IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 23 June 2023, accepted 22 August 2023, date of publication 25 August 2023, date of current version 31 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3308908

RESEARCH ARTICLE

Supporting Schema References in Keyword Queries Over Relational Databases

PAULO MARTINS, ALTIGRAN SOARES DA SILVA, ARIEL AFONSO, JOÃO CAVALCANTI, AND EDLENO DE MOURA
Institute of Computing, Universidade Federal do Amazonas, Manaus 69080-900, Brazil

Corresponding author: Paulo Martins (paulo.martins@icomp.ufam.edu.br)

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES-PROEX)-Finance Code 001; the Amazonas State Research Support Foundation-FAPEAM-through the POSGRAD 22-23 project and the NeuralBond Project (Proc. 6607 UNIVERSAL); the Innovation Center on Artificial Intelligence for Health project (CIIA-Health 2020/09866-4) in partnership with the São Paulo Research Foundation (FAPESP), the Ministry of Science and Technology of Brazil (MCTI) and the Brazilian Internet Steering Committee (CGI); and an individual grant from the National Council for Scientific and Technological Development (CNPq) to Altigran da Silva (Proc. 307248/2019-4).

ABSTRACT Relational Keyword Search (R-KwS) systems enable naive/informal users to explore and retrieve information from relational databases without knowing schema details or query languages. They take a keyword query, locate their corresponding elements in the target database, and connect them using information on PK/FK constraints. Although there are many such systems in the literature, most of them only support queries with keywords referring to the contents of the database and just very few support queries with keywords refering the database schema. We propose Lathe, a novel R-KwS that supports such queries. To this end, we first generalize the well-known concepts of Candidate Joining Networks (CJNs) and Query Matches (QMs) to handle keywords referring to schema elements and propose new algorithms to generate them. Then, we introduce two major innovations: a ranking algorithm for selecting better QMs, yielding the generation of fewer but better CJNs, and an eager evaluation strategy for pruning void useless CJNs. We present experiments performed with query sets and datasets previously experimented with state-of-the-art R-KwS systems. Our results indicate that Lathe can handle a wider variety of queries while remaining highly effective, even for databases with intricate schemas.

INDEX TERMS Relational databases, keyword search, information retrieval.

I. INTRODUCTION

Keyword *search over relational databases* enables naive/informal users to retrieve information from relational databases (DBs) without any knowledge about schema details or query languages. The success of search engines shows that untrained users are at ease using keyword search to find information of interest.

However, this can be challenging because the information sought frequently spans multiple relations and attributes, depending on the schema design of the underlying DB. Therefore, Relational Keyword Search (R-KwS[1]) systems must automatically determine which pieces of information to retrieve from the database and how to connect them to provide a relevant answer to the user.

In general, keywords may refer to both database values in tuples and schema elements, such as relation and attribute names. For instance, consider the query "*will smith films*" over a database on movies. The keywords "*will*" and "*smith*" may refer to values of person names. The keyword "*films*" on the other hand is more likely to refer to the name of a relation about movies. Although a significant number of query keywords correspond to schema references [1], the majority of previous work on R-KwS does not support references to the schema.

Handling keywords that refer to schema elements makes R-KwS significantly more challenging than the usual setting where only keywords referring to attribute values are considered. Firstly, it increases the complexity of the search process by requiring an understanding of the underlying database schema and its structure. Secondly, keywords referring to the schema introduce semantic ambiguity, making it difficult to disambiguate between schema references and attribute

The associate editor coordinating the review of this manuscript and approving it for publication was Adnan Abid.

[1]A Table of Acronyms is shown in Appendix A.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

values. This ambiguity further complicates the search process and can lead to incorrect or incomplete results. Furthermore,integrating schema knowledge into the search process becomes crucial when handling schema references. Understanding PK/FK relationships and connecting relevant information adds an extra layer of complexity to the problem. Finally, ranking and relevance determination become more challenging when schema elements are involved. Existing systems may prioritize attribute values even if they do not provide useful answers. Accurately assessing relevance requires considering both attribute values and schema references. These challenges require dedicated techniques and algorithms specifically designed to handle schema references effectively.

In this work, we study new techniques for supporting schema references in keyword queries over relational databases. Specifically, we propose Lathe[2], a new R-KwS system to generate a suitable SQL query from a keyword query, considering that keywords refer either to instance values or schema elements. Lathe follows the *Schema Graph* approach for R-KwS systems [2], [3]. Given a keyword query, this approach consists of generating relational algebra expressions called *Candidate Joining Networks*[3] (CJNs), which are likely to express user intent when formulating the original query. The generated CJNs are *evaluated*, that is, they are translated into SQL queries and executed by a DBMS, resulting in several *Joining Networks of Tuples* (JNTs) which are collected and supplied to the user.

In the literature, the most well-known algorithm for CJN Generation is CNGen, which was first presented in the system DISCOVER [4], but was adopted by most R-KwS systems [5], [6], [7], [8]. Despite the possibly large number of CJNs, most works in the literature focused on improving CJN Evaluation and ranking of JNTs instead. Specifically, DISCOVER-II [6], SPARK [7], and CD [8] used information retrieval (IR) style score functions to rank the top-K JNTs. KwS-F [9] imposed a time limit for CJN evaluation, returning potentially partial results as well as a summary of the CJNs that have yet to be evaluated. Later, CNRank [10] introduces a CJN ranking, requiring only the top-ranked CJNs to be evaluated. MatCNGen [2], [11] proposed a novel method for generating CJNs that efficiently enumerated the possible matches for the query in the DB. These *Query Matches* (QMs) are then used to guide the CJN generation process, greatly decreasing the number of generated CJNs and improving the performance of CJN evaluation.

Among the methods based on the Schema Graph approach, Lathe is, to the best of our knowledge, the first method to address the problem of generating and ranking CJNs considering queries with keywords that can refer to either schema elements or attribute values. We revisited and generalized concepts introduced in previous approaches [2], [4], [10], [11], such as tuples-sets, QMs, and the CJNs themselves, to enable schema references. In addition, we proposed a more effective approach to CJN Generation that included two major innovations: QM ranking and Eager CJN Evaluation. Lathe roughly matches keywords to the values of the attributes or to schema elements. Next, the system combines the keyword matches into QMs that cover all the keywords from the query. The QMs are ranked and only the most relevant ones are used to generate CJNs. The CJN generation explores the primary key/foreign key relationships to connect all the elements of the QMs. In addition, Lathe employs an eager CJN evaluation strategy, which ensures that all CJNs generated will yield non-empty results when evaluated. The CJNs are then ranked and evaluated. Finally, the CJN evaluation results are delivered to the user. Unlike the previous methods, Lathe provides the user with the most relevant answer without relying on JNTs rankings. This is due to the effective rankings of QMs and CJNs that we propose, which are absent in the majority of previous work.

We performed several experiments to assess the effectiveness and efficiency of Lathe. First, we compared the results with those obtained with several previous R-KwS systems, including the state-of-the-art QUEST [12] system using a benchmark proposed by Coffman & Weaver [3]. Second, we assessed the quality of our ranking of QMs. The ranking of CJNs was then evaluated by comparing different configurations in terms of the number of QMs, the number of CJNs generated per QM, and the use of the eager evaluation strategy. Finally, we assessed the performance of each phase of Lathe, as well as the trade-off between quality and performance of various system configurations. Lathe achieved better results than all of the R-KwS systems tested in our experiments. Also, our results indicate that the ranking of QMs and the eager CJN evaluation greatly improved the quality of the CJN generation.

Our key contributions are: (i) a novel method for generating and ranking CJNs with support for keywords referring to schema elements; (ii) a novel algorithm for ranking QMs, which avoids the processing of less likely answers to a keyword query; (iii) an eager CJN evaluation for discarding spurious CJNs; (iv) a simple and yet effective ranking of CJNs which exploits the ranking of QMs.

The remainder of this paper is organized as follows: Section II reviews the related literature on relational keywords search systems based on schema graphs and support to schema references. Section IV summarizes all of the phases of our method, which are discussed in detail in Sections V-VII. Section VIII summarizes the findings of the experiments we conducted. Finally, Section IX summarizes the findings and outlines our plans for the future.

II. BACKGROUND AND RELATED WORK

In this section, we discuss the background and related work on keyword search systems over relational databases and on supporting schema references in such systems. For a more

[2]The name Lathe refers to the fact that our system assigns a structure or form to an unstructured keyword-based query.

[3]Most of the previous work uses the term *Candidate Networks* instead. Here, we use *Candidate Joining Networks* because we consider it more meaningful.

comprehensive view of the state-of-the-art in keyword-based and natural language queries over databases, we refer the interested reader to a recent survey [13].

A. RELATIONAL KEYWORD SEARCH SYSTEMS

Current R-KwS systems fall in one of two distinct categories: systems based on *Schema Graphs* and systems based on *Instance Graphs*. Systems in the first category are based on the concept of *Candidate Joining Networks* (CJNs), which are networks of joined relations that are used to generate SQL queries and whose evaluation return several *Joining Networks of Tuples* (JNTs) which are collected and supplied to the user. This method was proposed in DISCOVER [4] and DBXplorer [5], and it was later adopted by several other systems, including DISCOVER-II [6], SPARK [7], CD [8], KwS-F [9], CNRank [10], and MatCNGen [2], [11]. Systems in this category make use of the underlying basic functionality of the RDBMS by generating appropriate SQL queries to retrieve answers to keyword queries posed by users.

Systems in the second category are based on a structure called *Instance Graph*, whose nodes represent tuples associated with the keywords they contain, and the edges connect these tuples based on referential integrity constraints. BANKS [14], BANKS-II [15], BLINKS [16] and, Effective [17] use this approach to compute keyword queries results by finding subtrees in a data graph that minimizes the distance between nodes matching the given keywords. These systems typically generate the query answer in a single phase that combines the tuple retrieval task and the answer schema extraction. However, the Instance Graph approach requires a materialization of the DB and requests a higher computational cost to deliver answers to the user. Furthermore, the important structural information provided by the database schema is ignored, once the data graph has been built.

B. R-KwS SYSTEMS BASED ON SCHEMA GRAPHS

In our research, we focus on systems based on Schema Graphs, since we assume that the data we want to query are stored in a relational database and we want to use an RDBMS capable of processing SQL queries. Also, our work expands on the concepts and terminology introduced in DISCOVER [4], [6] and expanded in CNRank [10] and MatCNGen [2], [11]. This formal framework is used and expanded to handle keyword queries that may refer to attribute values or to database schema elements. As a result, we can inherit and maintain all guarantees regarding the generation of minimal, complete, sound, and meaningful CJNs.

The best-known algorithm for CJN Generation is CNGen, which was introduced in DISCOVER [4] but was later adopted as a default in most of the R-KwS systems proposed in the literature [5], [6], [7], [8]. To generate a complete, non-redundant set of CJNs, this algorithm employs a Breadth-First Search approach [18]. As a result, CNGen frequently generates a large number of CJNs, resulting in a costly CJN generation and evaluation process.

Initially, most of the subsequent work focused on the CJN evaluation only. Specifically, as many CJNs were generated by CNGen that should be evaluated, producing a larger number of JNTs, such systems as DISCOVER-II [6], SPARK [7], and CD [8] introduced algorithms for ranking JNTs using IR style score functions.

KwS-F [9] addressed the efficiency and scalability problems in CJN evaluation in a different way. Their approach consists of two steps. First, a limit is imposed on the time the system spends evaluating CJNs. After this limit is reached, the system must return the (possibly partial) top-K JNTs. Second, if there are any CJNs that have yet to be evaluated, they are presented to the user in the form of query forms, from which the user can choose one and the system will evaluate the corresponding CJN.

CNRank [10] proposed a method for lowering the cost of CJN evaluation by ranking them based on the likelihood that they will provide relevant answers to the user. Specifically, CNRank presented a probabilistic ranking model that uses a *Bayesian Belief Network* [19] to estimate the relevance of a CJN given the current state of the underlying database. A score is assigned to each generated CJN, so that only a few CJNs with the highest scores need to be evaluated.

MatCNGen [2], [11] introduced a match-based approach for generating CJNs. The system enumerates the possible ways which the query keywords can be matched in the DB beforehand, to generate query answers. MatCNGen then generates a single CJN, for each of these QMs, drastically reducing the time required to generate CJNs. Furthermore, because the system assumes that answers must contain all of the query keywords, each keyword must appear in at least one element of a CJN. As a result of the generation process avoiding generating too many keyword occurrence combinations, a smaller but better set of CJNs is generated.

Lastly, Coffman & Weaver [3] proposed a framework for evaluating R-KwS systems and reported experimental results over three representative standardized datasets they built, namely MONDIAL, IMDb, and Wikipedia, along with their respective query workloads. The authors compare nine R-KwS systems, assessing their effectiveness and performance in a variety of ways. The resources of this framework were also used in the experiments of several other studies on R-KwS systems [2], [7], [10], [11], [20].

C. SUPPORT TO SCHEMA REFERENCES IN R-KwS

Overall there are few systems in the literature that support schema references in keywords queries. One of the first such systems was BANKS [21], a R-KwS system based on Instance Graphs. However, hence the query evaluation with keywords matching metadata can be relatively slow, since a large number of tuples may be defined to be relevant to the keyword.

Support for schema references in keyword queries was extensively addressed by Bergamaschi et al. in Keymantic [1],

KEYRY [22], and QUEST [12]. All these systems can be classified as schema-based since they aim at generating a suitable SQL query given an input keyword query. They do not, however, rely on the concept of CJNs, as Lathe and all DISCOVER-based systems do. Keymantic [1] and KEYRY [22] consider a scenario in which data instances are not acessible, such as in databases on the hidden web and sources hidden behind wrappers in data integration settings, where typically only metadata is made available. Both systems rely on similarity techniques based on structural and lexical knowledge that can be extracted from the available metadata, e.g., names of attributes and tables, attribute domains, regular expressions, or from other external sources, such as ontologies, vocabularies, domain terminologies, etc. The two systems mainly differ in the way they rank the possible interpretations they generate for an input query. While Keymantic relies on an extension the authors proposed for the Hungarian algorithm, KEYRY is based on the Hidden Markov Model, a probabilistic sequence model, adapted for keyword query modeling. QUEST [12] can be thought of as an extension of KEYRY because it uses a similar strategy to rank the mappings from keywords to database elements. QUEST, on the other hand, considers the database instance to be accessible and includes features derived from it for ranking interpretations, in contrast to KEYRY.

From these systems, QUEST is the one most similar to Lathe. However, it is difficult to draw a direct comparison between the two systems as QUEST does not rely on the formal framework from CJN-related previous work [2], [4], [6], [10], [11] and it also resolves a smaller set of keyword queries then Lathe. QUEST, in particular, does not support keyword queries whose resolution necessitates SQL queries with self-joins. As a result, when comparing QUEST to other approaches, the authors limited the experimentation to 35 queries rather then the 50 included in the original benchmark [3], [12]. Lathe, on the other hand, supports all 50 queries.

Finally, there are systems that propose going beyond the retrieval of tuples that fulfill a query expressed using keywords and try to provide a functionality close to structured query languages. This is the case of SQAK [23] that allows users to specify aggregation functions over schema elements. Such an approach was later expanded in systems such as SODA [24] and SQUIRREL [25], which aim to handle not only aggregation functions, but also keywords that represent predicates, groupings, orderings and so on. To support such features, these systems rely on a variety of resources that are not part of the database schema or instances. Among these are conceptual schemas, generic and domain-specific ontologies, lists of reserved keywords, and user-defined metadata patterns. We see such useful systems as being closer to natural language query systems [13]. In contrast, Lathe, like any typical R-KwS system, aims at retrieving sets of JNTs that fulfill the query, and not computing results with the tuples. In addition, it does not rely on any external resources.

