fewer FP rates compared to the state-of-the-art hash functions and bloom filters. This approach enables efficient pruning of non-joinable rows and faster table discovery.

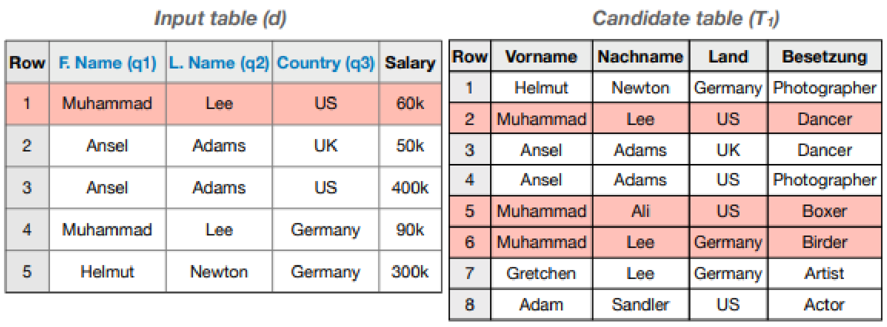
Thirdly, the paper introduces a two-tier filtering strategy that prunes candidate tables that cannot be part of top-k and candidate rows that are not joinable on all key columns without evaluating the actual row values. This strategy reduces the computational overhead of the system and enables faster table discovery.

In summary, the MATE: Multi-Attribute Table Extraction paper proposes a new table discovery system that efficiently handles n-ary join keys, leading to faster and more accurate table discovery. Introduction with Xash, a novel hash function that enables efficient pruning of non-joinable rows and faster table discovery.

5. Problem-Statements-

The goal of the paper MATE: Multi-Attribute Table Extraction is to identify joinable tables using composite key joins, overcoming the drawbacks of current indexing and retrieving techniques that are restricted to unary joins. Finding the top-k joinable tables for a given query table using a chosen composite key is how the authors define the table discovery problem. Below image shown how the join discovery works where query dataset will join search on data lakes and find joinable tables with join key.

The problem of n-ary join discovery from large data lakes and formalize the joinability between multiple tables.



Input table (d)

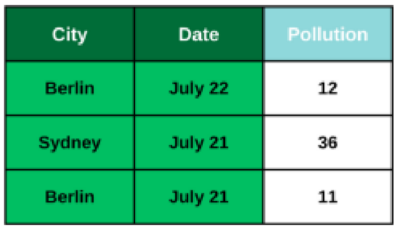
Row	F. Name (q1)	L. Name (q2)	Country (q3)	Salary
1	Muhammad	Lee	US	60k
2	Ansel	Adams	UK	50k
3	Ansel	Adams	US	400k
4	Muhammad	Lee	Germany	90k
5	Helmut	Newton	Germany	300k

Candidate table (T_1)

Row	Vorname	Nachname	Land	Besetzung
1	Helmut	Newton	Germany	Photographer
2	Muhammad	Lee	US	Dancer
3	Ansel	Adams	UK	Dancer
4	Ansel	Adams	US	Photographer
5	Muhammad	Ali	US	Boxer
6	Muhammad	Lee	Germany	Birder
7	Gretchen	Lee	Germany	Artist
8	Adam	Sandler	US	Actor

Fig 5.1 Running Example

For example, if we map F. Name from d to Vorname from $T1$, where L.Name to Nachname. If we mapped Country to Land , we would get The highest joinability score among all potential mappings.



City	Date	Pollution
Berlin	July 22	12
Sydney	July 21	36
Berlin	July 21	11

Fig 5.2: sample query table

Cell Value	Table	Column	Row
Berlin	12	3	5
Berlin	19	9	2
Sydney	12	3	13
...	...	...	...

Fig 5.3: sample inverted index

As shown above sample example, the issue arises when the input contains only composite keys because the most advanced joint algorithm uses only single attributes or a small number of unary keys, and we can only query the inverted index with single cell values. Therefor MATE is needed.

In order to find n-ary joinable tables, one must either (I) use the inverted index for unary joins and accept a high number of FPs, or (ii) create a Mult attribute inverted index that maps various combinations of cell values to their locations in the tables.

5.1 Composite (Multi-Attribute Keys) - Unary Join search

Limitations of existing systems that leverage inverted index with small alterations for single-attribute join keys. These systems retrieve the PL items for each value in the key column and the number of returned PL items represents the joinability score for each table. However, this approach leads to a large number of false positive rows that require a second verification step. A false positive row (FP row) is a row from a candidate table that only contains a subset of join attribute values.

