ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20230402000493
Original scientific paper

A Method for Automatically Generating Join Queries Based on Relations-Attributes Distance Matrix over Data Lakes

Caicai ZHANG, Chenglang LU, Zhuolin MEI, Bin WU, Jing YU*

Abstract: Techniques for identifying joinable or unionable tables in data lakes can yield valuable information for data scientists. However, more than half of their working time is spent familiarizing themselves with the metadata and correlations of datasets. Simplifying the use of information in data lakes is crucial for enhancing their utilization. The existing solution of integrating correlated relations into a single large data table via full disjunction requires integration updating when either data or metadata changes, complicating data maintenance. This paper proposes a method for automatically generating join queries based on the distance matrix of relations and attributes in data lakes. The distance matrix only requires updating when metadata changes, simplifying data maintenance. Experimental results demonstrate that once the distance matrix is generated, the time required to generate the join queries is negligible. Compared to the existing solution, the time cost for executing join queries over correlated tables is nearly identical to that of selection queries over integrated tables. The results of these two queries are also the same, showcasing the effectiveness and efficiency of our method.

Keywords: data integration; data lakes; distance matrix; join queries

1 INTRODUCTION

The explosion of big data has led to millions of datasets available in data lakes, covering fields such as social science, life science, and more [1-3]. Access to this data is vital for facilitating reproducible research results, aiding scientists in advancing their work, and providing data journalists with easier access to information and its provenance [4-6]. However, the vast scale of datasets in data lakes presents new and intriguing challenges for data management research [7, 8]. While metadata varies across datasets in data lakes, attributes with similar semantics can be considered the same attribute based on value overlap, ontology overlap and natural language similarity [9]. Join queries enable the utilization of correlated information over data sets with common attributes, which is critical for data scientists. Yet, the enormous amount of metadata and correlations between datasets creates barriers for data scientists seeking to harness data lakes' potential for their research.

A survey found that 41% of data scientists use data lakes to collect interesting datasets for their research [10]. They spend 51% of their working time collecting and preparing datasets, primarily familiarizing themselves with the metadata and correlations of datasets. Fig. 1 shows an example of correlated tables in data lakes. Therefore, a solution for querying and collecting interesting datasets without requiring knowledge of each specific relation's structure and their correlations has broad potential applications.

One solution is to integrate all correlated relations into a single large data table, generated by full disjunction [11]. However, the massive size and inevitable null values associated with the large table may lead to decreased data quality and increased data storage and query costs. In this paper, we propose a new solution based on automatically generated join queries over data lakes. The main contributions of this paper are as follows.

(1) We introduce the concept of automatically generating join queries over data lakes based on pairs of attributes. Query users do not need knowledge of the structures of relations or correlations within data lakes. Users only need to focus on the set of all attributes present in data lakes and the pair of interesting attributes, significantly increasing the convenience of utilizing data lakes to search for valuable and interesting information.

(2) We propose a relations-attributes distance matrix generation algorithm and a join path generation algorithm based on the relations-attributes distance matrix. Once the relations-attributes distance matrix is generated, the join path for a given pair of attributes can be produced efficiently. Moreover, when new correlated relations become available, updating the distance matrix incurs a much lower time cost compared to generating large integrated tables.

(3) We conducted experiments using various datasets scales, to demonstrate the effectiveness and efficiency of our proposed algorithms.

T1

TID	Stadium	Team
1	NRG Stadium	Houston Texans
2	AT&T Stadium	Dallas Cowboys
3	Sofi Stadium	Angeles Chargers

T2

TID	Stadium	Location	Opened
4	Soldier Field	Chicago	1924
5	AT&T Stadium	Texas	2009
6	Sofi Stadium	California	2020

T3

TID	Team	Coach
7	Houston Texans	Lovie Smith
8	Dallas Cowboys	Mike McCarthy
9	Detroit Lions	Dan Campbell

Figure 1 An example of tables in data lakes

The remainder of this article is organized as follows. Section 2 discusses the related work. Section 3 describes our proposed join query automatic generation method. Section 4 reports on the performance evaluation of our techniques over several data benchmarks. Finally, Section 5 concludes this paper.

2 RALATED WORK

To the best of our knowledge, no previous works specifically focus on generating join paths of relations in data lakes. Therefore, we will discuss related work primarily related to data integration in data lakes, data searching in data lakes and automatic query generation.

Data integration in data lakes. Data integration in data lakes typically involves several steps, such as assigning column integration IDs and applying full disjunction. Assigning column integration IDs aims to identify correspondences among database attributes. Koutras et al. [12] investigated how to find pairs of matching columns based on traditional schema matching methods. Some works [13, 14] use clustering-based approaches with schema information to match a set of schemas simultaneously, while others [11] rely on data values instead of metadata to overcome unreliable metadata in data lakes. Many researchers have developed table search and management solutions for data lakes [15]. Miller et al. [16] analyzed the necessary data management techniques to make open data accessible to data scientists and defined the problem of finding unionable tables in data lakes. Subsequently, other works designed methods to find unionable tables based on column relationships [9, 17, 18]. Dong et al. [19] proposed a framework for the discovery of joinable tables based on similarity predicates on high-dimensional vectors that represent columns. This framework can capture the semantic similarity of attributes and handle misspellings and different formats. Lastly, some works, such as [20], search for a set of tables with semantic similarity by representing tables as vector representations. The techniques for identifying correlated attributes and tables form the basis of our approach. However, our approach applies join queries over correlated tables rather than generating large table via data disjunction.