III. PROBLEM STATEMENT

Given a database that has n relations $R_1, \ldots, R_n$, where each relation has m_i attributes $a^i_1, \ldots, a^i_{m_i}$. Let a keyword query be a set of keywords $k_1, k_2, \ldots, k_n$. Answering a keyword query over the database means finding a set of relational algebra expressions that match the query, that is, they match each keyword to at least one database element, which can be the name of a relation, an attribute name, or a value of an attribute.

We represent these expressions with Candidate Joining Networks, where the nodes comprise selections or projections over relations, and the edges represent join operations. That is, C is a candidate joining network for Q if, for each $k \in Q$, exists at least one node u in C so that one of the following is true:

- $u = \sigma_{a \ni k}(R_u)$
- $u = \pi_a(R_u)$, where $k = a$
- $u = \sigma(R_u)$, where $k = R_u$

The first condition indicates whether a keyword matches the value of an attribute, while the second and third verifies if the keyword matches to an attribute name or a relation name, respectively.

For notational simplicity, we assume that the attributes of a primary to foreign key relationship have the same name, so we can freely join relations using natural joins. The generalization of the problem and the solution when these assumptions do not hold is trivial.

Also, for each edge $u \rightarrow v$ in C, there exists a primary to foreign key relationship from R_u to R_v, so that we can join $u \bowtie v$.

To ensure the connectivity, C may also have some nodes which are not associated with any keyword, but they act as intermediate tables for the join operations. An intermediate node $u = R_u$ cannot be a leaf in C, that is, its degree must be greater than 1.

IV. LATHE OVERVIEW

In this section we present an overview of Lathe. We begin by presenting a simple example of the task carried out by our system. For this, we illustrate in Figure 1 a simplified excerpt from the well-known IMDb.[4]

Let Q be the keyword query Q="*will smith films*", where the user wants the system to list the movies in which Will Smith appears. Notice that, informally, the terms "*will*" and "*smith*" are likely to match the contents of a relation from the DB, while the term "*films*" is likely to match the name of a relation or attribute.

As other methods previously proposed in the literature, such as CNGen [4] and MatCNGen [2], [11], the main goal of Lathe is, given a query such as Q, generating a SQL query that, when executed, fulfills the information needed for the user. The difference between Lathe and these previous methods is that they are not able to handle references to schema elements, such as "films" in Q.

[4]Internet Movie Database. https://www.imdb.com/interfaces/

PERSON

	ID	Name
t_1	1	Will Smith
t_2	2	Will Theakston
t_3	3	Maggie Smith
t_4	4	Sean Bean
t_5	5	Elijah Wood
t_6	6	Angelina Jolie

CHARACTER

	ID	Name
t_7	7	Agent J
t_8	8	Robert Neville
t_9	9	Marcus Flint
t_{10}	10	Minerva McGonagall
t_{11}	11	Boromir
t_{12}	12	Frodo Baggins
t_{13}	13	Jane Smith

MOVIE

	ID	Title	Year
t_{14}	14	Men in Black	1997
t_{15}	15	I am Legend	2007
t_{16}	16	Harry Potter and the Sorcerer's Stone	2001
t_{17}	14	The Lord of the Rings: The Fellowship of the Ring	2001
t_{18}	18	The Lord of the Rings: The Return of the King	2003
t_{19}	19	Mr. & Mrs. Smith	2005

ROLE

	ID	Name
t_{20}	20	Actor
t_{21}	21	Actress
t_{22}	22	Producer
t_{23}	23	Writer
t_{24}	24	Director
t_{25}	25	Editor

CASTING

	ID	PID	MID	ChID	RID
t_{26}	26	1	14	7	20
t_{27}	27	1	15	8	20
t_{28}	28	2	16	9	20
t_{29}	29	3	16	10	21
t_{30}	30	4	17	11	20
t_{31}	31	4	18	11	20
t_{32}	32	5	17	12	20
t_{33}	33	5	18	12	20
t_{34}	34	6	19	13	21

FIGURE 1. A simplified excerpt from IMDb.

```
SELECT m.title, p.name
FROM person p
JOIN casting c ON p.id=c.pid
JOIN movie m ON m.id = c.mid
WHERE p.name ILIKE '%will%'
AND p.name ILIKE '%smith%';
```

(a)

m.title	p.name
Men in Black	Will Smith
I am Legend	Will Smith

(c)

```
SELECT m.title, p1.name, p2.name
FROM person p1
JOIN casting c1 ON p1.id=c1.pid
JOIN movie m ON m.id = c1.mid
JOIN casting c2 ON m.id = c2.mid
JOIN person p2 ON p2.id=c2.pid
WHERE p1.name ILIKE '%will%'
AND p2.name ILIKE '%smith%'
AND p1.id<>p2.id;
```

(b)

m.title	p1.name	p2.name
Harry Potter and the Sorcerer's Stone	Will Theakston	Maggie Smith

(d)

FIGURE 2. SQL queries generated for the keyword query *"will smith movies"* and their returned results.

For query Q, two of the possible SQL queries that would be generated are presented in Figures 2 (a) (S_1) and (b) (S_2), whose respective results for the database of Figure 1 are presented in Figures 2(c) and (d). In the query S_1, the keywords "will" and "smith" match the value of a single tuple of relation PERSON, while the keyword "films" matches the name of the relation MOVIE. As a result, S_1 retrieves the movies which the person Will Smith was in, and thus, satisfies the original user intent. As for query S_2, the keywords "will" and "smith" match values of two different tuples in relation PERSON, that is, they refer to two different persons. The keyword "films" matches the name of the relation MOVIE again. Therefore, S_2 retrieves movies in which two different persons, whose names respectively include the terms "*will*" and "*smith*", participated in. In these case, the persons are Will Theakston and Maggie Smith.

As this example indicates, there may be several plausible SQL queries related to a given keyword query. Therefore, it is necessary to decide which alternative is more likely to fulfill the user intent. This task is also carried out by Lathe.

Next, we present an overview of the components and the functioning of Lathe.

TABLE 1. Keyword matched for the query *"will smith films"*.

Keywords	Type	Database Element	Algebra Expression
will smith	value	PERSON.name	$\sigma_{\texttt{name}\supseteq\{will,smith\}}(\texttt{PERSON})$
will	value	PERSON.name	$\sigma_{\texttt{name}\ni will}(\texttt{PERSON})$
smith	value	PERSON.name	$\sigma_{\texttt{name}\ni smith}(\texttt{PERSON})$
	value	CHARACTER.name	$\sigma_{\texttt{name}\ni smith}(\texttt{CHARACTER})$
	value	MOVIE.title	$\sigma_{\texttt{title}\ni smith}(\texttt{MOVIE})$
films	schema	MOVIE.self	$\sigma(\texttt{MOVIE})$

A. SYSTEM ARCHITECTURE

In this section, we present the overall architecture of Lathe. We base our discussion on Figure 3, which illustrates the main phases that comprise the operation of the method.

The process begins with an input keyword query posed by the user. The system then attempts to associate each of the keywords from the query with a database schema element, such as a relation or an attribute. The system relies on the DB schema, i.e., the names of relations and attributes, or on the DB instance, i.e., on the values of the attributes, for this. This phase, called *Keyword Matching* ①, generates sets of *Value-Keyword Matches* (VKMs), which associate keywords with sets of tuples whose attribute values contain these keywords, and *Schema-Keyword Matches* (SKMs), which associate keywords with names of relations or attributes deemed as similar to these keywords.

In Table 1 we show possible matches between keywords in the input query and the database elements. For example, the keywords "will smith" are found together in the values of the attribute `name` of the `PERSON` relation. The keyword "will" is also found alone in the values of `PERSON.name`, which is the case of the person Will Theakston present in instance shown in Figure 1. The term "*smith*" is can refer to either the name of a person, the name of a character or even the title of a movie, in this case "Mr. & Mrs. Smith". Since these keywords are part of attribute values, these matches are considered VKMs. In the case of the keyword "*films*", it actually matches the *name* of the `Movie` relation, which is why in Table 1 the keyword "*films*" matches `MOVIE.self`. Thus, this match is considered an SKM. The *Keyword Matching* phase is detailed in Section V.

In the next phase, *Query Matching* ②, Lathe generates combinations of VKMs and SKMs. In these combinations, we consider that all keywords in the query must be matched; in other words, the combination must be *total*. Furthermore, we also consider that all pairs of keywords and attributes are "useful"; that is, if we remove any of the pairs, this would result in a non-total combination. All combinations that satisfy both criteria are called *Query Matches* (QMs).

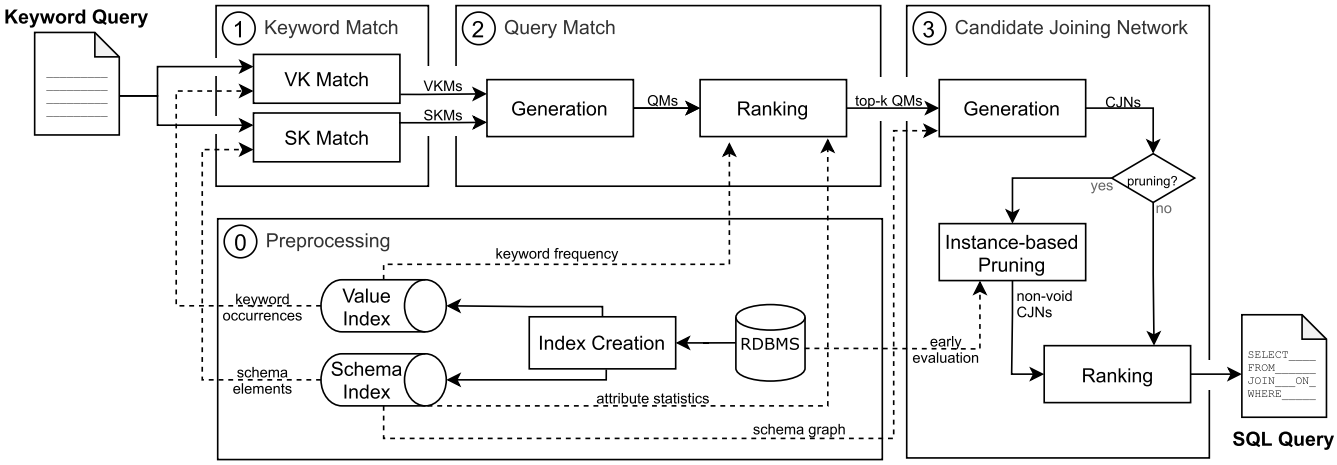

FIGURE 3. Main phases and architecture of Lathe.

will smith→PERSON.name
films→MOVIE
(a)

will→PERSON.name
smith→PERSON.name
films→MOVIE
(b)

will→PERSON.name
smith→CHARACTER.name
films→MOVIE
(c)

will→PERSON.name
smith→MOVIE.title
films→MOVIE
(d)

FIGURE 4. Examples of combinations of keywords matched.

In Figure 4 we present all possible QMs of the KMs illustrated in Table 1.

Although the Query Matching phase may generate a large number of QMs due to its combinatorial nature, only a few of them are useful in producing plausible answers to the user. As a result, we propose the first algorithm for *Ranking Query Matches* in the literature. This ranking assigns a score to QMs based on their likelihood of satisfying the needs of the user when formulating the keyword query. Thus, the system only outputs a few top-ranked QMs to the next phases. By doing so, it avoids having to process less likely QMs. We present the details on QMs, their generation, and ranking in Section VI.

Lastly, in the *Candidate Joining Network Generation* ③ phase, the system searches for interpretations for the keyword query. That is, the system tries to connect all the keyword matches from the QMs through CJNs, which are based on the schema graph. CJNs can be thought as relational algebra joining expressions that can be directly translated into SQL queries.

For instance, both the QMs shown in Figure 4 (a) and (b) can be connected using the `CASTING` relation, resulting in CJNs whose SQL translation is presented in Figure 2 (a) and (b), respectively.

Also, the system performs a *Candidate Joining Network Ranking*, which takes advantage of the previous QM rank, but also favors CJNs that are more concise in terms of the number of relations they employ. Once we have identified the most likely CJNs, they can be evaluated as SQL queries that are executed by a DBMS to the users. We notice that some of the generated CJNs may return empty results when they are evaluated. Thus, Lathe can alternatively evaluate CJNs before ranking them and prune such void CJNs. We call this process *instance-based pruning*.

During the whole process of generating CJNs, Lathe uses two data structures which are created in a *Preprocessing stage* ⓪: the *Value Index* and the *Schema Index*.

The Value Index is an inverted index that stores keyword occurrences in the database, indicating the relations, attributes, and tuples where a keyword appears. These occurrences are retrieved to generate VKMs. Furthermore, the Value Index is used to calculate *term frequencies* for the QMs and CJNs Rankings. The Schema Index is an inverted index that stores database schema information, as well as statistics about relations and attributes. While database schema information, such as PK/FK relationships, are used for the generation of CJNs, the statistics about attributes, such as *norm* and *inverted frequency*, are used for rankings of QMs and CJNs.

In the following sections we present each of the phases of Figure 3, describing the steps, definitions, data structures, and algorithms we used.

V. KEYWORD MATCHING

In this section, we present the details on keyword matches and their generation. Their role in our work is to associate each keyword from the query to some attribute or relation in the database schema. Initially, we classify them as either VKMs and SKMs, according to the type of associations they represent. Later, we provide a generalization of the keyword matches and we introduce the concept of *Keyword-Free Matches*, which will be used in the next phases of our method.

A. VALUE-KEYWORD MATCHING

We may associate the keywords from the query to some attribute in the database schema-based on the values of this

attribute in the tuples that contain these keywords using *value-keyword matches*, according to Definition 1.

Definition 1: Let Q be a keyword query and R be a relation state over the relation schema $R(A_1, \ldots, A_m)$. A ***value-keyword match*** *from R over Q is given by*:

$$R^V[A_1^{K_1}, \ldots, A_m^{K_m}] = \{t | t \in R \wedge \forall A_i : W(t[A_i]) \cap Q = K_i\}$$

where K_i is the set of keywords from Q that are associated to the attribute A_i, $W(t[A_i])$ returns the set of words in t for attribute A_i and V denotes a match of keywords to the database values.

Notice that each tuple from the database can be a member of only one value-keyword match. Therefore, the VKMs of a given query are **disjoint sets of tuples**.

Throughout our discussion, for the sake of compactness in the notation, we often omit mappings of attributes to empty keyword sets in the representation of a VKM. For instance, we use the notation $R^V[A_1^{K_1}]$ to represent $R^V[A_1^{K_1}, A_2^{\{\}}, \ldots, A_n^{\{\}}]$.

Example 1: Consider the database instance of Figure 1. The following VKMs can be generated for the query "will smith films".

$$PERSON^V[name^{\{will,smith\}}] = \{t_1\}$$
$$PERSON^V[name^{\{will\}}] = \{t_2\}$$
$$PERSON^V[name^{\{smith\}}] = \{t_3\}$$

VKMs play a similar role to the tuple-sets from related literature [2], [4]. They are, however, more expressive because they specify which attribute is associated with each keyword. Previous R-KwS systems based on the DISCOVER system, on the other hand, are unable to create tuple-sets that span multiple attributes [4], [6], [10]. Example 2 shows a keyword query that includes more than one attribute.