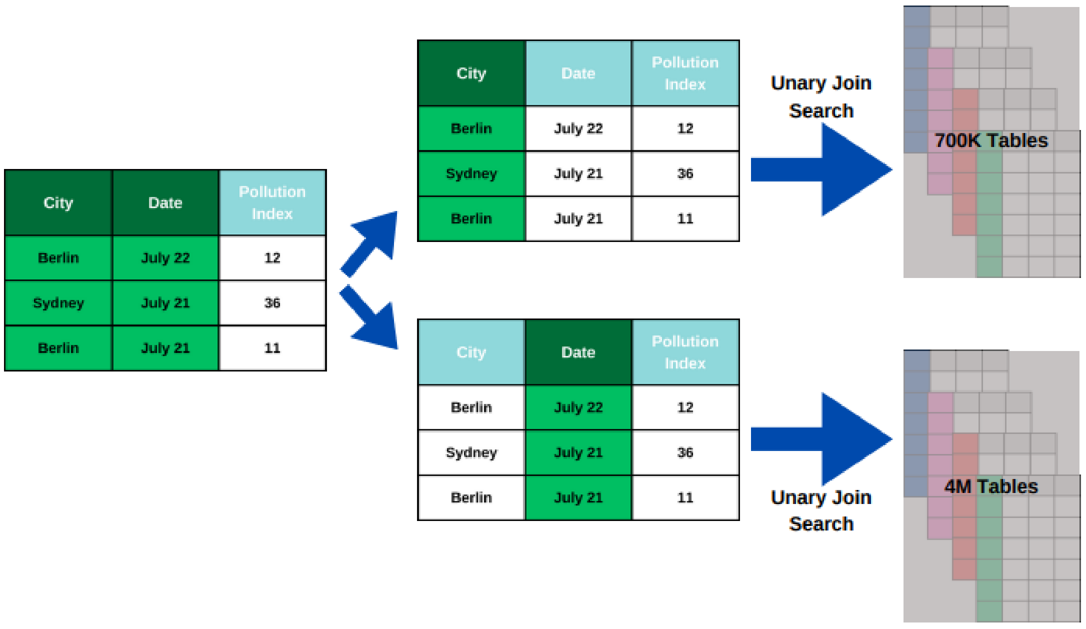
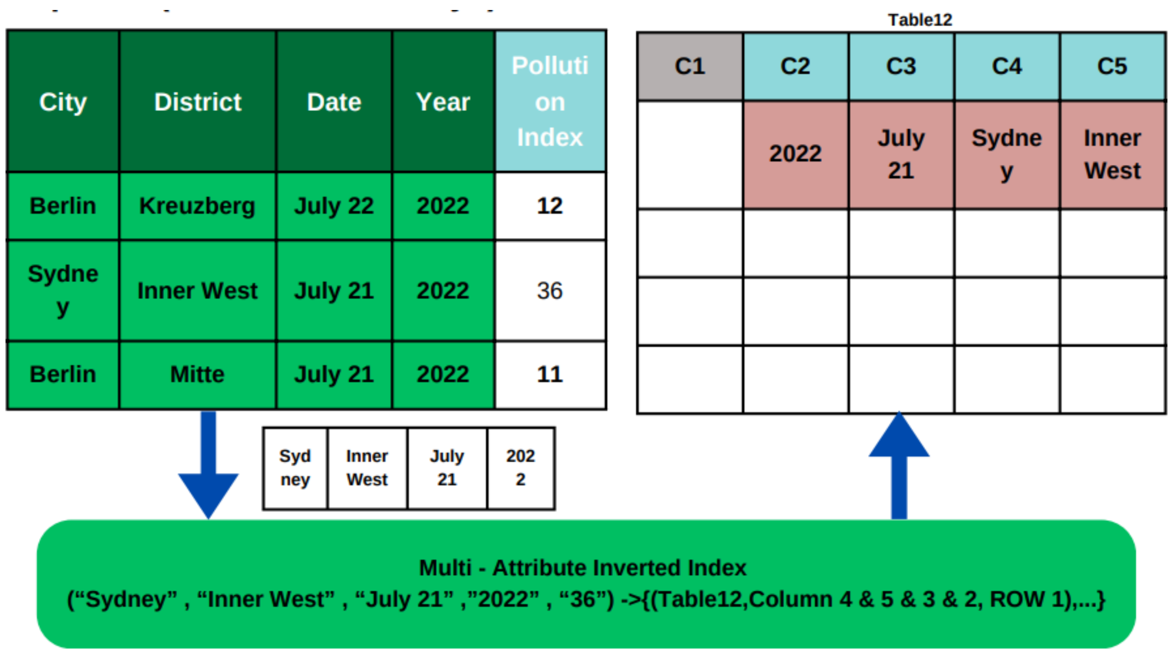


Fig 5.1 Unary join search

As mentioned above image shows that select one of the columns of composites keys and unary join search run on those tables. It will give filtered joinable tables of single column composite key. Each solution lead large number of false positives.

5.2 Composite (Multi-Attribute Keys) - Multi-Attribute Inverted Index

We can query inverted index with multiple attribute and inverted index trying to locate the location of those value inside corpus and we can find that joinable table but problem is multi-attribute inverted requires all combination of the values and that will lead exponential storage complexity. See example below shown where record for Sydney's location of those value in the multi attribute inverted index.



City	District	Date	Year	Pollution Index
Berlin	Kreuzberg	July 22	2022	12
Sydney	Inner West	July 21	2022	36
Berlin	Mitte	July 21	2022	11

Table12

C1	C2	C3	C4	C5
	2022	July 21	Sydney	Inner West

Sydney	Inner West	July 21	2022

Multi - Attribute Inverted Index
("Sydney" , "Inner West" , "July 21" ,"2022" , "36") ->{(Table12,Column 4 & 5 & 3 & 2, ROW 1),...}

Fig 5.2.1 Multi Attribute Inverted Index

Example for finding all possible combination of value, consider a query table *d* with columns A, B, and C, and a candidate table *T* with columns X, Y, Z, A, B, and C. To calculate the joinability score between *d* and *T*, we need to find the mapping between the query columns and the candidate columns that maximizes the joinability score. There are 6 possible mappings: (A, B, C), (A, B), (A, C), (B, C), (A), and (B). The number of possible mappings is calculated as

(6, 3) = 6! / (6-3)!3!
= 20.

6. System Overview

The abstract workflow of the Mate system is depicted in below figure. It consists of two phases: the offline phase and the online phase.