Data searching from data lakes. After assigning column integration IDs and identifying unionable or joinable tables, there are two primary methods to search data from data lakes. One is to select data from integrated tables generated by full disjunction [21]. Many works have been done to efficiently compute full disjunction [22, 23]. The other approach is to use deep learning models to learn vector representations of tables, attributes and data, and treat queries like query answering systems [24]. For example, TURL [25] uses a structure-aware Transformer encoder to model the row-column structure information in relational tables. The Transformer encoder has achieved significant success in extracting information from unstructured text data [26]. However, little effort has been made to study such paradigms on structured relational tables. Tab. 2 Vec [27] serializes a table into a sequence of words and entities and learns embedding vectors for words and entities using Word2Vec [28].

Automatic query generation. In order to make database accessible interfaces more intelligent and capable of understanding natural language expressions, many researchers have explored the automatic generation of SQL language from natural language [29-31]. Sangeetha [32] focused on mapping spoken natural language into words forming the foundation of SQL by using a dictionary to record semantic sets for columns and tables. Speech recognition techniques are utilized to convert spoken language input into text, and semantic matching techniques are employed to convert natural language queries into SQL words [33]. Yu [34] adopted the Transformer model [26] to semantically match query request statements and query conditions to improve the correctness of the automatically generated WHERE clause in SQL statements. Badhya [35] proposed Elastic search to convert natural language inputs into optimized SQL queries, which can be understood by different databases, such as MySQL, PostgreSQL and MongoDB.

3 JOIN QUERY AUTOMATIC GENERATION METHOD

3.1 Definitions

Definition 1. Inclusion matrix of relations and attributes M^1. Given a set of relations D, where A is the union of the set of attributes in all relations, M^1 is a boolean matrix of size $|A|\times|D|$. $M^1[i,j]=1$ if the attribute A_i exists in relation T_j, otherwise, $M^1[i,j]=0$.

Example 1. Here is an example of relationships between four relations $T_1(A_1,A_2)$, $T_2(A_2,A_3)$, $T_3(A_3,A_4)$, and $T_4(A_5,A_6)$, shown in Fig. 2. Tab. 1 presents the corresponding inclusion matrix.

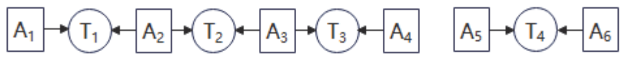

Figure 2 An example of four relations

Table 1 An example of inclusion matrix

	T_1	T_2	T_3	T_4
A_1	1	0	0	0
A_2	1	1	0	0
A_3	0	1	1	0
A_4	0	0	1	0
A_5	0	0	0	1
A_6	0	0	0	1

Definition 2. Joinable relations sequence. A sequence of relations $< T_{x_1}, T_{x_2}, ..., T_{x_m} >$ is said to be a joinable relations sequence if every pair of neighboring tables in the sequence has common attributes.

Definition 3. The distance matrix of relations and attributes M. Given a set of relations D, and the union set of attributes of all relations A, the distance matrix of relations and attributes M is a matrix of size $|A|\times|D|$. $M[i,j]=1$ if attribute A_i exists in relation T_j; $M[i,k]=m$ if attribute A_i does not exist in relation T_k, and there exists a joinable relations sequence $\langle T_j,...,T_k\rangle$ with at least length m; $M[i,k]=0$ if there is no joinable realtions sequence between T_k and T_j.

Example 2. Tab. 2 displays the distance matrix of relations and attributes associated with Fig. 2.

Table 2 An example of distance matrix of relations and attributes

	T_1	T_2	T_3	T_4
A_1	1	2	3	0
A_2	1	1	2	0
A_3	2	1	1	0
A_4	3	2	1	0
A_5	0	0	0	1
A_6	0	0	0	1

Definition 4. Join path automatic generation problem. Given a pair of attributes $\langle A_t, A_c \rangle, A_t \in A, A_c \in A$, the join path automatic generation problem involves obtaining a sequence of relations $\langle T_{x_1}, T_{x_2}, ..., T_{x_m} \rangle$ and a set of joinable attributes set sequence $\langle [A_{y_2}^1, ..., A_{y_2}^{n_2}], ..., [A_{y_m}^1, ..., A_{y_m}^{n_m}] \rangle$ which satisfy the following conditions:

(1) $A_t \in A(T_{x_1}), A_c \in A(T_{x_m})$;

(2) $[A_{y_i}^1, ..., A_{y_i}^{n_i}] = A(T_{x_{i-1}}) \cap A(T_{x_i}), i \in [2, m]$.

Example 3. Given the distance matrix of relations and attributes in Tab. 2, and a pair of attributes $\langle A_1, A_4 \rangle$, the join path is $\langle T_1, T_2, T_3 \rangle$, and the joinable attributes sets sequence is $\langle [A_2], [A_3] \rangle$.

3.2 Architecture

Given the distance matrix M and a pair of attributes $\langle A_t, A_c \rangle$, the join path can be automatically generated by Algorithm 2, as presented in Section 3.3. Users only need to have knowledge of the attribute sets in the relations. Queries submitted by users based on the attribute set can be transformed into join queries based on the automatically generated join path. The system architecture is depicted in Fig. 3.