Example 2: Consider the query "lord rings 2001" whose intent is to return which Lord of the Rings movie was launched in 2001. We can represent it with the following value-keyword match:

$$MOVIE^V[title^{\{lord,rings\}}, year^{\{2001\}}] = \{t_{17}\}$$

The generation of VKMs uses a structure we call the *Value Index*. This index stores the occurrences of keywords in the database, indicating the relations and tuples a keyword appears and which attributes are mapped to the keyword. Lathe creates the Value Index during a preprocessing phase that scans all target relations only once. This phase comes before the query processing and it is not expected to be repeated frequently. As a result, without further interaction with the DBMS, answers are generated for each query. The Value Index has following the structure, which is shown in Example 3.

$$I_V = \{term : \{relation : \{attribute : \{tuples\}\}\}\}$$

Example 3: The VKMs presented in Example 1 are based on the following keyword occurrences:.

$$I_V[will] = \{PERSON : \{name : \{t_1, t_2\}\}\}$$
$$I_V[smith] = \{PERSON : \{name : \{t_1, t_3\}\}\}$$
$$I_V[smith][PERSON] = \{name : \{t_1, t_3\}\}$$
$$I_V[smith][PERSON][name] = \{t_1, t_3\}$$

In Lathe, the generation of VKMs is carried out by the VKMGen algorithm, presented in details in Appendix B.

B. SCHEMA-KEYWORD MATCHING

We may associate the keywords from the query to some attribute or relation in the database schema based on the name of the attribute or relation using *Schema-Keyword Matches*, according to Definition 2. Specifically, our method matches keywords to the names of relations and attributes using similarity metrics.

Definition 2: Let $k \in Q$ be a keyword from the query, $R(A_1, \ldots, A_m)$ be a relation schema. A ***schema-keyword match*** *from R over Q is given by*:

$$R^S[A_1^{K_1}, \ldots, A_m^{K_m}] = \{t | t \in R \wedge \forall k \in K_i : sim(A_i, k) \geq \varepsilon\}$$

where $1 \leq i \leq m$, K_i is the set of keywords from Q that are associated with the schema element A_i, $sim(A_i, k)$ gives the similarity between the name of a schema element A_i and the keyword k, which must be above a threshold ε, and S denotes a match of keywords to the database schema.

In this representation, we use the artificial attribute *self* when we match a keyword to the name of a relation. Example 4 shows an instance of a schema-keyword match wherein the keyword "*films*" is matched to the relation *MOVIE*.

Example 4: The following schema-based relation matches are created for the query "will smith films", considering a threshold $\varepsilon = 0.6$.

$$MOVIE^S[self^{\{films\}}] = \{t_{14}, t_{15}, t_{16}, t_{17}, t_{18}, t_{19}\}$$
$$MOVIE^S[title^{\{will\}}] = \{t_{14}, t_{15}, t_{16}, t_{17}, t_{18}, t_{19}\}$$
$$PERSON^S[name^{\{smith\}}] = \{t_1, t_2, t_3, t_4, t_5\}$$

where $sim(a, b)$ gives the similarity between the schema element a and the keyword b, $sim(movie, films) = 1.00$, $sim(title, will) = 0.87$ and $sim(name, smith) = 0.63$.

Despite their similarity to VKMs, the schema-keyword matches serve a different purpose in our method, ensuring that the attributes of a relation appear in the query results. As a result, they do not "filter" any of the tuples from the database, implying that they do not represent any selection operation over database relations.

1) SIMILARITY METRICS

For the matching of keywords to schema elements, we used two similarity metrics based on the lexical database *WordNet*: the *Path similarity* [26], [27] and the *Wu-Palmer similarity* [27], [28]. We introduce the WordNet database and the two similarity metrics below.

2) WORDNET DATABASE

WordNet [26] is a large lexical database that resembles a thesaurus, as it groups words based on their meanings. One use of WordNet is to measure similarity between words based on the relatedness of their *senses*, the many different meanings that words can have [29]. As a result, the word "film" can refer to a movie, as well as the act of recording or the plastic film. Each of these senses have a different relation to the sense of a "show". Wordnet represents sense relationships, such as *synonymy*, *hyponymy*, and *hypernymy*, to measure similarity between words. Synonyms are two word senses that share the same meaning. In addition, we say that the sense c_1 is a hyponym of the sense c_2 if c_1 is more specific, denoting a subclass of c_2. For instance, *"protagonist"* is a hyponym of *"character"*; *"actor"* is a hyponym of *"person"*, and *"movie"* is a hyponym of *"show"*. The hypernymy is the opposite of hyponymy relation. Thus, c_2 us a hypernymy of c_1.

3) PATH SIMILARITY

The Path similarity [26], [27] exploits the structure and content of the WordNet database. The relatedness score is inversely proportional to the number of nodes along the shortest path between the senses of two words. If the two senses are synonyms, the path between them has length 1. The relatedness score is calculated as follows:

$$sim_{path}(w_1, w_2) = \max_{\substack{c_1 \in senses(w_1) \\ c_2 \in senses(w_2)}} \left[\frac{1}{|shortest\_path(c_1, c_2)|} \right]$$

4) WU-PALMER SIMILARITY

The Wu-Palmer measure (WUP) [27], [28] calculates relatedness by considering the depths of the two synsets c_1 and c_2 in the WordNet taxonomies, along with the depth of the *Least Common Subsumer*(LCS). The most specific synset c_3 is the LCS, which is the ancestor of both synsets c_1 and c_2. Because the depth of the LCS is never zero, the score can never be zero (the depth of the root of a taxonomy is one). Also, the score is 1 if the two input synsets are the same. The WUP similarity for two words w_1 and w_2 is given by:

$$sim_{wup}(w_1, w_2) = \max_{\substack{c_1 \in senses(w_1) \\ c_2 \in senses(w_2)}} \left[2 \times \frac{depth(lcs(c_1, c_2))}{depth(c_1, c_2)} \right]$$

As in the case of VKMs, we detail the SKMGen algorithm used in Lathe in Appendix C.

C. GENERALIZATION OF KEYWORD MATCHES

Initially, we presented Definitions 1 and 2 which, respectively, introduce VKMs and SKMs. We chose to explain the specificity of these concepts separately for didactic purposes. They are, however, both components of a broader concept, *Keyword Match* (KM), which we define in Definition 3. In the following phases, this generalization will be useful when merging VKMs and SKMs.

*Definition 3: Let Q be a keyword query and R be a relation state over the relation schema $R(A_1, \ldots, A_m)$. Let $VKM = R^V[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]$ be a value-keyword match from R over Q. Let $SKM = R^S[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]$ be a schema-keyword match from R over Q. A general **keyword match** from R over Q is given by:*

$$R^S[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]^V[A_1^{K_1^V}, \ldots, A_m^{K_m^V}] = VKM \cap SKM$$

The representations of VKMs and SKMs in the general notation are given as follows:

$$R^S[A_1^{K_1}, \ldots, A_m^{K_m}] = R^S[A_1^{K_1}, \ldots, A_m^{K_m}]^V[A_1^{\{\}}, \ldots, A_m^{\{\}}]$$
$$R^V[A_1^{K_1}, \ldots, A_m^{K_m}] = R^S[A_1^{\{\}}, \ldots, A_m^{\{\}}]^V[A_1^{K_1}, \ldots, A_m^{K_m}]$$

Another concept required for the generation of QMs and CNs is *keyword-free matches*, which we describe in Definition 4. They are KMs that are not associated with any keyword but are used as auxiliary structures, such as intermediate nodes in CJNs.

Definition 4: We say that a keyword match KM given by:

$$KM = R^S[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]^V[A_1^{K_1^V}, \ldots, A_m^{K_m^V}]$$

*is a **keyword-free match** if, and only if, $\nexists K_i^S \neq \{\} \wedge \nexists K_i^V \neq \{\}$, where $1 \leq i \leq m$.*

For the sake of simplifying the notation, we will represent a keyword-free match as $R^S[\,]^V[\,]$ or simply by R.

VI. QUERY MATCHING

In this section, we describe the processes of generating and ranking QMs, which are combinations of the keyword matches generated in the previous phases that comprise every keyword from the keyword query.

A. QUERY MATCHES GENERATION

We combine the associations present in the KMs to form total and non-redundant answers for the user. In other words, Lathe looks for KM combinations that satisfy two conditions: (i) every keyword from the query must appear in at least one of the KMs and (ii) if any KM is removed from the combination, the combination no longer meets the first condition. These combinations, called *Query Matches* (QMs), are described in Definition 5

Definition 5: Let Q be a keyword query. Let $M = \{KM_1, \ldots, KM_n\}$ be a set of keyword matches for Q in a certain database instance I, where:

$$KM_i = R_i^S[A_{i,1}^{K_{i,1}^S}, \ldots, A_{i,m_i}^{K_{i,m_i}^S}]^V[A_{i,1}^{K_{i,1}^V}, \ldots, A_{i,m_i}^{K_{i,m_i}^V}]$$

*Also, let $C_{KM_i} = \bigcup_{\substack{1 \leq j \leq m_i \\ X \in \{S,V\}}} K_{i,j}^X$ and $C_M = \bigcup_{1 \leq i \leq n} C_{KM_i}$ be the sets of all keywords associated with KM_i and with M, respectively. We say that M is a **query match** for Q if, and only if, C_M forms a **minimal set cover** of the keywords in Q. That is, $C_M = Q$ and $C_M \setminus C_{KM_i} \neq Q, \forall KM_i \in M$.*

Notice that a QM cannot contain any keyword-free match, as it would not be minimal anymore. Example 5 presents combinations of KMs which are or are not QMs.

Example 5: Considering the KMs from the Examples 1 and 4, only some of the following sets are considered QMs for the query "will smith films":

$$M_1 = \{PERSON^V[name^{\{will,smith\}}], MOVIE^S[self^{\{films\}}]\}$$
$$M_2 = \{PERSON^V[name^{\{will\}}], PERSON^V[name^{\{smith\}}], MOVIE^S[self^{\{films\}}]\}$$
$$M_3 = \{PERSON^V[name^{\{will\}}], PERSON^V[name^{\{smith\}}]\}$$
$$M_4 = \{PERSON^V[name^{\{will,smith\}}], MOVIE^S[self^{\{films\}}], CHARACTER\}$$
$$M_5 = \{PERSON^V[name^{\{will,smith\}}], MOVIE^S[self^{\{films\}}], PERSON^V[name^{\{smith\}}]\}$$

The sets M_1 and M_2 are considered QMs. In contrast, the sets of keyword matches M_3, M_4 and M_5 are not QMs. While M_3 does not include all query keywords, M_4 and M_5 are not minimal, that is, they have unnecessary KMs.

We present the QMGen algorithm for generating QMs in Appendix D.

B. QUERY MATCHES RANKING

As described in Section IV, Lathe performs a ranking of the QMs generated in the previous step. This ranking is necessary because frequently many QMs are generated, yet, only a few of them are useful to produce plausible answers to the user.

Lathe estimates the relevance of QMs based on a *Bayesian Belief Network* model for the current state of the underlying database. In practice, this model assess two types of relevance when ranking query matches. The TF-IDF model is used to calculate the *value-based score*, which adapts the traditional Vector space model to the context of relational databases, as done in LABRADOR [30] and CNRank [10]. The *schema-based score*, on the other hand, is calculated by estimating the similarity between keywords and schema elements names.

In Lathe, only the top-k QMs in the ranking are considered in the succeeding phases. By doing so, we avoid generating CJNs that are less likely to properly interpret the keyword query.

Belief Bayesian Network:

We adopt the Bayesian framework proposed by [31] and [19] for modeling distinct IR problems. This framework is simple and allows for the incorporation of features from distinct models into the same representational scheme. Other keyword search systems, such as LABRADOR [30] and CNRank [10], have also used it.

In our model, we interpret the QMs as documents, which are ranked for the keyword query. Figure 5 illustrates an example of the adopted Bayesian Network. The nodes that represent the keyword query are located at the top of the network, on the Query Side. The Database Side, located at the bottom of the network, contains the nodes that represent the QM that will be scored. The center of the network is present on both sides and is made up of sets of keywords: the set V of all terms present in the values of the database and the set S of all schema element names.

In our Bayesian Network, we rank QMs based on their similarities with the keyword query. This similarity is interpreted as the probability of observing a query match QM given the keyword query Q, that is, $P(QM|Q) = \mu P(QM \wedge Q)$, where $\mu = 1/P(Q)$ is a normalizing constant, as used in [32].

Initially, we define a random binary variable associated with each keyword from the sets V and S, which indicates whether the keyword was observed in the keyword query. As these random variables are the root nodes of our Bayesian Network, all of the probabilities of the other nodes are dependent on them. Therefore, if we consider $v \subseteq V$ and $s \subseteq S$ as the sets of keywords observed, we can derive the probability of any non-root node x as follows: $P(x) = P(x|v, s) \times P(v) \times P(s)$.

As all the possibilities of v and s are equally likely *a priori*, we can calculate them as $P(v) = (1/2)^{|V|}$ and $P(s) = (1/2)^{|S|}$, respectively.

The instantiation of the root nodes of the network separates the query match nodes from the query nodes, making them mutually independent. Therefore:

$$P(QM \wedge Q) = P(Q|v, s)P(QM|v, s)P(v)P(s)$$

The probability of the keyword query $Q = \{q_1, \ldots, q_{|Q|}\}$ is split between the probability of each of its keywords:

$$P(Q|v, s) = \prod_{1 \le i \le |Q|} P(q_i|v, s)$$

A keyword q_i from the query is observed, given the sets s and v, either if q_i occurs in the values of the database or if q_i has a similarity above a threshold ε with a schema element.

$$P(q_i|v, s) = (q_i \in v) \veebar (\exists k \in s : sim(q_i, k) \ge \varepsilon)$$

Similarly, in our network, the probability of a query match QM is splited between the probability of each of its KMs.

$$P(QM|v, s) = \prod_{1 \le i \le |QM|} P(KM_i|v, s)$$

We compute the probability of KMs using two different metrics: a *schema score* based on the same similarities used in the generation of SKMs; and a *value score* based on a Vector model [33], [34] using the cosine similarity.

$$P(KM_i|v, s) = \prod_{\substack{1 \le j \le m_i \\ K^V_{i,j} \neq \emptyset}} cos(\overrightarrow{A_{i,j}}, \overrightarrow{v \cap K^V_{i,j}}) \times \prod_{\substack{1 \le j \le m_i \\ K^S_{i,j} \neq \emptyset}} \frac{\sum_{t \in s \cap K^S_{i,j}} sim(A_{i,j}, t)}{|s \cap K^S_{i,j}|}$$

where $KM_i = R^S_i[A^{K^S_{i,1}}_{i,1}, \ldots, A^{K^S_{i,m_i}}_{i,m_i}]^V[A^{K^V_{i,1}}_{i,1}, \ldots, A^{K^V_{i,m_i}}_{i,m_i}]$.

It is important to distinguish the documents from the Bayesian Network model and the Vector Model. The documents of the Bayesian Network are QMs, and the query is the keyword query itself, whereas the documents of the

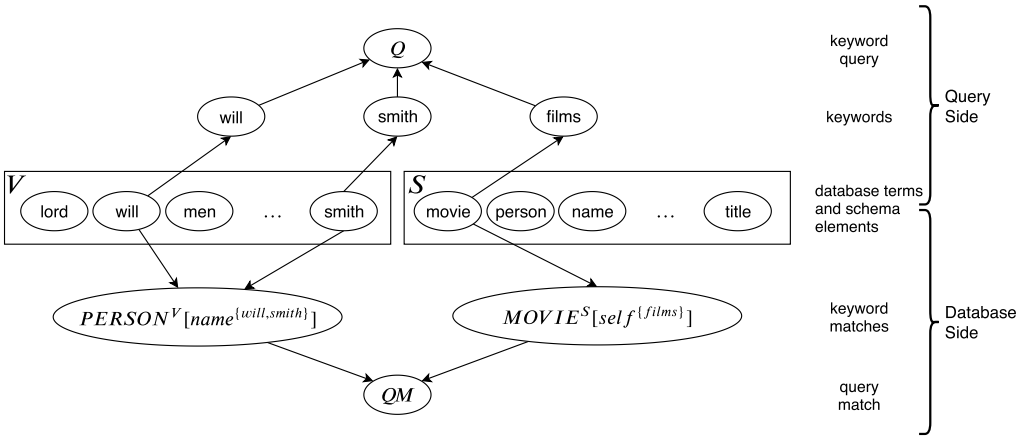

FIGURE 5. Bayesian network corresponding to the query "*will smith films*".