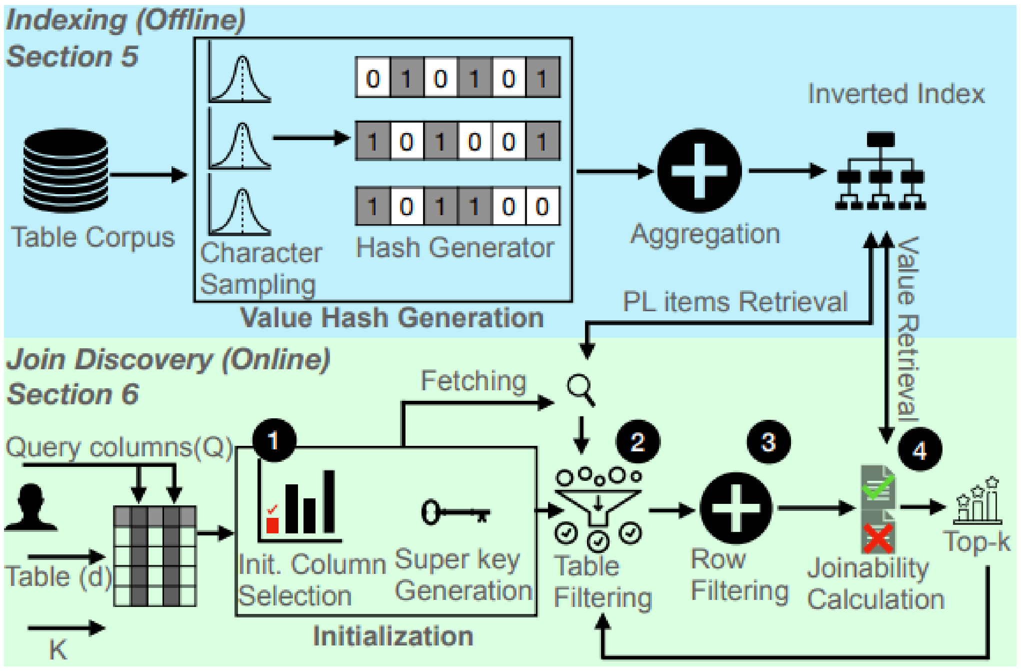


Fig 6.1 System Mate Overview

Overall, Mate efficiently discovers n-ary joinable tables by using a pruning technique, selecting an initial query column, generating a super key entry, and applying table and row filtering to find the most relevant tables for a given query table.

6.1 Background

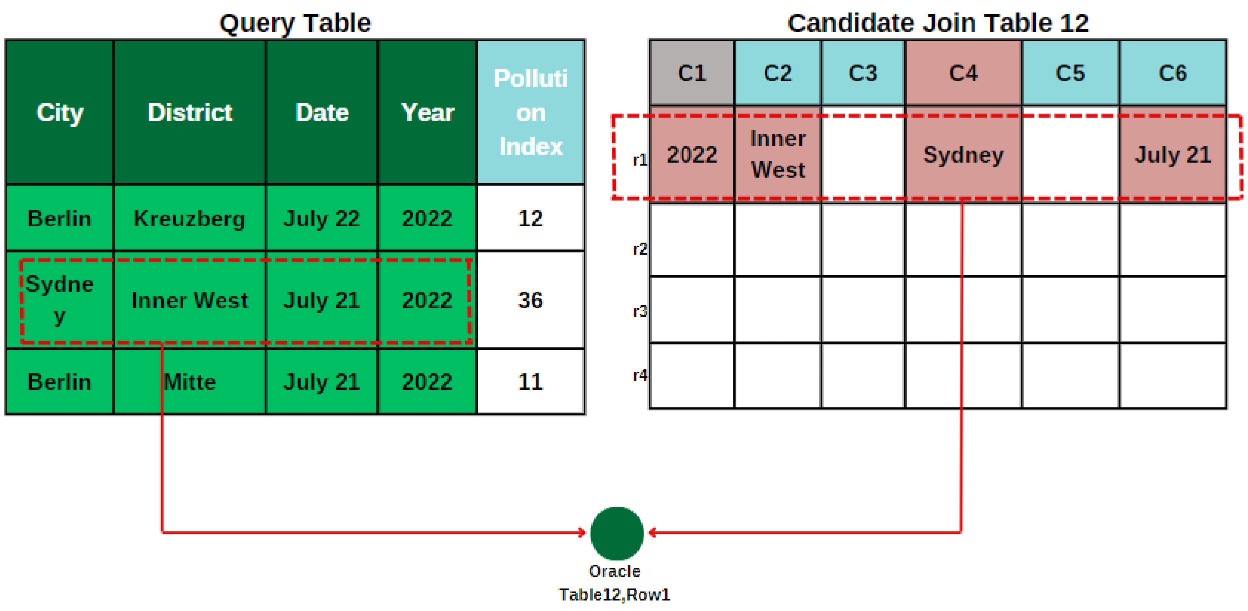
Our multi-attribute join enablement method expands upon the standard inverted index structure. The inverted index is a structure that associates words with the tables, rows, and columns that make up their containing structures. The index that was suggested for the DataXformer system is expanded in this work:

$$v\, i \twoheadrightarrow \rightarrow PL\, i = \{ (T\, i1, C\, i1, R\, i1), (T\, i2, C\, i2, R\, i2),\ldots \}.$$

The term "Posting List" (PL) refers to this triplet list. We also refer to each and every triplet as a PL item.

With small modifications, many modern systems make use of the inverted index. The current systems retrieve the PL items for each value in the key column given a single-attribute join key. the number of returned PL items indicates the joinability score for each table. We must create a multi-attribute inverted index that associates each possible combination of cell values with a specific location in the tables in order to take advantage of the same advantages and optimizations for n-ary joins.

For example, we would like to have a PL item in our below example that maps the key value of to the rows that simultaneously contain both of these values. To locate the multi-attribute joinable tables, a simple algorithm that makes use of the original inverted index can be applied. The PLs of a single query column are first obtained by this algorithm, which then checks to see if the values of the other query columns show up in the same tables and rows. If corresponding oracle which says if this composite key happens in specific candidate table row and If there is match we can say it’s candidate joinable table.



City	District	Date	Year	Polluti on Index
Berlin	Kreuzberg	July 22	2022	12
Sydne y	Inner West	July 21	2022	36
Berlin	Mitte	July 21	2022	11

	C1	C2	C3	C4	C5	C6
r1	2022	Inner West		Sydney		July 21
r2						
r3						
r4						

Fig 6.1.1 SuperKey

The expansion of the inverted index for joins with multiple attributes using different columns as join keys. Large dataset scalability and effective join discovery are made possible by this extension.

6.2 Offline Phase: Building Inverted Index

The system presents the idea of a super key, or an aggregated hash value, during the indexing stage. To quickly find tables that can be joined to a given table, utilize this super key. It works with any combination of keys and doesn't require knowledge of the table's actual keys. A new hash function named Xash is used to generate the super key. It encodes each row value in a highly disseminative manner. This indexing step decreases post-processing overhead and enables the system to swiftly ascertain whether a row is a candidate joinable row or not, which increases the efficiency of multi-attribute join discovery.

6.2 Online Phase: Discovery Phase

In the online phase,the user submits a dataset with a composite key and a parameter k in the online data discovery scenario. The objective is to identify the top-K tables in the input dataset that have the greatest number of joinable rows. Before calculating joinability, Mate filters out irrelevant tables using the super key entry. Initialization, table filtering, row filtering, and joinability calculation are the four stages of the process. For more detailing is in section 6.