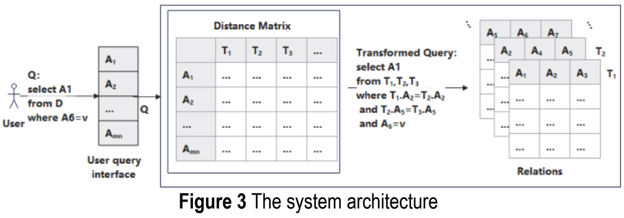

Figure 3 The system architecture

Given a query $\sigma_{f_1(A_t) \wedge f_2(A_c)}(D)$ over relations D, if the generated join path for $\langle A_t, A_c \rangle$ is $< T_{x_1}$, T_{x_2}, ..., $T_{x_m} >$, and the joinable attributes sets sequence is $< A_{y_1}$, A_{y_2}, ..., $A_{y_{m-1}} >$, then the transformed query can be expressed as:

$$\sigma_{f_1(A_t) \wedge f_2(A_c)}(T_{x_1} \bowtie_{T_{x_1}.A_{y_1} = T_{x_2}.A_{y_1}} T_{x_2} \bowtie ... \bowtie_{T_{x_{m-1}}.A_{y_{m-1}} = T_{x_m}.A_{y_{m-1}}} T_{x_m})$$

Example 4. Based on Example 3, if a user submits a query: select * from D where $A_1 = v_1$ and $A_4 = v_2$, then the transformed query is selected * from T_1, T_2, T_3 where $T_1 . A_2 = T_2 . A_2$ and $T_2 . A_3 = T_3 . A_3$ and $A_1 = v_1$ and $A_4 = v_2$.

Given a query $\Pi_{A_t}(\sigma_{f(A_c)}(D))$ over relations D, if the generated join path for $\langle A_t, A_c \rangle$ is $\langle T_{x_1}, T_{x_2}, ..., T_{x_m} \rangle$, and the join attributes sets sequence is $\langle A_{y_1}, A_{y_2}, ..., A_{y_{m-1}} \rangle$, then the transformed query is:

$$\Pi_{A_t}(\sigma_{fa(A_c)}(T_{x_1} \bowtie_{T_{x_1}.A_{y_1} = T_{x_2}.A_{y_1}} T_{x_2} \bowtie ... \bowtie_{T_{x_{m-1}}.A_{y_{m-1}} = T_{x_m}.A_{y_{m-1}}} T_{x_m}))$$

Example 5. Based on Example 3, if a user submits a query select A_1 from D where $A_4 = v$, then the transformed query is select A_1 from T_1, T_2, T_3 where $T_1 . A_2 = T_2 . A_2$ and $T_2 . A_3 = T_3 . A_3$ and $A_4 = v$.

3.3 Algorithms

To implement automatic query transformation, two key problems must be addressed: (1) generating the distance matrix of relations and attributes, and (2) generating the join path and joinable attribute set sequence based on the distance matrix and a pair of attributes.

3.3.1 Algorithm for Generating Distance Matrix

Definition 5. Maximum *dist* distance matrix of relations and attributes M^{dist}. Suppose an attribute A_i exists in relation T_j and not in relation T_k, and there exists a joinable relations sequence of at least length m, if $m \le dist$ then $M^{dist}[i, k] = m$; otherwise, $M^{dist}[i, k] = 0$.

Property 1. Given a set of relations D, attributes A and the maximum *dist* distance matrix M^{dist}, if there exists $M^{dist}[A_x, T_i] = d$, where $d > 0$, then $M^{dist+1}[A_x, T_i] = d$.

Based on Property 1, if $M^1[A_x, T_i] = 1$, then $M^2[A_x, T_i] = M^3[A_x, T_i] = 1$.

Theorem 1. Given a set of relations D, attributes A and maximum *dist* distance matrix M^{dist}, if $M^{dist}[A_x, T_i] = dist$ and $M^{dist}[A_x, T_j] = 1$, then $M^{dist+1}[A_y, T_i] = dist + 1$ for $A_y \in A(T_j)$ and $M^{dist}[A_y, T_i] = 0$.

Proof

$M^{dist}[A_x, T_j] = 1$ indicates that attribute A_x is present in relation T_j.

$M^{dist}\left[A_x, T_i\right] = dist$ indicates that there exists a sequence of joinable relations $\left\langle T_i, T_{x_1}, \ldots, T_{x_{dist-1}} \right\rangle$, $A_x \in A\left(T_{x_{dist-1}}\right)$.

If $A_y \in A\left(T_j\right)$ and $M^{dist}\left[A_y, T_i\right] = 0$, this indicates that there is no joinable path between T_i and T_j within a distance of *dist*, meaning $T_{x_{dist-1}} \neq T_j$, where $T_{x_{dist-1}}$ is the relation in the joinable path that is one step away from T_j.

We can conclude that there exists a sequence of joinable relations $< T_i$, T_{x_1}, ..., $T_{x_{dist-1}}$, $T_j >$, and A_x is the common attribute of $T_{x_{dist-1}}$ and T_j.

Therefore, for $\forall A_y \in A\left(T_j\right)$ and $M^{dist}\left[A_y, T_i\right] = 0$, we have $M^{dist+1}\left[A_y, T_i\right] = dist + 1$.

Algorithm 1 outlines the steps to generate the distance matrix. The inclusion matrix M^1 can be obtained from the given relations. Using this matrix, we can iteratively generate M^2, M^3 and so on, until the distance matrix of adjacent distances no longer updates with new data. At this point, the final distance matrix of relations and attributes M is obtained. The time complexity of the algorithm is $O\left(dist * |D|^2 * |A|\right)$.