Vector model are database attributes, and the query is the set of keywords associated with the KM.

Once we know the document and the query of the Vector model, we can calculate the cosine similarity by taking inner product of the document and the query. The cosine similarity formula is given as follows:

$$cos(\overrightarrow{A_{i,j}}, \overrightarrow{v \cap K_{i,j}^V}) = (\overrightarrow{A_{i,j}^V} \cdot \overrightarrow{v \cap K_{i,j}^V})/(|\overrightarrow{A_{i,j}}| \times |\overrightarrow{v \cap K_{i,j}^V}|)$$
$$= \alpha \times \frac{\sum_{t \in V} w(\overrightarrow{A_{i,j}}, t) \times w(\overrightarrow{v \cap K_{i,j}^V}, t)}{\sqrt{\sum_{t \in V} w(\overrightarrow{A_{i,j}}, t)^2}}$$

where $\alpha = 1/(\sum_{t \in V} w(\overrightarrow{v \cap K_{i,j}^V}, t)^2)^{1/2}$ is the constant that represents the norm of the query, which is not necessary for the ranking.

The weights for each term are calculated using the TF-IDF measure. This measure is based on the term frequency and specificity in the collection. We use the *raw frequency* and *inverse frequency*, which are the most recommended form of TF-IDF weights [33].

$$w(\vec{X}, t) = freq_{X,t} \times \log \frac{N_A}{n_t}$$

where $\vec{X} \in \{\overrightarrow{A_{i,j}}, \overrightarrow{v \cap K_{i,j}^V}\}$ can be either the document or the query, N_A is the number of attributes in the database, and n_t is the number of attributes that are mapped to the occurrences of the term t. In the case of $\vec{X}$ be the query, $freq_{X,t}$ gives the number of occurrences of a term t in the keyword query, which is generally 1. In the case of $\vec{X}$ be an attribute(document), $freq_{X,t}$ gives the occurrences of a term t in an attribute, which is obtained from the Value Index.

We present the algorithm for ranking QMs in Appendix E.

VII. CANDIDATE JOINING NETWORKS

In this section we present the details on our method for generating and ranking Candidate Joining Networks (CJNs), which represent different interpretations of the keyword query. We recall that our definition of CJNs expands on the definition presented in [4] to support keywords referring to schema elements.

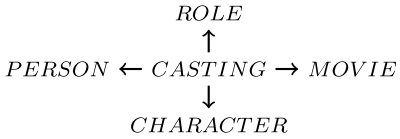

FIGURE 6. A schema graph for the sample movie database of Figure 1.

The generation of CJNs uses a structure we call a *Schema Graph*. In this graph, there is a node representing each relation in the database and the edges correspond to the *referential integrity constraints* (RIC) in the database schema. In practice, this graph is built in a preprocessing phase based on information gathered from the database schema.

Definition 6: Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a set of relation schemas from the database. Let E be a subset of the ordered pairs from $\mathcal{R}^2$ given by:

$$E = \{\langle R_a, R_b \rangle | \langle R_a, R_b \rangle \in \mathcal{R}^2 \wedge R_a \neq R_b \wedge RIC(R_a, R_b) \geq 1\}$$

where $RIC(R_a, R_b)$ gives the number of Referential Integrity Constraints from a relation R_a to a relation R_b. We say that a ***schema graph*** *is an ordered pair $G_S = \langle \mathcal{R}, E \rangle$, where $\mathcal{R}$ is the set of vertices (nodes) of G_S, and E is the set of edges of G_S.*

Example 6: Considering the sample movie database introduced in Figure 1, our method generates the schema graph below.

$$G_S = < \{PERSON, MOVIE, CASTING, CHARACTER, ROLE\}, \{\langle CASTING, PERSON \rangle, \langle CASTING, MOVIE \rangle, \langle CASTING, CHARACTER \rangle, \langle CASTING, ROLE \rangle\} >$$

In Figure 6, we represent a graphical illustration of G_S.

Once we defined the schema graph, we can introduce an important concept, the *Joining Network of Keyword Matches*

(JNKM). Intuitively, a joining network of keyword matches J contains every KM from a query match M. J may also contain some free-keyword matches for the sake of connectivity. Finally, J is a connected graph that is structured according to the schema graph G_S. The definition of joining network of keyword matches is given as follows:

Definition 7: Let M be a query match for a keyword query Q. Let G_S be a schema graph. Let F be a set of keyword-free matches from the relations of G_S. Consider a connected graph of keyword matches $J = \langle \mathcal{V}, E \rangle$, where $\mathcal{V}$ and E are the vertices and edges of J. We say that J is a ***joining network of keyword matches*** *from M over G_S if the following conditions hold*:

$$i) \mathcal{V} = M \cup F$$
$$ii) \forall \langle KM_a, KM_b \rangle \in E \implies \exists \langle R_a, R_b \rangle \in G_S$$

For the sake of simplifying the notation, we will use a graphical illustration to represent JNKMs, which is shown in Example 7.

Example 7: Considering the query match M_1 previously generated in Example 5, the following JNKMs can be generated:

$$J_1 = PERSON^V[name^{\{will,smith\}}] \leftarrow CASTING \downarrow MOVIE^S[self^{\{films\}}]$$

$$J_2 = CHARACTER \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}] \downarrow PERSON^V[name^{\{will,smith\}}]$$

The JNKMs J_1 and J_2 cover the query match M_1. The interpretation of J_1 looks for the movies *of the* person *will smith. J_2 looks for the* movies *of the* person *will smith and which* character *will smith played in these movies.*

Notice that a JNKM might have unnecessary information for the keyword query, which was the case of J_2 presented in Example 7. One approach to avoid generating unnecessary information is to generate Minimal Joining Networks of Keyword Matches (MJNKM), which are addressed in Definition 8. Roughly, a MJNKM cannot have any keyword-free match as a leaf, that is, a keyword-free match incident to a single edge.

Definition 8: Let G_S be a schema graph. Let M be a query match for a query Q. We say that $J = \langle \mathcal{V}, E \rangle$ from M over G_S is ***minimal joining network of keyword matches*** *(MJNKM) if, and only if, the following condition holds*:

$$\forall KM_i \in \mathcal{V} \; (\; \exists! \langle KM_a, KM_b \rangle \in E | i \in \{a, b\} \implies KM_i \neq R_i^S[\,]^V[\,] \;)$$

Example 8: Considering the query match M_2 previously generated in Example 5, the following MJNKMs can be generated:

$$J_3 = \begin{array}{c} PERSON^V[name^{\{will\}}] \\ \uparrow \\ CASTING \rightarrow MOVIE^S[self^{\{films\}}] \\ \downarrow \\ PERSON^V[name^{\{smith\}}] \end{array}$$

Another issue that a JNKM might have is representing an inconsistent interpretation. For instance, it is impossible for J_3 presented in Example 8 to return any results from the database. By Definition 1, the VKMs $PERSON^V[name^{\{will\}}]$ and $PERSON^V[name^{\{smith\}}]$ are disjoint. However, a tuple from *CASTING* cannot refer to two different tuples of *PERSON*. Thus J_3 is inconsistent. We notice that previous work in literature for CJN generation had addressed this kind of inconsistency [2], [4]. They did not, however, consider the situation in which there exist more than one RIC from one relation to another. In contrast, based on the theorems and definitions presented in [4], Lathe proposes a novel approach for checking consistency in CJNs that support such scenarios. Theorem 1 presents a criterion that determines when a JNKM is *sound*, that is, it can only produce JNTs that do not have more than one occurrences of a tuple. The proof of Theorem 1 is presented in AppendixF.

Theorem: Let $G_S = \langle \mathcal{R}, E_G \rangle$ be a schema graph. Let $J = \langle \mathcal{V}, E_J \rangle$ be a joining network of keyword matches. We say that J is ***sound****, that is, it does not have more than one occurrences of the same tuple for every instance of the database if, and only if, the following condition holds $\forall KM_a \in \mathcal{V}, \forall \langle R_a, R_b \rangle \in E_G$:*

$$RIC(R_a, R_b) \geq |\{KM_c | \langle KM_a, KM_c \rangle \in E_J \wedge R_c = R_b\}|$$

where $RIC(R_a, R_b)$ indicates the number of *Referential Integrity Constraints* from a relation R_a to a relation R_b.

Example 9 presents a JNKM that is sound, although it would be deemed not sound by previous approaches [2], [4].

Example 9: Consider a simplified excerpt from the MONDIAL database [35], presented in Figure 7. As there exists 2 RICs from the relation BORDER to COUNTRY, represented by the attributes Ctry1_Code *e* Ctry2_Code*, a tuple from BORDER can be joined to at most two distinct tuples from Country, which is the case of $t_{35} \bowtie t_{38} \bowtie t_{36}$. Thus, the following MJNKM is sound*:

$$J_4 = COUNTRY^V[name^{\{colombia\}}] \leftarrow BORDER \downarrow COUNTRY^V[name^{\{brazil\}}]$$

Finally, Definition 9 describes a Candidate Joining Network (CJN), which is roughly a sound minimal joining network of keyword matches.

Definition 9: Let M be a query match for the keyword query Q. Let G_S be a schema graph. Let CJN be a joining network of keyword matches from M over G_S given by CJN =

COUNTRY

	Code	Name	Capital_ID
t_{35}	CO	Colombia	1
t_{36}	BR	Brazil	2
t_{37}	PE	Peru	3

BORDER

	Ctry1_Code	Ctry2_Code	Length
t_{38}	CO	BR	1643
t_{39}	PE	BR	1560

CITY

	ID	Name	Population
t_{40}	1	Bogota	1643
t_{41}	2	Brasilia	1560
t_{42}	3	Lima	1560

FIGURE 7. A simplified excerpt from MONDIAL.

$\langle \mathcal{V}, E \rangle$. *We say that CJN is a* ***candidate joining network*** *if, and only if, CJN is minimal and sound.*

Example 10: Considering the query match M_2 previously generated in Example 5, the following CJN can be generated:

$$CJN_1 = \begin{array}{ccc} & CASTING & \\ \swarrow & & \searrow \\ PERSON^V[name^{\{will\}}] & & MOVIE^S[self^{\{films\}}] \leftarrow CASTING \\ & & \searrow \\ & & PERSON^V[name^{\{smith\}}] \end{array}$$

The candidate joining networks CJN_1 covers the query match M_2. CJN_1 is a minimal and sound JNKM. The interpretation of CJN_1 searches for the movies where both persons "will" (e.g. Will Theakston) and "smith" (e.g. Maggie Smith) participate in. The two keyword-free matches from the CASTING are treated as different nodes in the candidate joining network CJN_1.

The details on how we generate CJNs in Lathe are described by the CNKMGen Algorithm in Appendix G.

A. CANDIDATE JOINING NETWORK RANKING

In this section, we present a novel ranking of CJNs based on the ranking of QMs. This ranking is necessary because often many CJNs are generated, yet, only a few of them are indeed useful to produce relevant answers.

We present in Section VI-B a QM ranking that advances the majority of the features present in the ranking of CJNs of other proposed systems, such as CNRank [10]. Thus, we can exploit the scores of the QMs to rank the CJNs. For this reason, our CJN ranking strategy is straightforward yet effective. Roughly, it uses the ranking of QMs adding a penalization for large CJNs. Therefore, the score of a candidate joining network CJN_M from a query match M is given by:

$$score(CJN_M) = score(M) \times \frac{1}{|CJN_M|}$$

To ensure that CJNs with the same score are placed in the same order that they were generated we used a stable sorting algorithm [18].

B. CANDIDATE JOINING NETWORK PRUNING

In this section we present an *eager evaluation* strategy for pruning CJNs. Even if CJNs contain valid interpretations of the keyword query, some of them may fail to produce any JNTs as a result. Thus, we can improve the results of our CJN generation and ranking if by pruning what we call *void* CJNs, which are CJNs with no JNTs in their results.

Example 11: Considering the database instance of Figure 1 and the keyword query "will smith films", the following CJNs can be generated:

$$CJN_2 = \begin{array}{ccc} PERSON^V[name^{\{will\}}] & \leftarrow & CASTING \\ & & \downarrow \\ & & MOVIE^S[self^{\{films\}}]^V[name^{\{smith\}}] \end{array}$$

$$CJN_3 = \begin{array}{ccc} & CASTING & \\ \swarrow & & \searrow \\ PERSON^V[name^{\{will\}}] & \leftarrow MOVIE^S[self^{\{films\}}] \leftarrow & CHARACTER^V[name^{\{smith\}}] \end{array}$$

The interpretation of CJN_2 looks for the movies whose name contains the keyword "smith" (e.g. "Mr. & Mrs. Smith") and in which a person whose contains "will" (e.g. "Will Theakston") participate in. The interpretation of CJN_3 looks for the movies where a person whose name contains "will" (e.g. "Will Theakston") played the character "smith" (e.g. "Jane Smith"). Notice that although the candidate joining networks CJN_2 and CJN_3 both provide valid interpretations for the keyword query, they do not produce any tuples as a result in the given database instance.

As most of the previous work does not rank CJNs but only evaluates them and ranks their resulting JNTs instead, the pruning of void CJNs has previously never been addressed. Lathe employs a pruning strategy that evaluates CJNs as soon as they are generated, pruning the void ones. This strategy, as demonstrated in our experiments, can significantly improve the quality of the CJN generation process, particularly in scenarios where the schema graph contains a large number of nodes and edges.

For instance, one of the datasets we use in our experiments, the MONDIAL database, contains a large number of relations and relational integrity constraint (RICs). This results in a schema graph with several nodes and edges, which, intuitively, incur a large number of possible CJNs for a single QM. In contrast, we discovered that such schema graphs are prone to produce a large number of void CJNs. In particular, while approximately 20% of the keyword queries used in our experiments required us to consider 9 CJNs per QM, the eager evaluation strategy reduced this value to 2 CJNs per QM.

Notice, however, that to find if some CJN is void, we must execute it as an SQL in the DBMS, which incurs an additional cost and an increase in the CJN generation time. Despite that, we notice in our experiments that the eager evaluation strategy does not necessarily hinder the performance of a R-KwS system. In fact, the reducing the number of CJNs per QM alone improves the system efficiency because this parameter influences the CJN generation process. Furthermore, the eager evaluation advances the CJN evaluation, which is already a required step in the majority of R-KwS systems in

TABLE 2. Datasets we used in our experiments.

Dataset	Size(MB)	Relations	Attributes	RIC	Tuples
IMDb	701	6	33	5	1,673,076
MONDIAL	14	28	48	38	17,115
Yelp	7898	7	24	5	12,856,448

the related work. Lastly, we can set a maximum number of CJNs to probe during the eager evaluation, which limits the increase in CJN generation time.

VIII. EXPERIMENTS

In this section, we report a set of experiments performed using datasets and query sets previously used in similar experiments reported in the literature. Our goal is to evaluate the quality of the CJN Ranking, the quality QM ranking, and how our Eager Evaluation strategy can improve the CJN Generation.

A. EXPERIMENTAL SETUP

1) SYSTEM DETAILS

We ran the experiments on a Linux machine running Artix Linux (64-bit, 32GB RAM, AMD Ryzen™ 5 5600X CPU @ 3.7GHz) We used PostgreSQL as the underlying RDBMS with a default configuration. All implementations were made in Python 3.

2) DATASETS

For all the experiments, we used three datasets, *IMDb*, *MONDIAL*, and *Yelp*, which were used for the experiments performed with previous R-KwS systems and methods [2], [3], [7], [10], [11], [20], [36]. The IMDb dataset is a subset of the well-known Internet Movie Database (IMDb)[5], which comprises information related to films, television shows, and home videos – including actors, characters, etc. The MONDIAL dataset [35] comprises geographical and demographic information from the well-known *CIA World Factbook*[6], the International Atlas, the TERRA database, and other web sources.