7. Super Key Generation

7.1 prerequisites

The need to extend the traditional single-attribute inverted index to support multi-attribute joins. To achieve this, an additional index element called the super key is added to the index structure. The super key is a fixed-size bit array that aggregates hash values for each row value, allowing for the verification of the existence of a composite key without checking all values of the row. The super key masks the hash values of each single row value, ensuring that no value is missed when probing with the super key using the same hash function.

By aggregating hash values for every row value using a fixed-size super key, we can confirm the existence of a composite key without having to check every row value. Mate, the system under discussion, uses the logical bit-wise OR operation to aggregate the hash results of individual values into a fixed-sized bit vector. By using the same hash function to probe with the super key, this guarantees that no value is ignored.

7.2XASH

The document proposes Xash as a hash function with the goal of encoding syntactic features into unique hashes. After that, a bitwise OR operation is used to combine the hash values to produce a super key. False positives (FPs) could be passed and nonexistent values could be concealed by this super key. Xash aims to distribute the 1-bits throughout the super key in a way that minimizes the possibility of different values from various columns setting the same set of bits to 1. In below image can give idea about the how XASH works:

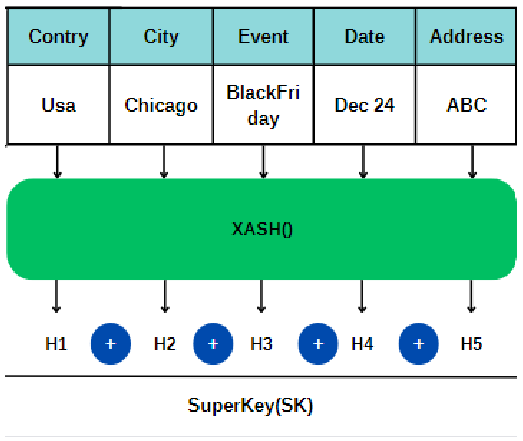


Fig 7.2.1 Super Key Genration

7.2.1 XASH Goal

we focus on the process of extracting features from a row value to apply Xash hashing. We start by discussing the number of bits required to generate the hash and its relationship to the table corpus.

For generating XASH:

1. Value -> Hash

The Xash hash function is designed to encode row values in a way that minimizes collisions and maximizes the uniqueness of hash results. It achieves this by leveraging three syntactic properties of the cell values: the least frequent characters, their location, and the value length.
To illustrate how Xash works, let's consider an example. Aims to generate hash results that have as few overlapping bits as possible for different values. It does this by considering the rarity of characters, their positions, and the length of values.

2. Minimum number of 1 bits

The super key is a key that is used to encode values into a fixed-size bit array. In order to avoid masking false positives (FPs), the super key should contain as few '1s' as possible. This means that there should be an upper bound on the number of '1 bits' for each hash.

7.2.2 Features Encoding

XASH requires three different features to generate hash values.

1. Rare Characters

We use one bit to encode each character so that different values can be encoded using different bit segments. We rely on the character frequency to achieve maximum differentiation between words. Because replacing any character with a less frequent one decreases the probability of collision, the least frequent characters result in fewer collisions.

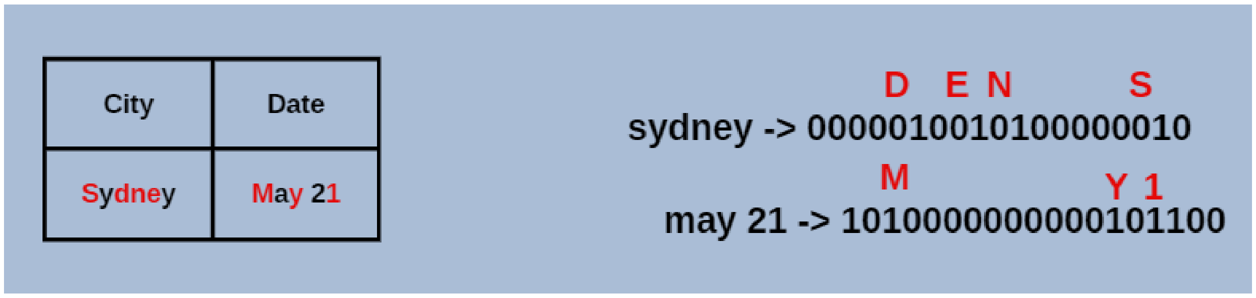


Fig 7.2.2.1 Rare Characters

We divide the hash space into smaller blocks according to the length of the hash array in order to encode the features of a cell value. A specific amount of bits is needed for each feature. The hash array was divided into smaller, fixed-size segments for Xash, one segment for each character. In the example, highlighted character represent the selected least frequent character.

2. Character position

To further distinguished the hash results of various values, we can encode the relative character position inside the original string using multiple bits. In order to accomplish this, we first split the string into equal sections from left to right, then we encode each character by determining which section it appears in. Then, working left to right across the character bits, we only set the matching bit.

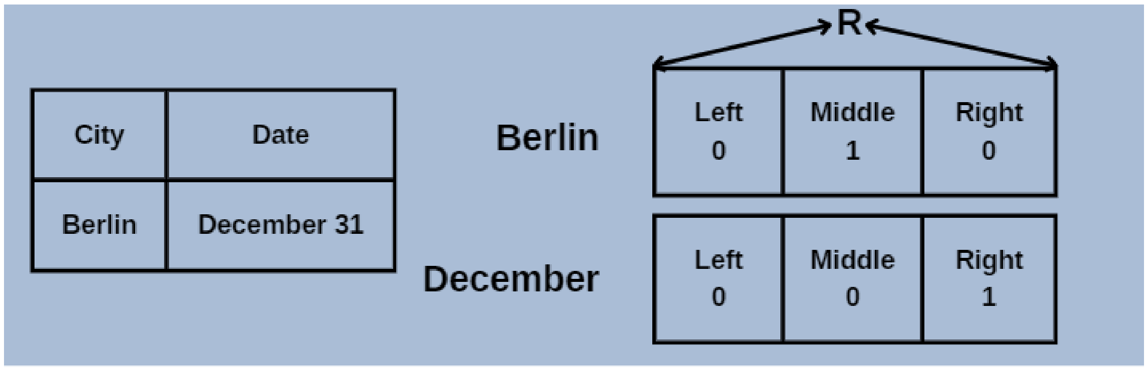


Fig 7.2.2.1 Characters Positions

The bit index can be found using the formula

$\boldsymbol{x} = \mathbf{l}\, \boldsymbol{\lambda} \cdot \boldsymbol{\beta}\, \boldsymbol{l}\, \boldsymbol{v}\, \mathbf{m},$

in which **Z** v denotes the value's length, λ denotes the character's location, and β is the total number of bits that are available. Based on the relative character positions of the values, this encoding aids in differentiating the hash results of various values.