Algorithm 1: Generating distance matrix

Input: the set of relations D, the set of attributes A, the inclusion matrix M^1
Output: the distance matrix M

flag = True
dist = 1
$M = M^1$
While flag do
  flag = False
  for each i in range($|D|$)
    for each j in range($|D|$)
      If $M\left[:, T_i\right] == dist * \dot{M}\left[:, T_i\right] == 1 > 0$
        $M\left[x, T_i\right] = dist + 1$ for $\left(M\left[x, T_j\right] = 0 \text{ and } M\left[x, T_j\right] = 1\right)$
        flag = True
  $dist = dist + 1$
Return M

3.3.2 Automatic Generation Algorithm for Join Paths of Relations

Property 2. Given the distance matrix M and a pair of attributes $\left\langle A_t, A_c \right\rangle$, if $M\left[A_t, T_i\right] = 1$ and $M\left[A_c, T_i\right] = dist > 0$, then there exists a sequence of joinable relations $\left\langle T_i, T_{x_2}, T_{x_3}, \ldots, T_{x_{dist}} \right\rangle$, such that $A_t \in A\left(T_i\right)$, $A_c \in A\left(T_{x_{dist}}\right)$, and for all $j \in \left[2, dist\right]$, we have $M\left[A_t, T_{x_j}\right] = j$ and $M\left[A_c, T_{x_j}\right] = dist - j + 1$.

Property 2 implies that a sequence of relations can be joined together to connect attribute A_t in relation T_i to attribute A_c in the last relation in the sequence, with a total length of *dist*. The distance matrix helps identify the joinable relations in the sequence and their distance from the starting and ending attributes in the sequence.

Property 3. If $M\left[A_x, T_i\right] = 1$ and $M\left[A_x, T_j\right] = 1$, then A_x is a common attribute between the relational tables T_i and T_j.

Algorithm 2 outlines the steps to generate the sequence of join relations and the sequence of join attributes sets for a given pair of attributes $\left\langle A_t, A_c \right\rangle$ and the distance matrix M. Since A_t and A_c may appear in more than one relation, Step 5 and 6 in Algorithm 2 first find the shortest distance between attribute A_t and attribute A_c. Then, according to Property 2, we can find each joinable relation T_i that satisfies $M\left[A_t, T_{x_j}\right] = j$ and $M\left[A_c, T_{x_j}\right] = dist - j + 1$ for j ranging from 2 to *dist*. Based on Property 3, we can obtain the sequence of join attributes sets. The time complexity of Algorithm 2 is $O\left(dist * |A|\right)$, where *dist* is the shortest distance between relations for the given pair of attributes.

Algorithm 2: Join paths automatic generation

Input: the pair of attributes $\left\langle A_t, A_c \right\rangle$, the distance matrix M
Output: the sequence of join relations T, the sequence of join attributes sets SA

$T = [\ ]$
$SA = [\ ]$
$T_i \Leftarrow \min\{M\left[A_c, T_i\right] \mid M\left[A_t, T_i\right] == 1\}$
If $T_i == \varnothing$ then Return$\{T, SA\}$
$dist \leftarrow M\left[A_c, T_i\right]$
$T \Leftarrow$ add T_i
if $dist == 1$ then Return (T, SA)
for j in range(2, $dist$ + 1)
  $T_{x_j} \Leftarrow M\left[A_t, T_{x_j}\right] == j$ and $M\left[A_c, T_{x_j}\right] == dist - j + 1$
  $T \Leftarrow$ add T_{x_j}
  $A \Leftarrow$ add $\{A_x \mid M\left[A_x, T_i\right] == 1$ and $M\left[A_x, T_{x_j}\right] == 1\}$
  $T_i = T_{x_j}$
Return T, SA

4 EXPERIMENTS

All algorithms were implemented using Python 3.7, and experiments were conducted on a CentOS server equipped with an Intel(R) Core(TM) i7-10510U CPU @2.30 GHz processor. The primary objectives of these experiments were to determine: (1) the correctness of the automatically generated join paths, and (2) the efficiency of the algorithms in generating the distance matrix and join paths.

4.1 Data Sets

To the best of our knowledge, no existing data benchmarks could be employed to evaluate the automatic generation algorithms for both distance matrix and join paths. Therefore, we have created several data sets of varying scales by adjusting four parameters: *dist*, $|D|$, $|A|$, $n(join\_sets)$. *dist* refers to the largest number of tables in a sequence of join-reachable relations. $|D|$ and $|A|$ denote the number of tables and attributes, respectively. $n(join\_sets)$ represents the number of non-overlapping sets of tables (join sets), where all tables within the same set can be joined together. For each data set, $|D|$ tables were generated, each containing several different attributes. The tables were then divided into $n(join\_sets)$ non-overlapping data sets, ensuring that each set contained no more than *dist* tables. Subsequently, for each join set, common attributes among the tables in that particular set were added. The details of data sets are as follows.

Benchmark D_1. D_1 consists of 9 data sets with *dist* ranging from 6 to 14, while fixing $|D|$ is fixed at 500 and $n(join\_sets)$ at 100. The number of attributes, $|A|$ varies slightly for different *dist* values.

Benchmark D_2. D_2 includes 9 data sets with $|D|$ ranging from 200 to 1000, while *dist* is fixed at 10 and $n(join\_sets)$ at 100. The number of attributes, $|A|$ increases linearly with $|D|$.

Benchmark D_3. D_3 encompasses 9 data sets with $|A|$ increasing linearly, while $|D|$ is fixed at 500, *dist* at 10 and $n(join\_sets)$ at 100.