The Yelp dataset is a subset of Yelp[7], which comprises information about businesses, reviews, and user data. The three datasets have distinct characteristics. The IMDb dataset has a simple schema, but query keywords often occur in several relations. Although the MONDIAL dataset is smaller, its schema is more complex or dense, with more relations and relational integrity constraints (RICs). The Yelp dataset has the highest number of tuples but its schema is simple. Table 2 summarizes the details of each dataset.

3) QUERY SETS

We used the query sets provided by Coffman & Weaver [3] benchmark for the IMDb and MONDIAL datasets. The query set for Yelp was obtained from SQLizer [37] and consists of 28 queries formulated in Natural Language. We adapted all of its queries to our experiments by extracting only their keyword terms.

However, we notice that several queries from IMDb and MONDIAL query sets do not have a clear intent, compromising the proper evaluation of the results, for instance, the ranking of CJNs. Therefore, for the sake of providing a more fair evaluation, we generated an additional for each original query set replacing queries that we consider unclear with equivalent queries with added schema references. As an example, consider the query "*Saint Kitts Cambodia*" for the MONDIAL dataset, where *Saint Kitts* and *Cambodia* are the names of the two countries. There exist several interpretations of this keyword query, each of them with a distinct way to connect the tuples corresponding to these countries. For example, one might look for shared religions, languages, or ethnic groups between the two countries. While all these interpretations are valid in theory, the relevant interpretation defined by Coffman & Weaver [3] in their golden standard indicates that the query searches for organizations in which both countries are members. In this case, we replaced in the new query set with the query "Saint Kitts Cambodia Organizations".

TABLE 3. Query sets we used in our experiments.

Query Set	Target Dataset	Total Queries	Ambiguous Queries	Schema References
IMDb	IMDb	50	5	20
IMDb-DI	IMDb	50	-	25
MONDIAL	MONDIAL	50	7	12
MONDIAL-DI	MONDIAL	50	-	19
Yelp	Yelp	28	-	24

Table 3 presents the query sets we used in our experiments, along with some of their features. Query sets whose names include the suffix "-DI" correspond to those in which we have replaced ambiguous queries as explained above. Thus, these queries sets have no ambiguous queries and they have a higher number of Schema References.

4) GOLDEN STANDARDS

The benchmark from Coffman & Weaver [3] provided the relevant interpretation and its relevant SQL results for each query of the IMDb and MONDIAL datasets. In the case of the Yelp dataset, SQLizer [37] provided the relevant SQL queries for natural language queries. Since we derived keyword queries from the latter, we also adapted the SQL queries to reflect this change. We then manually generated the golden standards for CJNs and QMs using relevant SQL provided by Coffman & Weaver and in SQLizer.

5) METRICS

We evaluate the ranking of CJNs and QMs using three metrics: Precision at ranking position 1 ($P@1$), Recall,

[5]https://www.imdb.com/
[6]https://www.cia.gov/library/publications/the-world-factbook/
[7]https://www.yelp.com/dataset

Recall at ranking position K ($R@K$), and Mean Reciprocal Rank (MRR).

Precision at 1 ($P@1$) is the ratio of relevant results found in the first position for each query to the number of queries. Recall is the ratio of relevant results retrieved to the total number of relevant results. Recall at K ($R@K$) is the mean recall across multiple queries considering only first K results. If fewer than K results are retrieved by a system, we calculate the recall value at the last result. For instance, if the system returns the relevant CJN in at most position 3 of the ranking for 35 out of 50 queries, then the system would obtain an $R@3$ of 0.7.

The Mean Reciprocal Ranking (MRR) value indicates how close the correct CJN is from the first position of the ranking. Given a keyword query Q, the value of the *reciprocal ranking for* Q is given by $RR_Q = \frac{1}{K}$, where K is the rank position of the relevant result. Then, the MRR obtained for the queries in a query set is the average of RR_Q, for all Q in the query set.

6) LATHE SETUP

For the experiments we report here, we set a maximum size for QMs and CJNs of 3 and 5, respectively. Also, we consider three important parameters for running Lathe: N_{QM}, the maximum number of QMs considered from the QM ranking; N_{CJN}, the maximum number of CJNs considered from each QM; and P_{CJN}, the number of CJNs probed per QM by the eager evaluation. In this context, a *setup* for Lathe is a triple $N_{QM}/N_{CJN}/P_{CJN}$. The most common setup we used in our experiments is $8/1/9$, in which we take the top-5 QMs in the ranking, generate and probe up to 9 CJNs for each QM, and take only the first non-empty CJN, if any, from each QM. We call this the *default setup*. Later in this section, we will discuss how these parameters affect the effectiveness and the performance of Lathe, as well as why we use the default configuration.

All the resources, including source code, query sets, datasets and golden standards used in our experiments are available at https://github.com/pr3martins/Lathe.

B. PRELIMINARY RESULTS

We present in this section some statistics about the CJN generation process. Table 4 shows the maximum and average numbers of KMs, QMs, and CJNs generated for each query set. The last two columns refer to the ratio of the number of CJNs to the number of QMs. Notice that we removed the maximum caps for the number of CJNs and CJNs per QM in the experiment reported here. However, we maintained the limit sizes of 3 and 5 for the QMs and CJNs, respectively.

Overall, the query sets for both IMDb and Yelp datasets achieved higher maximum and average numbers of KMs and QMs. This result is due to a higher number of tuples and the keywords being present in multiple relations or combinations. For example, in the IMDb dataset, several persons, characters, and even movies share the same name or part of it. In the case of Yelp, for instance, the keyword *"texas"* can match a state, a restaurant name, or a username. On the other hand, in MONDIAL, the keywords often match a few attributes only. For example, a city name probably does not overlap with the names of countries, continents, etc. Consequently, the system produces a low number of KMs and QMs for the query sets of this dataset.

Regarding the CJN generation, the query sets for IMDb and Yelp achieved high numbers of CJNs because of their already high numbers of QMs, but a low ratio of CJNs to QMs due to their simple schema graphs. As for the query sets for the MONDIAL dataset, they achieved opposite results due to their complex schema graph.

TABLE 4. Statistics for the CJN process of each query set.

Query sets	Num. KMs		Num. QMs		Num. CJNs		CJNs / QMs	
	Max	Avg	Max	Avg	Max	Avg	Max	Avg
IMDb	47	15.38	702	80.50	656	97.28	0.93	1.21
IMDb-DI	64	16.20	702	85.34	656	103.88	0.93	1.22
MONDIAL	8	3.12	9	2.10	35	9.40	3.89	4.48
MONDIAL-DI	8	3.40	9	2.42	44	11.04	4.89	4.56
Yelp	21	11.82	124	32.39	301	74.07	2.43	2.29

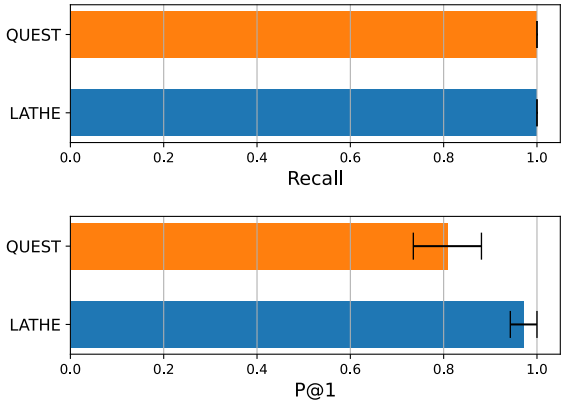

FIGURE 8. Comparison of Lathe with the QUEST system.

C. COMPARISON WITH OTHER R-KwS SYSTEMS

In this experiment, we first compare Lathe with QUEST [12], the current state of art R-KwS system with support to schema references and then we also compare Lathe with several other R-KwS systems. Here, we used the default Lathe setup, that is, $8/1/9$. We compare our results to those published by the authors, which refer to the MONDIAL dataset, because we were unable to run QUEST due to the lack of code and enough details for implementing it. Figure 8 depicts the results for the 35 queries supported by QUEST[8] out of the 50 queries provided in the original query set. The graphs show the recall and P@1 values for the raking produced by each system considering the golden standard supplied by Coffman & Weaver [3].

Both systems achieved perfect recall; that is, all the correct solutions for the given keyword queries were retrieved. Concerning P@1, Lathe obtained better results than QUEST, with an average of 0.97 with a standard error of 0.03, which

[8]Specifically, queries 01-20, 26-35 and 46-50.

indicates that, in most cases, the correct solution was the one corresponding to the CJN ranked as the first by Lathe.

Next, we compare the results obtained for Lathe with those published in the comprehensive evaluation published by Coffman & Weaver [3] for the systems BANKS [14], DISCOVER [4], DISCOVER-II [6], BANKS-II [15], DPBF [38], BLINKS [16] and STAR [39]. Because this comparison uses all 50 keyword queries from the MONDIAL dataset, we did not include QUEST in the comparison. Figure 9 shows the recall and P@1 values for the raking produced by each system when the golden standard provided by Coffman & Weaver [3] is taken into account.

Overall, Lathe achieved the best results in Recall and P@1 value. That the only systems that achieved similar recall, DPBF and BLINKS, are based on data graph, thus, require a materialization of the database. The difference between recall values of Lathe, DISCOVER, and DISCOVER-II is mainly due to not supporting schema references. Regarding the P@1, Lathe obtained a value of 0.96 with a standard error of 0.03, which is significantly higher than the results for other systems. This difference in P@1 value, especially compared with DISCOVER and DISCOVER-II, is due to the novel ranking of QMs as well as an improved ranking of CJNs.

D. EVALUATION OF QUERY MATCHES RANKING

In this experiment, we evaluate the quality of QMs ranking according to the metrics MRR and $R@K$. As shown by the results in Section VIII-B, there can be many QMs depending on the query. As a result, we want to verify how effective the QMRank algorithm is at selecting the most likely correct QM from among those generated in this experiment. Figure 10 shows the results obtained with $R@K$ up to the tenth ranking position and the MRR metric.

For all query sets, in most cases, the correct QM is at least in the eighth ranking position. In MONDIAL and MONDIAL-DI, the relevant QM is at least in the third position for all queries. Yelp obtained an $R@8$ of 1 and $R@3$ of 0.93, which indicates that the system returns the relevant QM by the eighth position, and in most cases, up to the third position. There is one query for the IMDb dataset whose relevant QM is not minimal. As QMs must be minimal by Definition 5, Lathe does not support this query. Consequently, the query sets for the IMDb dataset can obtain an $R@K$ value of 0.98 at most. IMDb and IMDb-DI achieved this value at position 5.

Regarding MRR, Lathe obtained 0.75 for both IMDb and IMDb-DI, 0.83 for Yelp, and 0.96 and 0.95 for MONDIAL and MONDIAL-DI, respectively. This result indicates that the relevant QM is often in the top positions of the ranking. Notice that the QM ranking indirectly impacts the generation and ranking of CJNs. In practice, a high $R@K$ value with a low K allows us to generate fewer CJNs without compromising the quality of the CJN ranking. Based on the obtained results, we set the parameter N_{QM} to 8, which indicates that Lathe will only generate CJNs for the top-8 query matches.

E. EVALUATION OF THE CANDIDATE JOINING NETWORK RANKING

In this experiment, we evaluate the quality of our approach for CJN generation and ranking. We used the metrics MRR and $R@K$ for K up to the tenth rank position. We tested several different setups but to save space we report here only those with representative distinct results. Specifically, we report the results of four setups without the eager evaluation, that is, $8/1/0$, $8/2/0$, $8/8/0$ and $8/9/0$ and two setups with the eager evaluation, that is $8/1/9$ and $8/2/9$.

Figure 11 shows the results for the IMDb and IMDb-DI query sets. As it can be seen, regardless of the configuration, our method was able to place the relevant CJNs in the top positions in the ranking, and the result is very similar for both IMDb and IMDb-DI query sets. This shows that in these datasets, our method was able to disambiguate the queries properly, even without the addition of schema references. It is worth noting that the values of $R@1$ in both query sets show that the configurations with the eager evaluation achieved better results because they place the relevant CJNs in the first ranking position more frequently. The $R@K$ metric also shows that the quality of the ranking decreases as the number of CJNs per QM increases, especially for K in the range $2 \leq K \leq 6$.

Figure 12 shows the results for MONDIAL and MONDIAL-DI. In these query sets, the configurations with the eager evaluation achieved significantly better results. The configurations $8/1/0$ and $8/2/0$ could not generate the relevant CJN for around 20% of the queries due to a low number of CJNs per QM, therefore, their results were capped at an MMR and $R@K$ value of 0.8, approximately. The configurations $8/8/0$ and $8/9/0$ were able to generate the relevant CJN for most of the cases, although the large number of CJNs per QM negatively affected the ranking of CJNs. Finally, the configurations $8/1/9$ and $8/2/9$ produced the best results because the pruning enables us to generate the relevant CJN with a low number of CJNs per QM while also placing the relevant CJN in higher rank positions. Notice that the disambiguation of queries in the MONDIAL-DI query set allowed configurations $8/8/0$ and $8/9/0$ to have better results, especially for the $R@K$ metric for K above 7. The eager evaluation configurations were able to disambiguate the queries without relying on the addition of schema references, therefore, their results were consistent across the MONDIAL and MONDIAL-DI query sets.

Figure 13 shows the results for the Yelp query set. Overall, the eager CJN evaluation did not affect the results for this query set, probably because the database schema graph was simple and the ways of connecting the query matches were straightforward. Configurations $8/1/0$ and $8/1/9$ achieved the best results, obtaining a MRR of 0.85 and $R@2$ of 0.92 for the CJN generation. This indicates that the relevant CJNs are often found up to the second ranking position, with exception of two queries, whose relevant CJN were found in positions 5 and 7, respectively. The other configurations

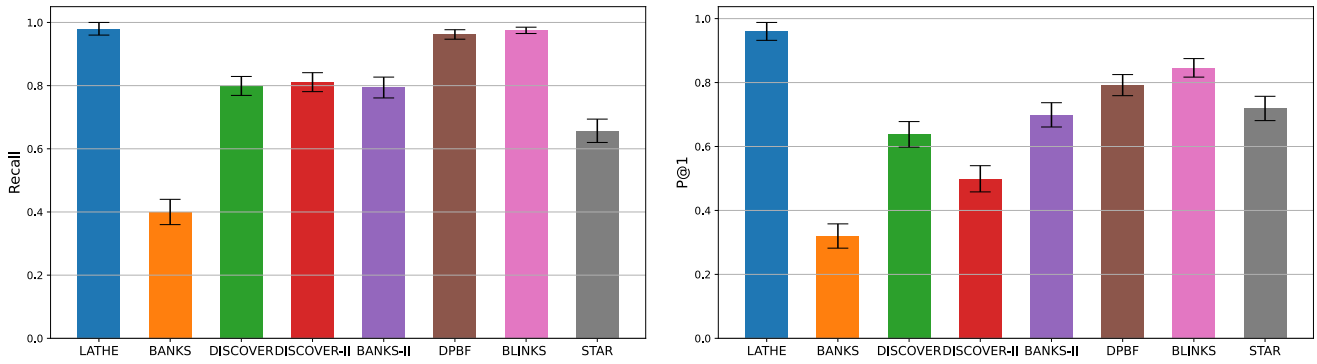

FIGURE 9. Comparison with other approaches using Recall and P@1 metrics.