3. Value length

The length of the value is encoded in a fixed-size segment in Xash. It can present challenges to store the binary representation of the value length since doing so results in an arbitrary number of 1 bits being stored in the segment. Additionally, this can hide columns with shorter values' length values. One bit is reserved for each possible length mod |a| in order to address this. Because each length sets a distinct bit, this makes sure that different length values do not obscure one another. By placing the length segment at the leftmost position. Character segments are not checked if there is no value in the candidate row that is the same length as a query value.

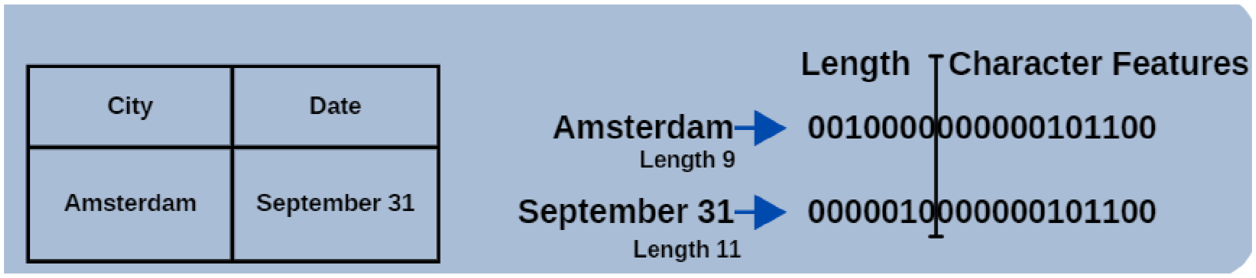


Fig 7.2.2.1 Value Length

In the given example here, character and it's position is same so encoded value length differentiating character feature. For encoding length of set of bits of hash value, left hand side is given length and right hand side is character features.

In Xash, bit rotation is a technique used to distinguish between hash values and lower the probability of random matches. It rotates the character hash segments of a value to the left by the value's length when applied to them. The bits that fall off to the left are shifted to the bit vector's beginning. By guaranteeing that the length of the key value matches the length of the column that overlaps in terms of the least frequent characters, this rotation which only applies to the character-related segments helps prevent false positives.

7.2.3 Representation of Xash and the summarized outcomes

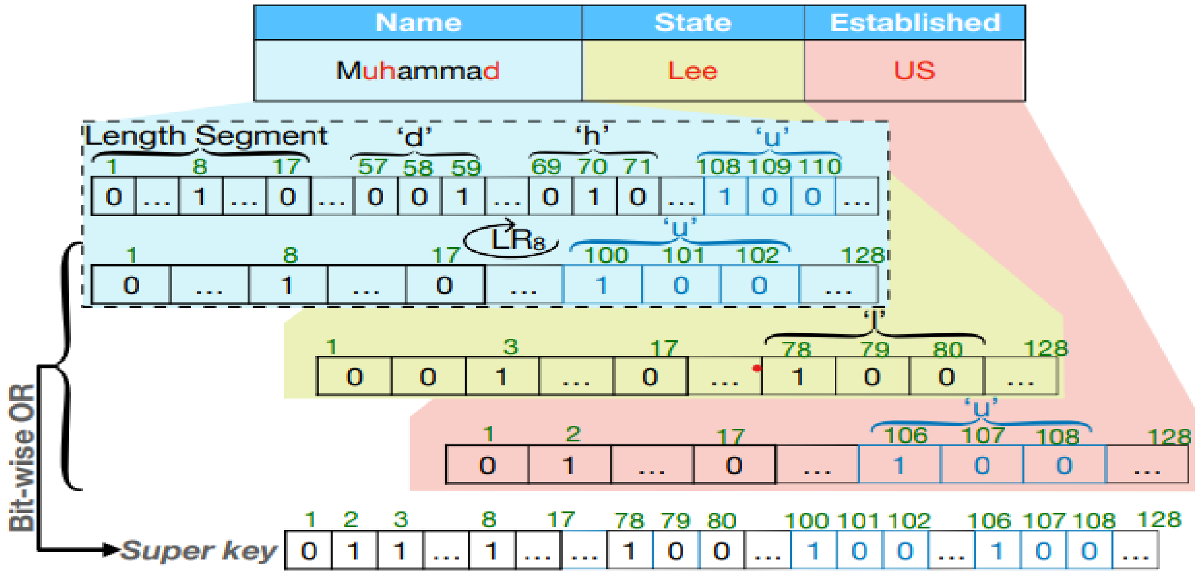
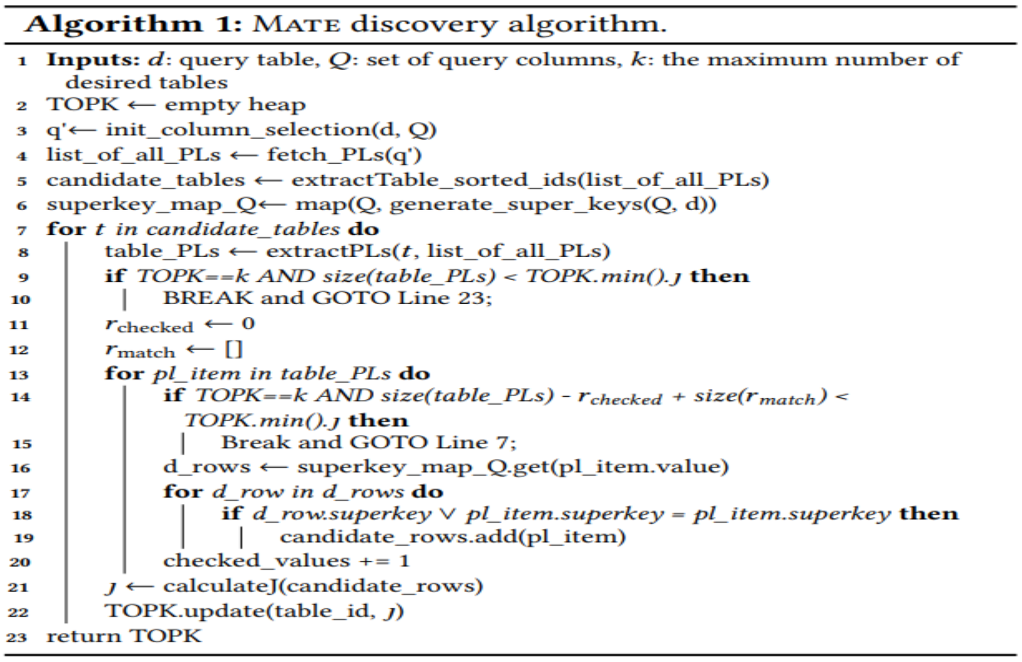