Benchmark D_4. D_4 comprises 9 data sets with $n(join\_sets)$ ranging from 500 to 4500, while $|D|$ is fixed at 500 and *dist* at 10. The number of attributes, $|A|$ remains relatively constant across different data sets.

4.2 Experimental Results

4.2.1 Performance of Creating Distance Matrix and Generating Join Path Algorithms

Tab. 3 presents the time cost of creating distance matrix and generating join paths separately for benchmark D_1. Fig. 4 is generated based on the data in Tab. 3.

Table 3 Performance over benchmark D_1

dist	$\|A\|$	Distance Matrix / s	Join path / s
6	2901	29.64613223	0.003311872
7	2901	33.79274988	0.002006531
8	2900	38.34130168	0.002560616
9	2901	42.1935966	0.00298214
10	2900	46.7707684	0.003078938
11	2901	50.50927401	0.001685381
12	2902	54.73476863	0.002343655
13	2900	59.39891458	0.00369668
14	2902	63.04812098	0.003664494

It is evident that the time cost of creating distance matrix increases linearly as the *dist* parameter ranges from 5 to 14. In contrast, the time required for generating join paths is negligible compared to that of creating distance matrix. According to Algorithm 1, the distance matrix M^{dist} is generated iteratively by updating M^i, with the value of i increasing from 1 to *dist*. Hence, the time required to create the distance matrix varies linearly with the value of the *dist* parameter. After generating the distance matrix, the sequences of join tables and join attribute sets can be directly selected in a stepwise manner according to the rules provided in Theorem 1. Therefore, the processing speed of this step is much faster compared to the process of generating the distance matrix.

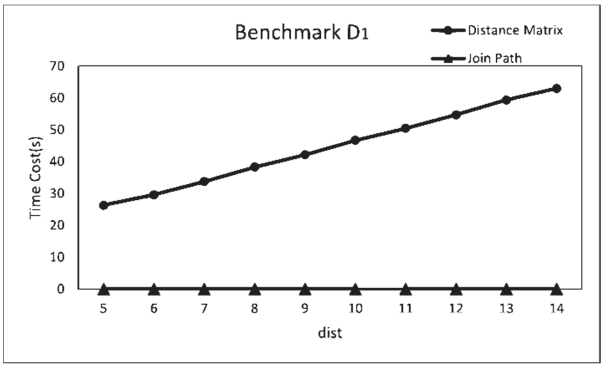

Figure 4 Benchmark D_1

Tab. 4 displays the time cost of creating the distance matrix and generating the join path separately for the D_2 benchmark. Fig. 5 is generated using the data in Tab. 4. It clearly indicates that the time cost of creating the distance matrix increases at a relatively high rate as the number of relations $|D|$ ranges from 200 to 1000. However, similar to Fig. 2, the time cost of generating the join path can be considered negligible when compared to that of creating the distance matrix. When creating the distance matrix, for each M^{dist}, the join reachable relationship between each pair of tables must be examined, thereby leading to a time complexity of $O(|D|^2)$. Consequently, the time cost of creating the distance matrix increases quadratically as the number of $|D|$ increases.

Table 4 Performance over benchmark D_2

$\|D\|$	$\|A\|$	Distance Matrix / s	Join path / s
200	1117	6.836089134	0.001358747
300	1706	14.62171721	0.001804829
400	2301	31.27872896	0.003068686
500	2901	51.86101723	0.00356245
600	3500	84.64539766	0.003222704
700	4100	133.1160092	0.003192425
800	4700	275.8449361	0.003716946
900	5300	300.3867085	0.004057884
1000	5900	365.5985801	0.004171848

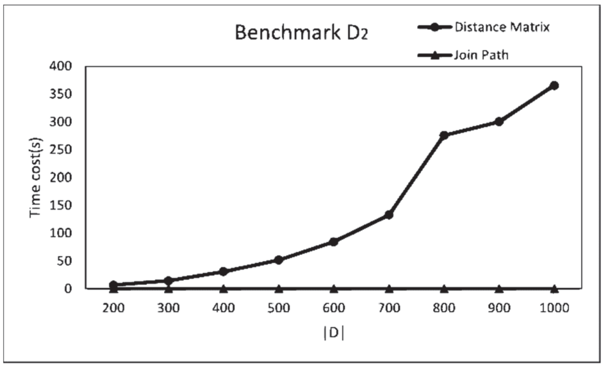

Figure 5 Benchmark D_2

Tab. 5 presents the time cost of creating the distance matrix and generating the join path separately for the D_3 benchmark. Fig. 6 is generated using the data in Tab. 5. It clearly indicates that the time cost of creating the distance matrix increases linearly as the number of attributes $|A|$ increases. Moreover, the time cost of generating the join path can be considered negligible when compared to that of creating the distance matrix. When finding each pair of reachable tables, all the attributes must be scanned to decide whether to update their distance to these two tables. Thus, the time cost of creating the distance matrix is linearly correlated with the number of attributes $|A|$.