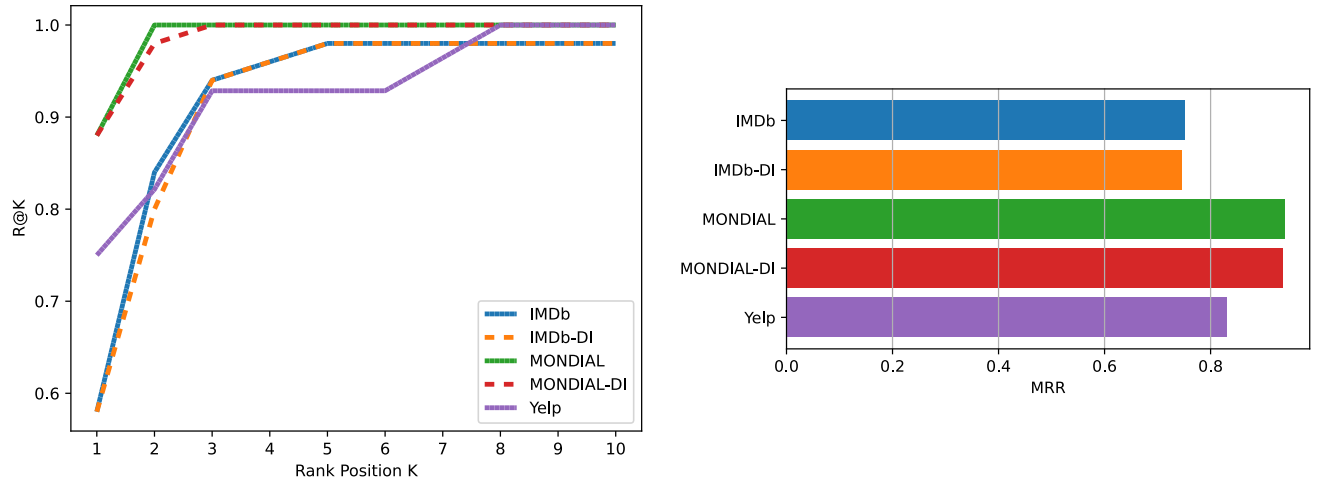

FIGURE 10. Evaluation of Query Matches.

obtained slightly worse results, with an MRR of 0.84 and an $R@2$ of 0.89, approximately.

Regardless of the datasets and configurations, our method achieved an MRR value above 0.7, which indicates that on average, the relevant CJN is found between the first and the second rank positions. In the IMDb dataset, the decrease of $R@K$ values according to the number of CJNs taken per QM is also reflected on the MRR metric. However, in the MONDIAL dataset, the improvement of the $R@K$ values due to the disambiguation of queries is not reflected on the MRR value, as this improvement only happens in low ranking positions ($K \leq 8$).

The eager CJN evaluation inherently affects the performance of the CJN generation process. Therefore it is important to look at the trade-off between the effectiveness and the efficiency in each configuration. We examine this trade-off in the next section.

F. PERFORMANCE EVALUATION

In this experiment, we aim at evaluating the time spent for obtaining the CJN given a keyword query, and analyze the trade-offs between efficiency and efficacy of the different configurations use in Lathe.

Lathe obtained better execution times for the IMDb dataset in all configurations. Also, the disambiguate variants of query sets yield slower execution times in comparison with the original counterparts.

Figure 14 summarizes the average execution time for each phase of the process: Keyword Matching, Query Matching and the Candidate Joining Network Generation. In this first experiment, we used the configuration $8/1/0$. Lathe obtained better total execution times for the IMDb dataset, followed by the Yelp dataset. In addition, the disambiguate variants of query sets yield slower execution times in comparison with the original counterparts. Also, it is worth noting that the execution times for each query set are related to the number of KMs, QMs, and CJNs in the query sets shown in Table 4.

Regarding keyword matching, the Yelp dataset yielded the worst execution times, with 167ms, probably because of its higher number of attributes and tuples. Although the MONDIAL dataset has fewer tuples than IMDb, its higher number of schema elements (28 relations and 48 attributes) results in a higher execution time than IMDb.

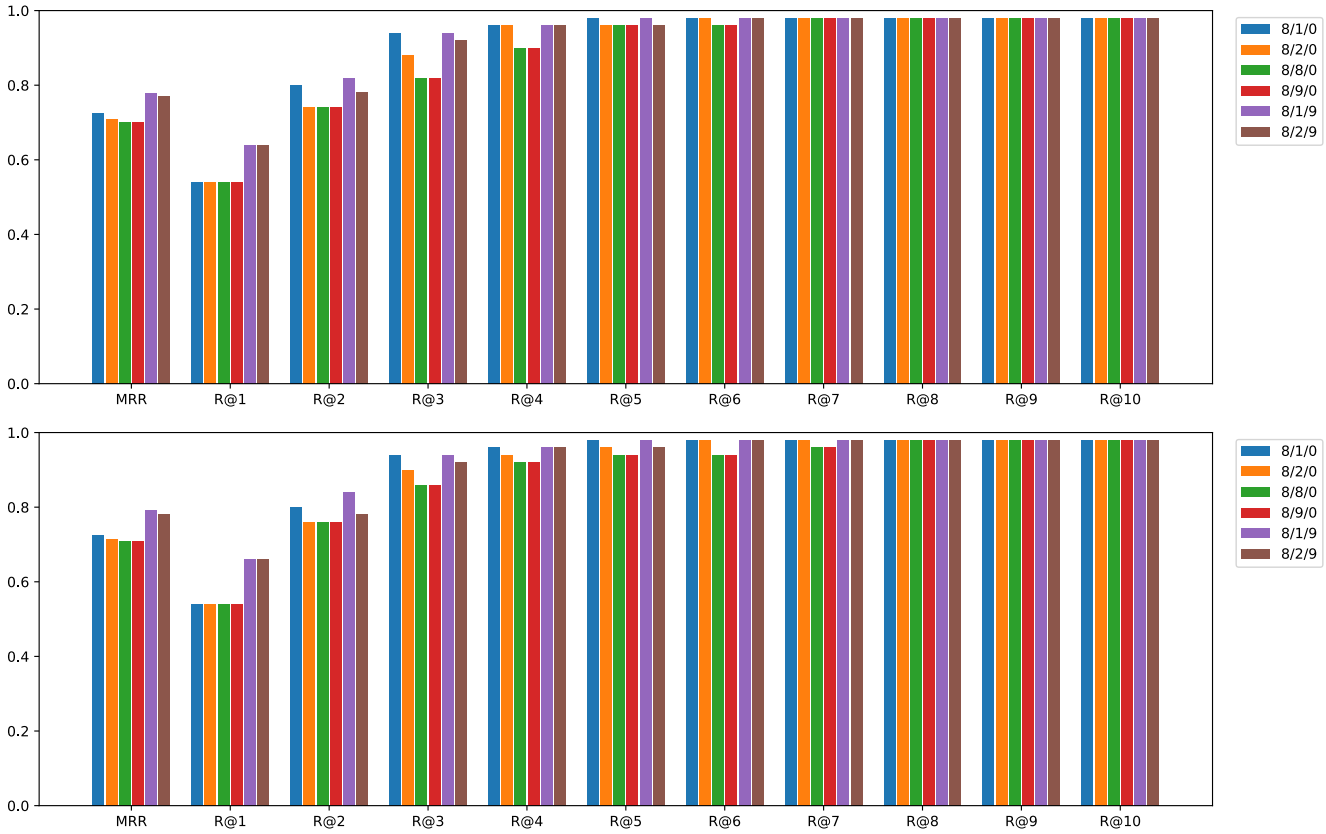

FIGURE 11. Ranking of Candidate Joining Networks - IMDb (top) and IMDb-DI (bottom).

Due to the combinatorial nature of QM generation, the execution times for the Query Matching phase are directly related to the number of QMs. While the execution times for the IMDb and IMDb-DI query sets that produced a high number of QMs are 247 and 256 milliseconds, respectively, the results for the MONDIAL and MONDIAL-DI are around 190 and 202 microseconds. The Yelp dataset achieved 121 miliseconds.

Concerning the CJN phase, the execution times for MONDIAL are significantly higher in comparison with the execution times for IMDb and Yelp, despite the lower number of CJNs for the MONDIAL. Because the CJN generation algorithm is based on a Breadth-First Search, the greater the number of vertices and edges in the schema graph of the MONDIAL dataset, the greater the number of iterations and, consequently, the slower the execution times. This behavior persists throughout different configurations, an issue we further analyze below.

G. QUALITY VERSUS PERFORMANCE

Figure 15 presents an evaluation of the CJN generation performance, comparing the same configurations used in the experiment of Section VIII-E. We present the results for the IMDB, MONDIAL and Yelp datasets in different scales because they differ by order of magnitude. Overall, execution times increase as the number of CJNs taken per QM increases. This pattern is more pronounced in the MONDIAL dataset. Also, the eager CJN evaluation incurs an unavoidable increase in the CJN generation time as the system has to probe the CJNs running queries into the database.

As the configurations have an impact on both the quality of the CJN ranking and the performance, it is important to examine the trade-off between effectiveness and efficiency. Configuration $8/1/0$ and $8/2/0$ achieved the best execution times due to the low number of CJNs per QM and not relying on database accesses. However, these configurations did not achieve the highest values of MRR and R@K for the IMDb and MONDIAL datasets. Therefore, they are recommended if one must prioritize efficiency.

Configuration $8/1/9$ obtained better results than configurations $8/8/0$, $8/9/0$ for the IMDb and MONDIAL datasets and better than $8/2/9$ for all datasets. Although this configuration is slower than $8/1/0$ and $8/2/0$, the significantly better results of MRR and $R@K$ values for MONDIAL and IMDb datasets make the $8/1/9$ configuration an overall recommended option, especially if one must prioritize effectiveness.

We do not recommend the configurations $8/2/0$, $8/8/0$, $8/9/0$ and $8/2/9$ because their MRR and $R@K$ values do not justify the increase in execution times. Although $8/2/9$ obtained the best MRR and $R@K$ values for the MONDIAL dataset, it is 37%-80% slower than $8/1/9$. Configurations $8/8/0$ and $8/9/0$ achieved a slight increase in the

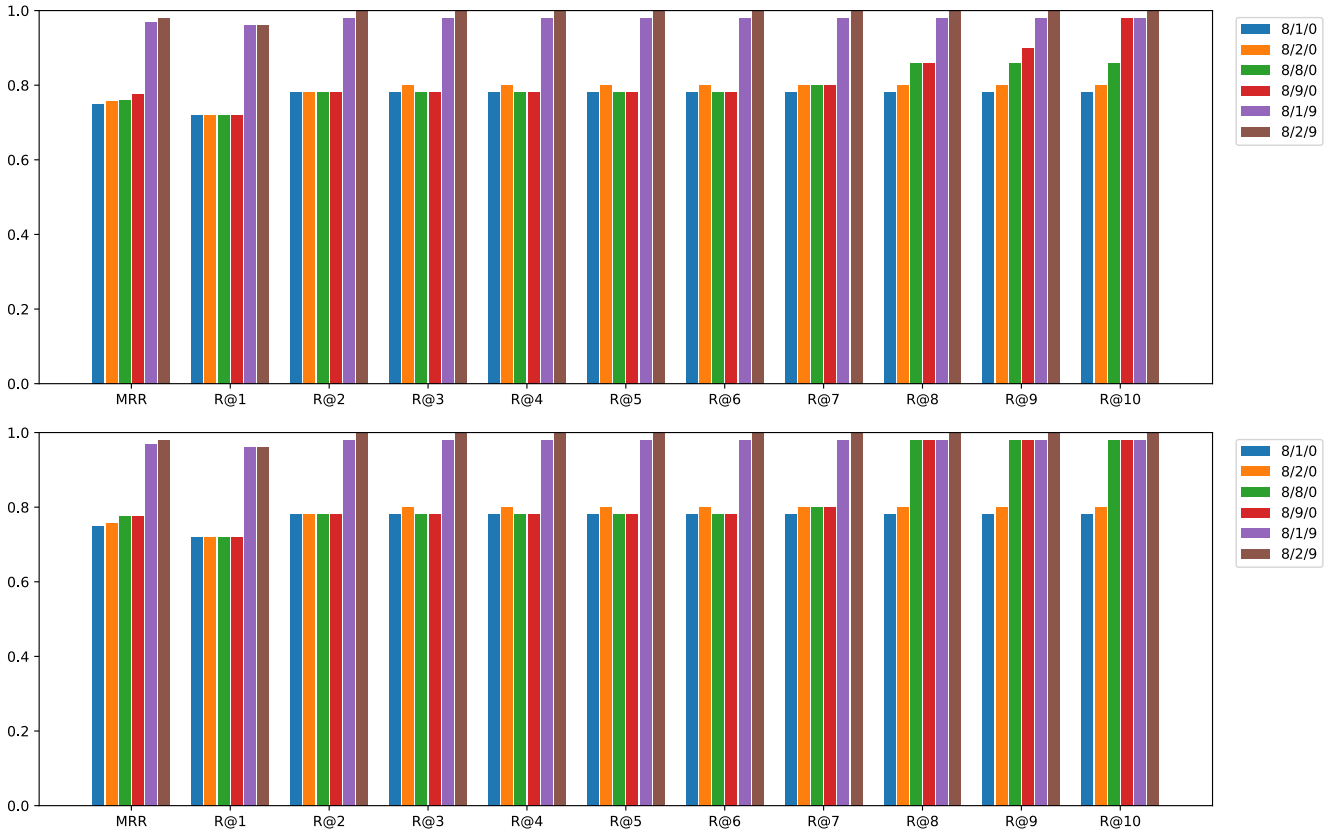

FIGURE 12. Ranking of Candidate Joining Networks - MONDIAL (top) and MONDIAL-DI (bottom).

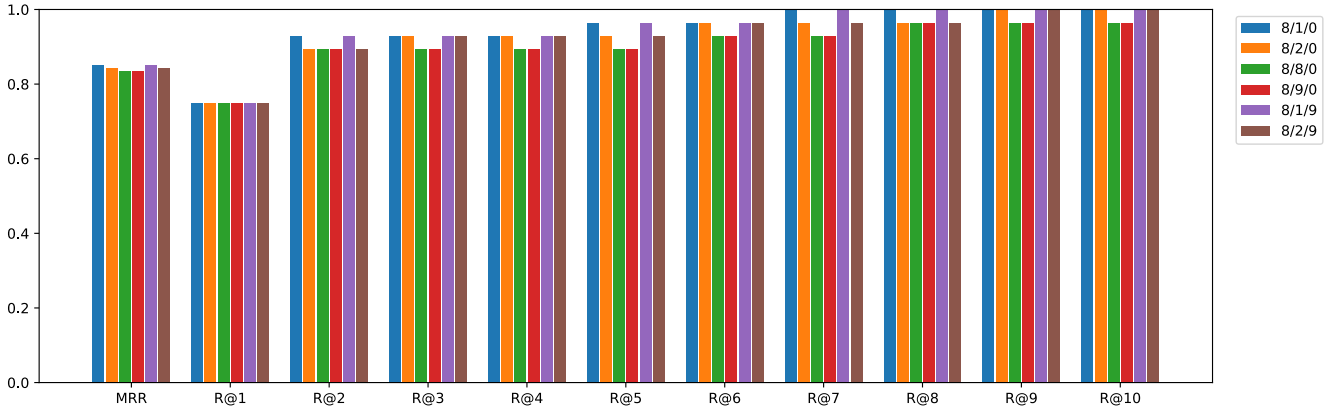

FIGURE 13. Ranking of Candidate Joining Networks - Yelp.

$R@K$ metric for the MONDIAL dataset, for $K \leq 8$, however, they obtained lower values of MRR and $R@K$ values for the IMDb and Yelp datasets.

It is interesting noting that although the configurations with eager CJN evaluation spend time to probe CJNs, sending queries to the DBMS. However, as they generate a smaller set of CJNs, the overall performance is not hindered in comparison with the configurations without it.