Fig 7.2.3.1 Xash Function Diagram

A diagram of how Xash functions and how the data is combined is shown. A hash function called Xash converts syntactic elements into unique hashes. In order to distribute 1-bits so that there is a lower chance of two different values from different columns turning the same set of bits to 1, it makes use of the least frequent characters, their location, and the value length. An OR-aggregation of these hash values is the super key. This aggregation has the potential to pass false positives (FPs) and mask nonexistent values. Xash distributes the 1-bits in an efficient manner in an effort to reduce the number of FPs.

8. MATE Algorithm



Algorithm 1: MATE discovery algorithm.

```
1  Inputs: d: query table, Q: set of query columns, k: the maximum number of
     desired tables
2  TOPK ← empty heap
3  q'← init_column_selection(d, Q)
4  list_of_all_PLs ← fetch_PLs(q')
5  candidate_tables ← extractTable_sorted_ids(list_of_all_PLs)
6  superkey_map_Q← map(Q, generate_super_keys(Q, d))
7  for t in candidate_tables do
8      table_PLs ← extractPLs(t, list_of_all_PLs)
9      if TOPK==k AND size(table_PLs) < TOPK.min().J then
10         BREAK and GOTO Line 23;
11     $r_{checked}$ ← 0
12     $r_{match}$ ← []
13     for pl_item in table_PLs do
14         if TOPK==k AND size(table_PLs) - $r_{checked}$ + size($r_{match}$) <
             TOPK.min().J then
15             Break and GOTO Line 7;
16         d_rows ← superkey_map_Q.get(pl_item.value)
17         for d_row in d_rows do
18             if d_row.superkey ∨ pl_item.superkey = pl_item.superkey then
19                 candidate_rows.add(pl_item)
20         checked_values += 1
21     J ← calculateJ(candidate_rows)
22     TOPK.update(table_id, J)
23 return TOPK
```

Inputs:
1. Query table
2. Set of query columns
3. Maximum number of desired tables

Step1 - Initializes an empty heap called TOPK.

Step2 - It selects an initial query column based on a cardinality-based heuristic. It fetches the PL items for the initial column and extracts the candidate tables.

Step3- It generates super keys for the join columns of the query table.

Step4- The algorithm iterates over each candidate table in the list of candidate_tables.For each candidate table, it extracts the PL items (probabilistic labels) associated with the table using the extractPLs() function.

Step5- If the number of PL items in the candidate table is less than the minimum number of PL items in the top-k list (TOPK.min()), the algorithm breaks the loop and goes to step10.

Step6- It initializes two variables: checked and match, which are used to keep track of the number of checked PL items and the number of PL items that match the join condition, respectively.

Step7- It iterates over each PL item in the table_PLs.If the number of PL items in the candidate table minus the checked PL items plus the size of the match is less than the minimum number of PL items in the top-k list (TOPK.min()), the algorithm breaks the loop and goes to step4.

Step8- It retrieves the corresponding rows from the input dataset (d_rows) using the superkey_map_Q dictionary.

Step9- For each d_row, it compares the superkey of the d_row with the superkey of the PL item.If the superkeys match, it adds the PL item to the candidate_rows set.Finally, it calculates the joinability score (calculateJ) for the candidate_rows and updates the TOPK list with the table_id and the joinability score.

Step10 -The algorithm returns the updated TOPK list.

9. JOINABLE TABLES DISCOVERY

the method of effectively using Xash and the index structure to find n-ary joins. The four primary steps in the discovery phase are joinability calculation, table filtering, row filtering, and initialization.

9.1 Initialization

The method of choosing the Mate system's first query column for effective retrieval. The choice of the first column is very important because it can affect the system's runtime. The system chooses the column with the fewest unique values as the optimal initial query column, taking into account the cardinality of the query table. In the index, this heuristic seeks to match fewer possible matches.

Finding Super Keys and PL Items:
Mate retrieves PL items with their generated super keys for the matching rows when the first column is selected. Next, a grouping of the retrieved PL items is done using the table identifiers.
Table pruning techniques and the evaluation of more promising tables first are made possible by the sorting of the tables according to the number of PL items in each table.

Super Key Generation for Join Columns:
Mate creates the appropriate super keys for the query table's join columns.

9.2 Table Filtering

Reducing the number of candidate tables during the join process is the goal of the suggested table filtering technique. Tables that are unlikely to be chosen as the top-k joinable candidates are pruned by Mate. It applies two methods:

Strategy 1: The system removes the candidate table from further evaluations if PL items for the current candidate table are Lt and the joinability score of the worst table in the top-k list is $\jmath k$. This rule is predicated on the inability of the remaining tables, which are sorted according to the number of PL items in each, to rank among the top k tables.

Strategy 2: Mate drops the current table if Lt - $\boldsymbol{U}$checked + $\boldsymbol{U}$match $\leq \jmath k$, where $\boldsymbol{U}$checked denotes the number of evaluated rows already evaluated and $\boldsymbol{U}$match denotes the number of joinable rows among the evaluated rows. The reason this rule ignores the current candidate table is that it cannot achieve a higher joinability score than the worst top-k table, even if all the remaining rows can be joined to the input table. During this check, joinable rows are filtered using the super key.