Table 5 Performance over benchmark D_3

$\lvert A \rvert$	Distance Matrix / s	Join path / s
1400	36.85569358	0.004063368
1901	37.04789495	0.003875971
2401	41.27752137	0.003296614
2901	46.60218167	0.001378536
3401	52.66672707	0.002160072
3900	58.85267878	0.00421834
4401	65.57376051	0.003408194
4900	72.62939739	0.003342152
5400	81.21669555	0.003337622

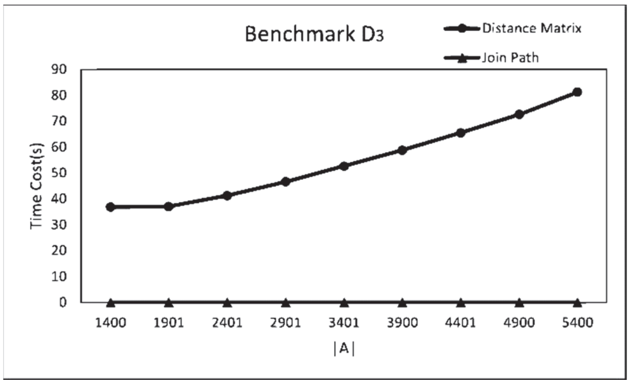

Figure 6 Benchmark D_3

Tab. 6 presents the time cost of creating the distance matrix and generating the join path separately over the D_4 benchmark. Fig. 7 is generated using the data in Tab. 6.

Table 6 Performance over benchmark D_4

Join sets	$\lvert A \rvert$	Distance Matrix / s	Join path / s
50	2950	47.46833539	0.003289223
100	2901	46.74626398	0.005617619
150	2858	46.39728141	0.005127907
200	2814	48.04738045	0.004442215
250	2783	51.58939457	0.004831553
300	2761	50.50927401	0.001685381
350	2738	51.0446527	0.004980803
400	2717	48.92301798	0.004063129
450	2698	44.30130768	0.004644871

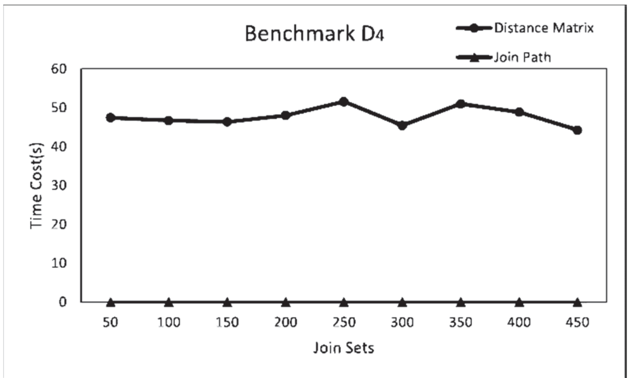

Figure 7 Benchmark D_4

It clearly indicates that the time cost of creating the distance matrix does not change significantly as the number of join sets increases. Furthermore, the time cost of generating the join path can be considered negligible when compared to that of creating the distance matrix. When constructing datasets manually, the number of attributes increases as the number of join sets decreases, because more relationship tables with common attributes are generated when the number of join sets decreases. Therefore, the variation in the time required for generating the distance matrix is essentially caused by the change in the number of attributes, and there is no direct correlation between the number of join sets and the time required for generating the distance matrix.

In conclusion, the time results reveal that the time required to generate the join path based on the distance matrix is negligible compared to the time for creating the distance matrix. The time complexity of creating the distance matrix is linearly related to *dist* and the number of attributes $|A|$, and quadratic to $|D|$. Although the time required to create the distance matrix increases significantly with an increase in $|D|$, *dist*, and $|A|$, the distance matrix needs updating only when the metadata of tables in data lakes updates.

4.2.2 Comparison of Automatically Join Queries Generation Method with Table Integration Method

We compare the proposed approach of automatic join queries generation method with the table integraion method over benchmark D_4. Each table contains approximately 10 K tuples. The results are shown in Tab. 7. Our approach includes the time cost for creating the distance matrix, the time cost for generating join queries, and the time cost for executing join queries. Time 1 encompasses the time cost for generating join queries and executing the quereis. The table integration method generates large tables over correlated tables, and executes simple selection queries over integrated large tables instead of executing join queries over correlated tables. Time 2 in Tab. 7 represents the time cost for executing simple select queries over the integrated large tables.

Table 7 Comparison over benchmark D_4

Join sets	$\lvert A \rvert$	Distance Matrix / s	Time 1 / s	Integration / s	Time 2 / s
50	2950	47.47	0.05	70.02	0.02
100	2901	46.75	0.05	57.4	0.02
150	2858	46.40	0.05	49.10	0.02
200	2814	48.05	0.05	42.40	0.02
250	2783	51.59	0.05	35.03	0.02
300	2761	50.51	0.05	29.13	0.02
350	2738	51.04	0.05	22.86	0.02
400	2717	48.92	0.05	14.71	0.02
450	2698	44.30	0.05	9.59	0.02

Tab. 7 reveals that: (1) the time cost for executing join queries or simple queries is significantly less than that for creating the distance matrix or data integration; (2) the time cost for data integration decreases as the number of join sets increases, since more join sets entail fewer correlated tables needing integration. Meanwhile, the time cost for creating the distance matrix does not change substantially. (3) The time for executing simple select queries is less than that for join queries; however, the time cost for join queries is still acceptable. Since our approach does not require updating the distance matrix when data updating occurs in data lakes.

The correctness of automatically generated join paths was verified through a manual check of the sequence of join tables and the sequence of join attribute sets. Additionally, the results of join queries over correlated tables are the same as the results of simple selection queries over integrated large tables. These results confirm the accuracy of our proposed algorithms.