IX. CONCLUSION

In this paper, we have proposed Lathe, a new relational keyword search (R-KwS) system for generating a suitable SQL query from a given keyword query. Lathe is the first to address the problem of generating and ranking Candidate Joining Networks (CJNs) based on queries with keywords that can refer to either instance values or database schema elements, such as relations and attributes.

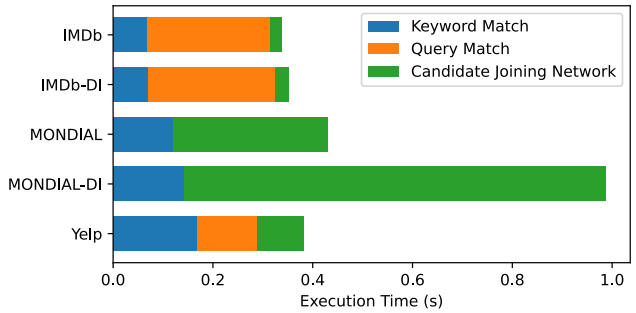

FIGURE 14. Average Execution Times for each phase of Lathe.

In addition, Lathe improves the quality of the CJN generated by introducing two major innovations: a ranking for selecting better Query Matches (QMs) in advance, yielding the generation of fewer but better CJNs, and an eager evaluation strategy for pruning void useless CJNs.

We present a comprehensive set of experiments performed with query sets and datasets previously used in experiments with previous state-of-the-art R-KwS systems and methods. Our experiments indicate that Lathe can handle a wider variety of keyword queries while remaining highly effective, even for large databases with intricate schemas.

Also, a full implementation of Lathe is publicly available at https://github.com/pr3martins/Lathe as a Python library for Keyword Search over Relational Databases called PyLatheDB [40]. This library is ready for developers to easily run Lathe or incorporate its features, such as keyword matching, into their own applications.

Our experience in the development of Lathe raised several ideas for future work. First, one important issue in our method is being able to correctly match keywords from the input query to the corresponding database elements. To improve this issue, we plan to investigate new alternative similarity functions. We are particularly interested in using word-embedding-based functions, such as the well-known Word Mover's Distance (WMD) [41]. We also consider investigating methods based on Neural Language Models (NLMs), particularly on transformers and attention-based models [42], [43], [44], which proved to be promising for several text-based Information Retrieval problems. For example, we believe that an interesting approach to the QM ranking problem is to interpret it as a variant of the *Table Retrieval task* [45], [46], where given a keyword query and a table corpus, this task consists of returning a ranked list of the tables that are relevant to the query.

Second, data exploration techniques have recently gained popularity because they allow for the extraction of knowledge from data even when the user is unsure of what to look for [47]. Keyword-based queries, we believe, can be used as an interesting tool for data exploration because they allow one to retrieve interesting portions of a database without knowing the details of the schema and its semantics.

Third, we believe that pruning strategies can improve the QM generation by reducing the search space of keyword matches. For this, we plan to exploit the relationship of the QM generation and the discovery of matching dependencies [48].

Fourth, although we have focused on relational databases in this paper, the ideas we discussed here can be extended to other types of databases as well. Currently, we are extending these ideas to address the so-called *document stores*, such as the very popular MongoDB[9] engine. Our preliminary findings [36] suggest that because queries of this type are frequently more complex than queries of relational databases, allowing the simplicity of keyword queries may have even more advantages in this context.

Finally, we anticipate that keyword queries will be useful as a tool for allowing the seamless integration of data from heterogeneous sources, as is the case in the so-called *polystore systems* and *data lakes*, which are becoming increasingly popular in recent years. There exist already research proposals in this direction [49], we believe that the schema graph approach we adopt in our work can be helpful to achieve this goal.

APPENDIX A ACRONYMS

R-KwS	Relational Keyword Search.
CJN	Candidate Joining Network.
QM	Query Match.
DB	Database.
SQL	Structured Query Language.
IR	Information Retrieval.
DBMS	Database Management System.
JNT	Joining Network of Tuples.
IMDb	Internet Movie Database.
KM	Keyword Match.
VKM	Value-Keyword Match.
SKM	Schema-Keyword Match.
PKFK	Primary Key/Foreign Key.
WUP	Wu-Palmer Measure.
LCS	Least Common Subsumer.
TF-IDF	Term Frequency – Inverse Document Frequency.
RIC	Relational Integrity Constraint.
JNKM	Joining Network of Keyword Matches.
MJNKM	Minimal Joining Networks of Keyword Matches.
R@K	Recall at K.
P@1	Precision at 1.
MRR	Mean Reciprocal Rank.
WMD	Word Mover's Distance.
NLM	Neural Language Model.
ECLAT	Equivalence Class Clustering and bottom-up Lattice Traversal.

[9]https://www.mongodb.com/

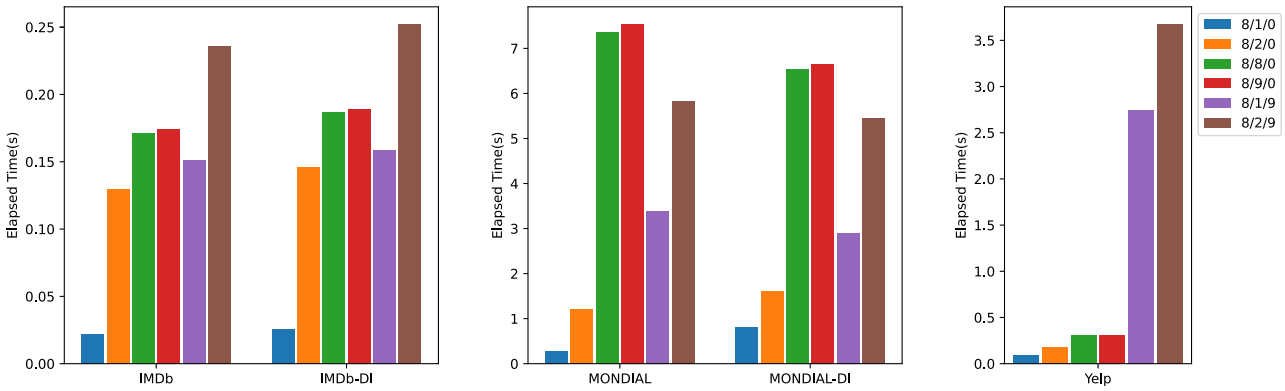

FIGURE 15. Performance Evaluation of the CJN Generating phase.

APPENDIX B VKMGen ALGORITHM

As shown in Algorithm 1, Lathe retrieves tuples from the database in which the keywords occur and uses them to generate value-keyword matches. Initially, the VKMGen Algorithm takes the occurrences of each keyword from the *Value Index* and form partial value-keyword matches, which are not guaranteed to be disjoint sets yet (Lines 3-8). The pool of VKMs is represented by the *Hash Table P*, whose keys are KMs and values are sets of tuple IDs.

Algorithm 1 VKMGen(Q)

Input: A keyword query Q=$\{k_1, k_2, \ldots, k_m\}$
Output: The set of value-keyword matches VK

1 **let** I_V the Value Index
2 **let** Pbe a Hash Table.
3 **for** keyword $k_i \in Q$ **do**
4 **if** $k_i \in I_V$ **then**
5 **for** relation $R_j \in I_V[k_i]$ **do**
6 **for** attribute $A_k \in I_V[k_i][R_j]$ **do**
7 **let** KM be the partial keyword match $R_j^V[A_k^{\{k_i\}}]$
8 $P[KM] \leftarrow I_V[k_i][R_j][A_k]$
9 $P \leftarrow VKMIter(P)$
10 **for** value-keyword match $KM_u \in P$ **do**
11 $VK \leftarrow VK \cup \{KM_u\}$
12 **return** VK

Next, Lathe ensures that VKMs are disjoint sets through the Algorithm 2, VKMInter, which is based on the ECLAT[10] algorithm [50] for finding frequent itemsets. VKMInter looks for non-empty intersections of the partial value-keyword matches recursively until all of them are disjoint sets, and thus, proper VKMs. These intersections are calculated as follows:

$$KM_1 \cap KM_2 = \begin{cases} \emptyset & , \text{if } R_a \neq R_b \\ R_{ab}^V[A_{ab,1}^{K_{ab,1}}, \ldots, A_{ab,m}^{K_{ab,m}}] & , \text{if } R_a = R_b \end{cases}$$

[10]Equivalence Class Clustering and bottom-up Lattice Traversal.

where $KM_x = R_x^V[A_{x,1}^{K_{x,1}}, \ldots, A_{x,m}^{K_{x,m}}]$ for $x \in \{a, b\}$, and $K_{ab,i} = K_{a,i} \cup K_{b,i}$.

Algorithm 2 VKMInter(P)

Input: A Hash Table P whose keys are partial value-keyword matches and values are tuples.
Output: A Hash Table P whose keys are proper value-keyword matches and values are tuples.

1 **let** P_{next} be a Hash Table.
2 **let** R be a Hash Table.
3 **for** value-keyword match $KM_u \in P$ **do**
4 $R[KM_u] \leftarrow \emptyset$
5 **for** pair of keyword matches $\{KM_a, KM_b\} \in \binom{P}{2}$ **do**
6 $KM_{ab} \leftarrow KM_a \cap KM_b$
7 $T_{ab} \leftarrow P[KM_a] \cap P[KM_b]$
8 **if** $T_{ab} \neq \emptyset$ **and** KM_{ab} is valid **then**
9 $P_{next}[KM_{ab}] \leftarrow T_{ab}$
10 $R[KM_a] \leftarrow R[KM_a] \cup T_{ab}$
11 $R[KM_b] \leftarrow R[KM_b] \cup T_{ab}$
12 **for** value-keyword match $KM_u \in R$ **do**
13 $P[KM_u] \leftarrow P[KM_u] - R[KM_u]$
14 **if** $P[KM_u] = \emptyset$ **then**
15 **remove** KM_u from P
16 $P.remove(KM_u)$
17 $P_{next} \leftarrow VKMInter(P_{next})$
18 **update** $PwithP_{next}$
19 **return** P

VKMInter uses three hash tables: P, P_{next} and R. The pool P contains the partial VKMs of the current iteration. The pool P_{next} contains the partial VKMs for the next iteration. The pool R stores the tuple IDs to be removed from the VKMs of P at the end of the current iteration, turning the partial VKMs into proper value-keyword matches.

VKMInter first defines the hash tables P_{next} and R, then initializes R with empty sets (Lines 1-4). Next, the algorithm iterates over all pairs $\{KM_a, KM_b\}$ of VKMs in P and tries to

create a new keyword match KM_{ab}, which is the intersection of KM_a e KM_b (Lines 5-11). If KM_{ab} is valid, that is, if KM_a e KM_b are VKMs over the same database relation, and the tuples T_{ab} within KM_{ab} are not empty, then we add KM_{ab} to the next iteration pool P_{next} and add the tuples T_{ab} to R for removal after the iteration (Lines 8-11). After all the possible intersections are processed, VKMInter iterates over R and removes the tuples for each VKM of the pool P, making them proper disjoint keyword matches (Lines 12-16). Lastly, VKMInter recursively process the pool P_{next} for the next iteration, then it updates and returns the current pool P (Lines 18-19).

After the execution of VKMInter, in Line 9 of VKMGen, we obtained the value-keyword matches and their tuples. As the sets of tuples are only required for the generation of VKMs, VKMGen generates and outputs the set of value-keyword matches, ignoring the tuples from P (Lines 10-11). From now on, Lathe does not need to manipulate the database tuples or their IDs.

APPENDIX C SKMGen ALGORITHM

The generation of schema-keyword matches uses a structure we call the *Schema Index*, which is created in a preprocessing phase, alongside with the Value Index. This index stores information about the database schema and statistics about attributes, which are used for the ranking of QMs, which will be explained in Chapter VI. The stored information follows the structure below:

$$I_S = \{relation : \{attribute : \{(norm, maxfrequency)\}\}\}$$

The generation of SKMs is carried out by Algorithm 3, SKMGen. First, the algorithm iterates over the relations and attributes from the Schema Index. Then, SKMGen calculates the similarity between each keyword and schema element. It only considers the pairs whose similarity is above a threshold ε (Line 8), which is used to generate SKMs (Line 3).

Algorithm 3 SKMGen(Q)

Input: A keyword query Q=$\{k_1, k_2, \ldots, k_m\}$, the Schema Index I_S
Output: The set of schema-keyword matches SK

```
1  SK ← {}
2  for keyword k_i ∈ Q do
3      for relation R_j ∈ I_S do
4          if sim(k_i, R_j) ≥ ε then
5              let KM be the schema-keyword match
                 R_j^S[self^{k_i}]
6              SK ← SK ∪ {KM})
7          for attribute A_l ∈ I_S[R_j] do
8              if sim(k_i, A_l) ≥ ε then
9                  let KM be the schema-keyword match
                     R_j^S[A_l^{k_i}]
10                 SK ← SK ∪ {KM}
11 return SK
```

APPENDIX D QMGen ALGORITHM

The generation of query matches is carried out by Algorithm 4, QMGen, which preserves the ideas proposed in MatCNGen [2], adapt them to keyword matches instead of tuple-sets. Let VK and SK be respectively sets of value-keyword matches, and schema-keyword matches previously generated. The algorithm looks for combinations of keyword matches in P=$VK \cup SK$ that form minimal covers for the query Q. At a first glance, this statement may suggest that we need to generate the whole power set of P to obtain the complete set of QMs. However, it can be shown that any minimal cover of a set of n elements has at most n subsets [51]. Therefore, no match for a query Q can be formed by more than $|Q|$ keyword matches.

It is easy to see that Algorithm 4 has a time complexity of $\sum_{i=1}^{|Q|} \binom{|P|}{i}$. This equation gives us an upper bound on the number of query matches that must be generated for a query. It shows that the running time depends on two important factors: the size of the query and on the size of the sets of keyword matches P. Regarding these two factors, the first one, the size of a query is usually small, e.g., less than two on average, and queries with more than four keywords are rare [2]. In such cases, this summation turns to be a low-degree polynomial. The second factor, $|P|$, is also dependent on the query size, but the main issue to observe is how termsets are distributed among relations. This factor is harder to predict, but usually very few subsets of query terms are frequent in many relations. In fact, larger subsets are increasingly less frequent. Thus, in practice, just a few query matches need to be generated.

Also, as the QM ranking presented in Section VI-B penalizes QMs with a large number of KMs, we can define a maximum QM size $t \leq |Q|$ to prune QMs which are less likely to be relevant. For this reason, QMGen iterates over all the subsets of P whose size is less than or equal to a maximum QM size t, which in our experiments we set a value of $t = 3$ (Lines 3-4). Next, QMGen checks whether the combination M of keyword matches form a minimal cover for the query. The evaluation of minimal cover is carried out by Algorithm 5.

The algorithm MinimalCover iterates through the KMs from the combination M, generating a set C_M which comprise all keywords covered by M (Lines 1-5). Next, the algorithm checks whether M is total, that is, whether C_M=Q. Notice that since KMs can only associate an attribute or relation in the database schema to keyword from the query Q, that is $C_M \subseteq Q$, then we can imply that C_M=Q if, and only if, $|C_M|$=$|Q|$ (Line 6). Next, MinimalCover checks whether M is minimal, that is, if we remove any keyword match from M it will no longer be total. For this reason, MinimalCover iterates again through the KMs and, for each one, it generates a set C_{KM} which comprise all keywords covered by KM. Then, the algorithm check whether the set difference of $C_M \setminus C_{KM}$

Algorithm 4 QMGen(Q, VK, SK)

Input: A keyword query Q=$\{k_1, k_2, \ldots, k_m\}$ The set of value-keyword matches VK The set of schema-keyword matches SK The maximum QM size t
Output: The set of query matches QM

```
1  P = VK ∪ SK
2  QM ← ∅
3  for i ∈ {1, ..., min(|Q|, t)} do
4      for combination of keyword matches M ∈ (P choose i) do
5          if MinimalCover(M, Q) then
6              M ← MergeKeywordMatches(M)
7              QM ← QM ∪ {M}
8  return QM
```

is still equal to Q, which can be achieved by comparing $|C_M \backslash C_{KM}|$=$|Q|$.