9.3 Row Filtering

Mate examines candidate tables to see if they have what it takes to rank among the top k tables. To quickly determine whether key values in candidate rows are present, it makes use of super keys that are generated for both the candidate and query tables. The system can exclude a query table row from joinability calculations if its super key is not obscured by the super key of the candidate joinable table. Mate also uses a generated dictionary that maps initial column values to corresponding super keys of the composite key value combination to determine which query rows should be compared with each candidate row.
The super keys of the current candidate row and its corresponding input rows are compared bit-wise OR masking for each corresponding input row. If the outcome of the bit-wise OR operation on their super keys equals the super key of the PL item, then an input row is regarded as a potential join candidate to its corresponding row of the PL item.

The hash values for the following names are provided: Muhammad, Lee, US, Ali, Germany, Dancer, Boxer, and Birder in running example.

Table 1's rows with the value "Muhammad" would have an aggregated super key of 11111110, 11011101, and 11101001 for rows 2, 5, and 6, respectively.

Comparably,
1001000 ∨ 01100000 ∨ 00010100 = 1111100
would be the super keys over the columns in one table and for the first row in another table.

The super keys of rows 5 and 6 (11011101 and 11101001) are not a subset of the resulting super key, 1111100. Thus, rows 5 and 6 are not included in any additional computations. Yet, the super key of the second row in Table 1 (11111110) encompasses the super key 1111100. Consequently, the list of candidate rows is expanded to include the second row.
False negative results are never possible with this operation, but false positives (FPs) are.

9.4 Analysis

Compared to Bloom Filters (BF), Xash is more efficient because of two key features.

Non-Uniform Distribution of Hash Values: Xash makes use of the fact that different values originate from different columns, in contrast to BF, which assumes a uniform distribution of hash functions. It avoids random overlaps by mapping hash results to distinct segments according to their domain. The possibility of collisions between disparate values is decreased by this method.

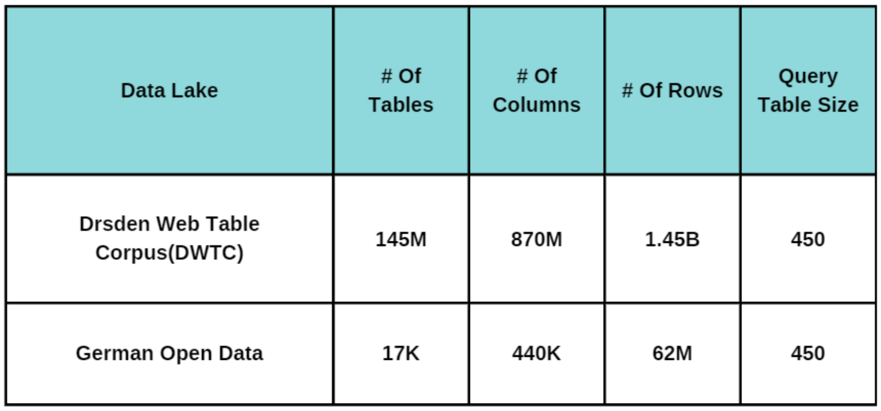
Length Encoding: To further improve its effectiveness, Xash also encodes values' lengths. Xash lowers the likelihood of collisions when encoding random characters by incorporating the length information into the hash encoding. This feature makes value matching more precise and effective.
Xash is more efficient than BF overall because it uses length encoding and non-uniform distribution.

10. Experiments

We consider certain questions when assessing Mate. Initially, we evaluate the runtime advantage of Mate over the most advanced inverted index for n-ary joins. Second, we examine the impact of various parameters on the runtime, including the hash size, |Q|, the cardinality of tables, and the choice of k. Finally, we compare Xash's filtering ability to that of other hash functions. These assessments aid in our comprehension of Mate's functionality and efficacy in various contexts.

10.1 Experimental Setup

The application of various query tables to two table corpora: the German Open Data repository and the Dresden Web Table Corpus (DWTC). The selection of these corpora was based on the presence of tables with different dimensions and sizes.



Data Lake	# Of Tables	# Of Columns	# Of Rows	Query Table Size
Drsden Web Table Corpus(DWTC)	145M	870M	1.45B	450
German Open Data	17K	440K	62M	450

Fig 10.1.1 Tables for experiment

Query Table Groups
A selection of 900 query tables, broken down into six groups of 150 tables each, is mentioned in the document. To make sure there are no duplicate tables in each group, these query tables are chosen at random from the webtable and open data corpora.

Distributions of Cardinality:
The query tables show cardinalities less than 100, 1000, and 10,000 for the open data groups, respectively. Regarding the DWTC corpus, there are tables in the groups that have cardinalities less than 10, 100, and 1000.

Academic Corpus
Apart from the DWTC and German Open Data corpora, the document also references the utilization of a 335-table School corpus. These tables typically contain 30,000 rows and more than 27,000 columns. With the help of this corpus, systems with larger tables can be evaluated.

Datasets for Machine Learning
The selection of machine learning datasets from Kaggle that need to be enhanced is also mentioned in the document. These query tables are used with the webtable corpus and have more generic content. Based on their first query column, these query tables yield approximately 7 joinable candidates on average.

Infrastructure and Implementation

Mate was implemented using Python 3.7 and operated on a server equipped with 2 TB of SSD storage, 500 GB of RAM, and 128 CPU cores. For storing the index, Vertica v10.1.1-0 was used. The datasets and code utilized in the experiments are openly accessible on github.

10.2 Mate VS. Baseline Systems

Measures MATE's performance against a number of baseline systems, such as JOSIE, SCR, and MCR implementations. Based on the amount of time needed to find multi-column joins in-memory using the most advanced single-attribute joinable table discovery systems available, a comparison is made. To assess MATE's and the baseline systems' performance, the authors ran a number of experiments. The ability of the systems to handle various table types and join keys was tested using a variety of datasets of differing sizes and complexity.