5 CONCLUSION

We introduce a novel approach for searching information in correlated tables within data lakes, without requiring knowledge of their specific structures and correlations. With the generation of the distance matrix, the join path for a given pair of attributes that users are interested in can be produced quickly. Furthermore, our proposed solution also enables easier maintenance of the data lake, which is of great practical importance. However, there are still limitations to our study. In our method, only one join path is generated, however, multiple join paths may exist, which contain different data information. The automatically generated join queries in this paper do not capture all the potential correlated information. We will address this issue in future work.

Acknowledgements

This work was supported by the Hangzhou Philosophy and Social Science Planning Project (No. M22JC067); the Zhejiang Province Educational Science Planning Project (No. 2022SCG293); the National Natural Science Foundation of China (Nos. 61962029, 62262033, and 62362042); the MOE (Ministry of Education in China), Youth Fund Project of Humanities and Social Sciences Research (No. 21YJCZH096); the Jiangxi Provincial Natural Science Foundation of China (Nos. 20202BAB212006, 20202ACBL202005, 20224BAB202012, and 20232BAB202007); the Scientific and Technological Research Project of Jiangxi Provincial Education Department of China (Nos. GJJ201832, GJJ211815, and GJJ2201914); and the Jiangxi Provincial Social Science Foundation of China (No. 23GL52D).

6 REFERENCES

[1] Cui, Z., Lu Z., Yang, T., Yu J., Chi L., Xiao, Y., & Zhang, S. (2023). Privacy and Accuracy for Cloud-Fog-Edge Collaborative Driver-Vehicle-Road Relation Graphs. *IEEE Transactions on Intelligent Transportation Systems*, *24*(8), 8749-8761. https://doi.org/10.1109/TITS.2023.3254370

[2] Chen, Y., Yang, L., & Cui, Z. (2023). Tensor-Based Lyapunov Deep Neural Networks Offloading Control Strategy with Cloud-Fog-Edge Orchestration. *IEEE Transactions on Industrial Informatics*. https://doi.org/10.1109/TII.2023.3266401

[3] Zhang, S., Yang, L., Feng, J., Wei, W., Cui, Z., Xie, X., & Yan, P. (2021). A Tensor-Network-Based Big Data Fusion Framework for Cyber-Physical-Social Systems (CPSS). *Information Fusion*, *76*, 337-354. https://doi.org/ 10.1016/j.inffus.2021.05.014

[4] Zhang, Q., Geng, G., & Tu, Q. (2023). Association mining-based method for enterprise's technological innovation intelligent decision making under big data. International *Journal of Computers Communications & Control*, *18*(2), 5241. https://doi.org/10.15837/ijccc.2023.2.5241

[5] Nasrollahi, M. & Ramezani, J. (2020). A Model to Evaluate the Organizational Readiness for Big Data Adoption. *International Journal of Computers Communications & Control*, *15*(3), 3874. https://doi.org/10.15837/ijccc.2020.3.3874

[6] Brickley, D., Burgess, M., & Noy, N. (2019). Google dataset search: Building a search engine for datasets in an open web ecosystem. *Proceedings of the World Wide Web Conference.* https://doi.org/10.1145/3308558.3313685

[7] Ouellette, P., Sciortino, A., Nargesian, F., Bashardoost, B. G., Zhu, E., Pu, K. Q., & Miller, R. J. (2021). Ronin: Data lake exploration. *Proceedings of the VLDB Endowment*, *14*(12), 2863-2866. https://doi.org/10.14778/3476311.3476364

[8] Nargesian, F., Zhu, E., Miller, R. J., Pu, K. Q., & Arocena, P. C. (2019). Data lake management: Challenges and opportunities. *Proceedings of the VLDB Endowment*, *12*(12), 1986-1989. https://doi.org/10.14778/3352063.3352116

[9] Nargesian, F., Zhu, E., Pu, K. Q., & Miller, R. J. (2018). Table union search on open data. *Proceedings of the VLDB Endowment*, *11*(7), 813-825. https://doi.org/10.14778/3192965.3192973

[10] Crowd Flower (2017). 2017 data scientist report.

[11] Aamod, K., Roee, S., Wolfgang, G., & Ren´ee, J. M. (2022). Integrating data lake tables. *Proceedings of the VLDB Endowment*, *16*(4), 932-945. https://doi.org/10.14778/3574245.3574274

[12] Koutras, C., Siachamis, G., Ionescu, A., Psarakis, K., Brons, J., Fragkoulis, M., Lofi, C., Bonifati, A., & Katsifodimos, A. (2021). Valentine: Evaluating matching techniques for dataset discovery. *Proceedings of the IEEE 37th International Conference on Data Engineering (ICDE).* https://doi.org/10.1109/icde51399.2021.00047

[13] Su, W., Wang, J., & Lochovsky, F. (2006). Holistic schema matching for web query interfaces. *Proceedings of the Advances in Database Technology - EDBT Conference.* https://doi.org/10.1007/11687238_8

[14] Pei, J., Hong, J., & Bell, D. (2006). A novel clustering-based approach to schema matching. *Advances in Information Systems, 4243*, 60-69. https://doi.org/10.1007/11890393_7

[15] Yi Zhang, Z. G. I. (2020). Finding related tables in data lakes for interactive data science. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* https://doi.org/10.1145/3318464.3389726

[16] Miller, R. J. (2018). Open data integration. *Proceedings of the VLDB Endowment*, *11*(12), 2130-2139. https://doi.org/10.14778/3229863.3240491

[17] Aamod, K., Grace, F., Roee, S., Zixuan, C., Wolfgang, G., Ren´ee J, M., & Mirek, R. (2023). Santos: Relationship-based semantic table union search. *Proceedings of SIGMOD Conference.* https://doi.org/10.48550/arXiv.2209.13589

[18] Oliver Lehmberg, C. B. (2017). Stitching web tables for improving matching quality. *Proceedings of the VLDB Endowment*, *10*(11), 1502-1513. https://doi.org/10.14778/3137628.3137657

[19] Dong, Y., Takeoka, K., Xiao, C., & Oyamada, M. (2021). Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. *Proceedings of the IEEE 37th International Conference on Data Engineering (ICDE).* https://doi.org/10.1109/icde51399.2021.00046

[20] Zhang, S. & Balog, K. (2018). Ad hoc table retrieval using semantic similarity. *Proceedings of the 2018 World Wide Web Conference on World Wide Web.* https://doi.org/10.1145/3178876.3186067

[21] Galindo-Legaria, C. A. (1994). Outer joins as disjunctions. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* https://doi.org/10.1145/191843.191908

[22] Vijay Kumar, T. V., Shridhar, A., & Ghoshal, A. (2009). Computing full disjunction using cojo. *Information Technology and Management*, *10*(1), 3-20. https://doi.org/10.1007/s10799-008-0043-0
[23] Paganelli, M., Beneventano, D., Guerra, F., & Sottovia, P. (2019). Parallelizing computations of full disjunctions. *Big Data Research*, *17*, 18-31. https://doi.org/10.1016/j.bdr.2019.07.002
[24] Cappuzzo, R., Papotti, P., & Thirumuruganathan, S. (2020). Creating embeddings of heterogeneous relational datasets for data integration tasks. *Proceedings of the 2020 ACM SIGMOD International Conference on management of data.* https://doi.org/10.1145/3318464.3389742
[25] Deng, X., Sun, H., Lees, A., Wu, Y., & Yu, C. (2022). Turl: Table understanding through representation learning. *ACM SIGMOD record*, *51*(1), 33-40. https://doi.org/10.1145/3542700.3542709
[26] Devlin, J., Chang, W., Ming, Lee, K., & Toutanova, K. (2019). Bert: Pretraining of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* https://doi.org/10.48550/arXiv.1810.04805
[27] Zhang, L., Zhang, S., & Balog, K. (2019). Table2vec: Neural word and entity embeddings for table population and retrieval. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval.* https://doi.org/10.1145/3331184.3331333
[28] Tomas, M., Ilya, S., Kai, C., Greg S, C., & Jeff, D. (2013). Distributed representations of words and phrases and their compositionality. *Proceedings of the 26th International Conference on Neural Information Processing Systems.* https://doi.org/10.48550/arXiv.1310.4546
[29] Sehrish Munawar Cheema, I. M. P. & Saman, T. (2023). A natural language interface for automatic generation of data flow diagram using web extraction techniques. *Journal of King Saud University - Computer and Information Sciences*, *35*(12), 626-640. https://doi.org/10.1016/j.jksuci.2023.01.006
[30] Solanki, A. & Kumar, A. (2022). A system to transform natural language queries into sql queries. *International journal of information technology*, *14*(1), 437-446. https://doi.org/10.1007/s41870-018-0095-2
[31] Tripathi, R., Sinha, P., Balabantaray, R., Pragyan Pujari, S., Thind, P., & Kar, S. (2019). Generating structured database queries using deeply bidirectional natural language encodings. *2019 International Conference on Information Technology (ICIT).* https://doi.org/10.1109/icit48102.2019.00092
[32] Sangeetha, J. & Hariprasad, R. (2019). An intelligent automatic query generation interface for relational databases using deep learning technique. *International Journal of Speech Technology*, *22*(3), 817-825. https://doi.org/10.1007/s10772-019-09624-7
[33] Liner Yang, Y. L. M. S. N. Y. G. F. & Meishan, Z. (2018). Joint pos tagging and dependence parsing with transition based neural networks. *ACM Transactions on Audio, Speech and Language Processing (TASLP)*, *26*(8), 1352-1358. https://doi.org/10.1109/taslp.2017.2788181
[34] Bo, Y. & Dunlu, P. (2021). Sql automatic generation model for complex query requests. *Journal of Chinese Computer Systems*, *41*(11), 2446-2451. https://doi.org/ 10.3969/j.issn.1000-1220.2021.11.032
[35] Badhya, S. S., Prasad, A., Rohan, S., Yashwanth, Y. S., Deepamala, N., & Shobha, G. (2019). Natural language to structured query language using elastic search for descriptive columns. *Proceedings of the 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS).* https://doi.org/10.1109/CSITSS47250.2019.9031030

Contact information:

Caicai ZHANG, Lecturer, PhD
Zhejiang Institute of Mechanical and Electrical Engineering,
No. 528, Binwen Road, Hangzhou, Zhejiang 310053, China
zhangcaicai@zime.edu.cn

Chenglang LU, Associate professor, PhD
Zhejiang Institute of Mechanical and Electrical Engineering,
No. 528, Binwen Road, Hangzhou, Zhejiang 310053, China
luchenglang@zime.edu.cn

Zhuolin MEI, Lecturer, PhD
School of Computer and Big Data Science, Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: meizhuolin@126.com

Bin WU, Lecturer, PhD
School of Computer and Big Data Science, Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: wubincs@gmail.com

Jing YU, Lecturer, PhD
(Corresponding author)
School of Computer and Big Data Science, Jiujiang University,
No. 551, Qianjin East Road, Jiujiang, Jiangxi 332005, China
E-mail: yujingellemma@gmail.com