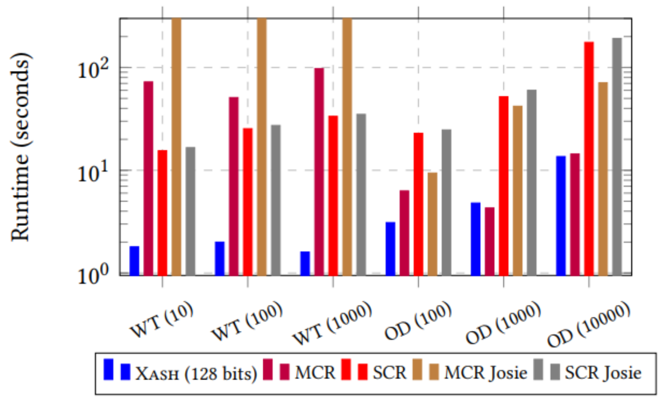


Fig 10.2 Output

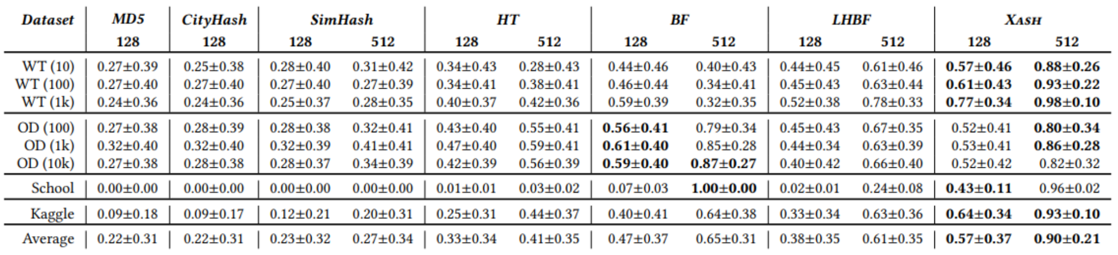
The experiments' findings demonstrated that MATE almost always performed better than the baseline systems. As an illustration, MATE outperformed MCR by up to 61 times, SCR by 13 times, MCR Josie by 9 times, and SCR Josie by 22 times. The efficiency with which MATE handles n-ary join keys and its capacity to effectively prune non-joinable rows by utilizing the Xash hash function are both demonstrated by these results.

FPs are rows that are incorrectly regarded as joinable, which causes more comparisons and slower handling. For instance, the authors discovered that in the case of SCR, even though the query table sizes for OD (1k) and WT (1k) are comparable, the OD (1k) queries result in a 35% longer runtime than WT (1k). The high number of FPs for OD (1k) tables—that is, three million more FP rows than WT (1k)—is the cause of this runtime increase. This illustrates how FPs affect table discovery system performance and highlights the significance of reducing them.

10.3 Precision and FP Rate

Precision and FP rate of the hash function used in the table discovery runtime are directly correlated. The ratio of true positives (TP) over both TPs and FPs is used to calculate precision. Since hash functions scatter values over a wider range, reducing the likelihood of super key collisions, precision typically rises with hash sizes. In both 128-bit and 512-bit hash spaces, Mate + Xash outperforms the other methods in terms of precision, reaching up to 25% higher than BF. That being said, there is only a 4% difference in five cases where BF performs better than Xash.

Table 3: Precision experiment.



Dataset	*MD5*	*CityHash*	*SimHash*		*HT*		*BF*		*LHBF*		*Xash*	
	128	**128**	**128**	**512**	**128**	**512**	**128**	**512**	**128**	**512**	**128**	**512**
WT (10)	0.27±0.39	0.25±0.38	0.28±0.40	0.31±0.42	0.34±0.43	0.28±0.43	0.44±0.46	0.40±0.43	0.44±0.45	0.61±0.46	**0.57±0.46**	**0.88±0.26**
WT (100)	0.27±0.40	0.27±0.40	0.27±0.40	0.27±0.39	0.34±0.41	0.38±0.41	0.46±0.44	0.34±0.41	0.45±0.43	0.63±0.44	**0.61±0.43**	**0.93±0.22**
WT (1k)	0.24±0.36	0.24±0.36	0.25±0.37	0.28±0.35	0.40±0.37	0.42±0.36	0.59±0.39	0.32±0.35	0.52±0.38	0.78±0.33	**0.77±0.34**	**0.98±0.10**
OD (100)	0.27±0.38	0.28±0.39	0.28±0.38	0.32±0.41	0.43±0.40	0.55±0.41	**0.56±0.41**	0.79±0.34	0.45±0.43	0.67±0.35	0.52±0.41	**0.80±0.34**
OD (1k)	0.32±0.40	0.32±0.40	0.32±0.39	0.41±0.41	0.47±0.40	0.59±0.41	**0.61±0.40**	0.85±0.28	0.44±0.34	0.63±0.39	0.53±0.41	**0.86±0.28**
OD (10k)	0.27±0.38	0.28±0.38	0.28±0.37	0.34±0.39	0.42±0.39	0.56±0.39	**0.59±0.40**	**0.87±0.27**	0.40±0.42	0.66±0.40	0.52±0.42	0.82±0.32
School	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.01±0.01	0.03±0.02	0.07±0.03	**1.00±0.00**	0.02±0.01	0.24±0.08	**0.43±0.11**	0.96±0.02
Kaggle	0.09±0.18	0.09±0.17	0.12±0.21	0.20±0.31	0.25±0.31	0.44±0.37	0.40±0.41	0.64±0.38	0.33±0.34	0.63±0.36	**0.64±0.34**	**0.93±0.10**
Average	0.22±0.31	0.22±0.31	0.23±0.32	0.27±0.34	0.33±0.34	0.41±0.35	0.47±0.37	0.65±0.31	0.38±0.35	0.61±0.35	**0.57±0.37**	**0.90±0.21**

Overview of the experiment:

(i)An approach's runtime and precision are correlated. This implies that an approach's execution time may increase with its level of precision.

(ii) Even in situations where precision rates are comparable, join discovery is accelerated by the bit segmentation in Xash. This shows that even when the precision rates are comparable to other approaches, Xash's technique of splitting the bits in a hash function enables faster identification of joinable tables.

11. Conclusion

The authors of this paper tackled the issue of multi-attribute or n-ary join search. They presented Mate, a hash-based filtering method that increases join discovery's scalability by eliminating unnecessary table rows. Additionally, they put forth the idea of Xash, a hash function that reduces false positives by using the syntactic characteristics of cell values. The authors emphasized the potential of Xash for finding similarity joins and showed how Mate is more effective and efficient than current systems. Additionally, they noted that Xash might result in false positives.