Algorithm 5 MinimalCover(Q, M)

Input: A keyword query Q=$\{k_1, k_2, \ldots, k_m\}$ The set of keyword matches M
Output: If the set of keywords from M forms a minimal cover over Q

```
1  C_M ← ∅
2  for keyword match KM ∈ M do
3      let KM be R^X[A_1^{K_1}, ..., A_m^{K_m}], where X ∈ {S, V}
4      for i ∈ {1, ..., m} do
5          C_M ← C_M ∪ K_i
6  if |C_M| ≠ |Q| then
7      return False
8  for keyword match KM ∈ M do
9      let KM be R^X[A_1^{K_1}, ..., A_m^{K_m}], where X ∈ {S, V}
10     C_KM = ∅
11     for i ∈ {1, ..., m} do
12         C_KM ← C_KM ∪ K_i
13     if |C_M \ C_KM| = Q then
14         return False
15 return True
```

If M forms a minimal cover for Q, then M is considered a query match. However, M may have some keyword matches which can be merged, especially SKMs. The merging of KMs from M is carried out by Algorithm 6. Notice that we cannot merge two VKMs since they are disjoint sets, however we can merge a schema-keyword match with both a SKM or a VKM. The algorithm MergeKeywordMatches uses the two *hash tables* P_{VK} and P_{SK} to store, respectively, the VKMs and SKMs based on the relation they are built upon (Lines 1-12). Next, the algorithm iterates through the relations present in P_{SK} and tries to merge all possible KMs from that relation, resulting in a keyword match KM_{merged}. KM_{merged} starts as a keyword-free match but it is merged with all existent SKMs (Lines 14-17), then it is merged an arbitrary value-keyword match VKM, if existent (Lines 18-21). Lastly, KM_{merged} and all values-keyword matches except VKM are added to the query match M', which is returned at the end of MergeKeywordMatches (Lines 22-23).

After merging all the possible elements from the query match M, QMGen adds M to the set of query matches QM, which is returned at the end of the algorithm.

Algorithm 6 MergeKeywordMatches(Q, M)

Input: The set of keyword matches M
Output: The set of keyword matches M'

```
1  let P_VK be a Hash Table.
2  let P_SK be a Hash Table.
3  for KM ∈ M do
4      let KM be R^X[A_1^{K_1}, ..., A_m^{K_m}], where X ∈ {S, V}
5      P_VK[R] ← ∅
6      P_SK[R] ← ∅
7  for KM ∈ M do
8      let KM be R^X[A_1^{K_1}, ..., A_m^{K_m}], where X ∈ {S, V}
9      if X = S then
10         P_SK[R] ← P_SK[R] ∪ {KM}
11     else
12         P_VK[R] ← P_VK[R] ∪ {KM}
13 M' ← ∅
14 for R ∈ P_SK do
15     let KM_merged be a keyword-free match from R
16     for SKM ∈ P_SK[R] do
17         KM_merged ← KM_merged ∩ SKM
18     if P_VK[R] ≠ ∅ then
19         let VKM be an element from P_VK[R]
20         KM_merged ← KM_merged ∩ VKM
21         P_VK[R] ← P_VK[R] − {VKM}
22     M' ← M' ∪ {KM_merged} ∪ P_VK[R]
23 return M'
```

APPENDIX E QMRank ALGORITHM

The ranking of Query Matches is carried out by Algorithm 7, QMRank. Notice, that, intuitively, the process of ranking QMs advances part of the relevance assessment of the CJNs, which was first proposed in CNRank [10]. This yields to an effective ranking of QMs and a simpler ranking of CJNs. QMRank uses a value score and a schema score, which are respectively related to the VKMs and SKMs that compose the QM.

The algorithm first iterates over each query match, assigning 1 to both *value_score* and *schema_score*. Next, QMRank goes through each keyword match from the QM. In the case of a KM matching the values of an attribute, the algorithm updates the *value_score* based on the cosine similarity using

Algorithm 7 QMRank(QM)

Input: A set of query matches QM
Output: The set of ranked query matches RQM

$RQM \leftarrow [\,]$
for $M \in QM$ **do**
  $value\_score \leftarrow 1$, $schema\_score \leftarrow 1$
  for $KM \in M$ **do**
    let KM be $R^S[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]^V[A_1^{K_1^V}, \ldots, A_m^{K_m^V}]$
    for $i \in \{1, \ldots, m\}$ **do**
      if $|K_i^V| \geq 1$ **then**
        $weight\_sum \leftarrow 0$
        $norm_{A_i} \leftarrow I_S[R][A_i]$
        for $word \in K_i^V$ **do**
          $tf \leftarrow |I_V[word][R][A_i]|$
          $weight\_sum \leftarrow weight\_sum + tf \times iaf(word)$
        $value\_score \leftarrow value\_score \times weight\_sum / norm_{A_i}$
      if $|K_i^S| \geq 1$ **then**
        $weight\_sum \leftarrow 0$
        for $word \in K_i^S$ **do**
          if $A_j = self$ **then**
            $schema\_element \leftarrow R$
          else
            $schema\_element \leftarrow A_i$)
          $weight\_sum \leftarrow weight\_sum + sim(schema\_element, word)$
        $schema\_score \leftarrow schema\_score \times weight\_sum / |K_i^S|$
  $final\_score \leftarrow value\_score \times schema\_score$
  RQM.append($\langle final\_score, M \rangle$)
Sort RQM in descending order
return RQM

TF-IDF weights. QMRank retrieves the term frequency and inverted attribute frequency from the Value Index, and the norm of an attribute from the Schema Index, which are all calculated in the preprocessing phase (see Section IV-A). In the case of a KM matching the name of a schema element, the algorithm updates the *schema_score* the average similarity of the keywords with the schema elements based on the similarity functions presented in Section V. Once the algorithm aggregates the scores of KMs to generate the score of QMs, the final step is to sort them in descending order.

APPENDIX F SOUND THEOREM

*Theorem 1: Let $G_S = \langle \mathcal{R}, E_G \rangle$ be a schema graph. Let $J = \langle \mathcal{V}, E_J \rangle$ be a joining network of keyword matches. We say that J is **sound**, that is, it does not have more than one occurrences of the same tuple for every instance of the database if, and only if, the following condition holds $\forall KM_a \in \mathcal{V}, \forall \langle R_a, R_b \rangle \in E_G$:*

$$RIC(R_a, R_b) \geq |\{KM_c | \langle KM_a, KM_c \rangle \in E_J \wedge R_c = R_b\}|$$

where $RIC(R_a, R_b)$ indicates the number of Referential Integrity Constraints from a relation R_a to a relation R_b.

Proof: Let R_a and R_b be database relations so that there exists n *Referential Integrity Constraint* (RICs) from R_a to R_b. Intuitively, a tuple from R_a may refer to at most n tuples from R_b. Consider a joining network of keyword matches J wherein a keyword match over R_a is adjacent to m keyword matches over R_b, that is $J = \langle \mathcal{V}, E \rangle$, where $\mathcal{V} = \{KM_1, \ldots, KM_{m+1}\}$, $E = \{\langle KM_1, KM_i \rangle | 2 \leq i \leq m\}$, and $R_1 = Ra \wedge R_i = R_b, 2 \leq i \leq m$. We can translate J into a relational algebra expression wherein the edges are join operations using RICs and keyword matches are selection operations over relations. For didactic purposes, we assume, without loss of generality, that all the KMs of J are keyword-free matches. Let k_j be a key attribute from R_i and $f_{i,j}$ be the attribute from R_i that references k_j. The SQL translation of J can be represented by T_{m+1}, which expands a join operation in each iteration.

$$\begin{aligned} 4T_1 &= R_1 \\ T_2 &= T_1 \bowtie_{f_{1,2}=k_2} R_2 \\ T_3 &= T_2 \bowtie_{f_{1,3}=k_3} R_3 \\ T_{n+1} &= T_n \bowtie_{f_{1,n+1}=k_{n+1}} R_{n+1} \\ T_{n+2} &= T_{n+1} \bowtie_{f_{1,x}=k_{n+2}} R_{n+2}, \text{ where } x \in \{2, \ldots, n+1\} \end{aligned}$$

Notice that by the iteration $n+2$, all RICs from R_a to R_b were already used once. Therefore, this expansion require that we use one of the RICs twice, which would lead to redundancy. For instance, if assume $x = 2$, without loss of generality, then:

$$\begin{aligned} 4T_2 &= T_1 \bowtie_{f_{1,2}=k_2} R_2 \\ T_{n+2} &= T_{n+1} \bowtie_{f_{1,2}=k_{n+2}} R_{n+2} \end{aligned}$$

As the join conditions are stacked in each iteration, we can say that:

$$f_{1,2} = k_2 \wedge f_{1,2} = k_{n+2}$$

which implies that $k_2 = k_{n+2}$ and, thus, all the returning JNTs would have more than one occurrence of the same tuple for every instance of the database.

$$T_{m+1} = T_m \bowtie_{f_{1,x}=k_{m+1}} R_{m+1}$$

□

APPENDIX G CJNGen ALGORITHM

The generation and ranking of CJNs is carried out by Algorithm 8, CJNGen, which uses a *Breadth-First Search* approach [18] to expand JNKMs until they comprehend all elements from a query match.

Despite being based on the MatCNGen Algorithm [2], CJNGen provides support for generating CJNs wherein there exists more than one RIC between one database relation to another, due to the definition of soundness presented in Theorem 1. Also, CJNGen does not require an intermediate structure such as the *Match Graph* in the MatCNGen system.

We describe CJNGen in Algorithm 8. For each query match, CJNGen generates the candidate joining networks for this query match using an internal algorithm called CJNInter, which we will focus on describing in the remainder of this section.

Algorithm 8 CJNGen($\mathcal{RQM}$, G_S)

Input: The set of ranked query matches $\mathcal{RQM}$ The schema graph G_S
Output: The set of candidate networks CJN

1 $CJN = \{\}$
2 **for** query match $M \in \mathcal{RQM}$ **do**
3 $\quad CJN_M \leftarrow$ **CJNInter**(M, G_S)
4 $\quad CJN \leftarrow CJN \cup CJN_M$
5 **return** CJN

In Algorithm 9, we present CJNInter. This algorithm takes as input a query match M and the schema graph G_S. Next, it chooses a KM from the QM as a starting point, resulting in an unitary graph (Lines 3-4). If the query match M has only one element, we already generated the one possible candidate joining network (Line 6).

Next, the CJNInter initializes a queue D, which is used to store the JNKMs which are not CJNs (Lines 7-8). In Loop 9-27, CJNInter takes one JNKM J from the queue and tries to expand it with KMs. N otice that J can be expanded with incoming and outgoing neighbors, therefore it uses an undirected schema graph G_S^U (Line 14). Also, the elements of M can only be added once in a JNKM but keyword-free matches can be added several times.

The expansion of J results in a JNKM J' (Lines 18-19). Then, CJNInter verifies whether J' was already generated and whether it is *sound*, according to Definition 9. If J' fails to meet these two conditions it is pruned (Line 20).

If J' was not pruned, CJNInter checks whether J' covers the query match M. If it does, J' is a candidate joining network and it will be added to the list CJN. If J' does not cover M, then it will be added to the deque D (Lines 21 -24). At the end of the procedure, CJNInter returns the set CJN of candidate joining networks for the query match M (Line 28).

Note that the complexity of the CJN generation is mainly because of two factors: (1) There can be multiple KMs for each subset of keywords. As a result, there may be several ways of combining these KMs into QMs, so that all keywords are covered. (2) Given QM, there can be many distinct ways of connecting its elements through PK/FK constraints and keyword-free matches.

Lathe implements some basic CJN pruning strategies to help decrease the candidate space, which are based on the following parameters: the top-k CJNs, the top-k CJNs per QM and the maximum CJN size. Also, the algorithm implements a few strategies to prune the JNKMs which are not minimal or not sound, the maximum node Degree, the maximum number of keyword-free matches, and the distinct foreign keys.

Algorithm 9 CJNInter(G_S, M, $score_M$)

Input: The query match M; The schema graph G_S
Output: A set CJN of candidate networks for the query match M

1 $CJN \leftarrow [\,]$
2 $J \leftarrow$ Graph()
3 **let** KM be an element from M
4 Add KM to $J.\mathcal{V}$
5 **if** $|M| = 1$ **then**
6 $\quad$ **return** $\{J\}$
7 $D \leftarrow$ queue()
8 D.enqueue(J)
9 **while** $D \neq \{\}$ **do**
10 $\quad J \leftarrow D$.dequeue()
11 $\quad$ **for** $KM_u \in J.\mathcal{V}$ **do**
12 $\quad\quad$ **let** KM_u be $R_u^S[A_{u,1}^{K_{u,1}^S}, \ldots, A_{u,m}^{K_{u,m}^S}]^V[A_{u,1}^{K_{u,1}^V}, \ldots, A_{u,m_u}^{K_{u,m_u}^V}]$
13 $\quad\quad$ **let** G_S^U be the undirected version of G_S
14 $\quad\quad$ **for** R_a adjacent to R_u in G_S^U **do**
15 $\quad\quad\quad$ **for** $KM_v \in M \setminus CN.\mathcal{V}$ **do**
16 $\quad\quad\quad\quad$ **let** KM_v be $R_v^S[A_{v,1}^{K_{v,1}^S}, \ldots, A_{v,m}^{K_{v,m}^S}]^V[A_{v,1}^{K_{v,1}^V}, \ldots, A_{v,m_v}^{K_{v,m_v}^V}]$
17 $\quad\quad\quad\quad$ **if** $R_v = R_a$ **then**
18 $\quad\quad\quad\quad\quad J' \leftarrow J$
19 $\quad\quad\quad\quad\quad$ Expand J' with KM_v joined to KM_u
20 $\quad\quad\quad\quad\quad$ **if** $J' \notin CN$ **and** J' is *sound* **then**
21 $\quad\quad\quad\quad\quad\quad$ **if** $J'.\mathcal{V} \supseteq M$ **then**
22 $\quad\quad\quad\quad\quad\quad\quad CN$.append($J'$)
23 $\quad\quad\quad\quad\quad\quad$ **else**
24 $\quad\quad\quad\quad\quad\quad\quad D$.enqueue($J'$)
25 $\quad\quad\quad J' \leftarrow J$
26 $\quad\quad\quad$ Expand J' with $R_a^S[\,]^V[\,]$ joined to KM_u
27 $\quad\quad\quad D$.enqueue(J')
28 **return** CJN

A. MAXIMUM NODE DEGREE

As the leaves of a CJN must be keyword matches from the query match, then a CJN must have at most $|\mathcal{QM}|$ leaves. Also, considering that the maximum node degree in a tree is less or equal to the number of its leaves, we can safely prune the JNKMs that contains a node with a degree greater than $|\mathcal{QM}|$.

B. MAXIMUM NUMBER of KEYWORD-FREE MATCHES

The size of a CJN is based on the size of the query match and the number of keyword-free matches, that is, the size of a candidate joining network CJN_M for a query match M is given by $|CJN_M|=|M|+|F|$, where F is a set of keyword-free matches. Thus, if we consider a maximum CJN size T_{max}, we can also set a maximum number of keyword-free matches for a CJN, given by $|F|\leq T_{max}-|M|$. Therefore, we can prune all JNKMs that contain more keyword-free matches than this maximum number set.

The number of CJNs generated can be further reduced by the pruning and ranking them. In Section VII-A, we present a ranking of the candidate joining networks returned by CJN-Gen. In Section VII-B, we present pruning techniques for the generation of the candidate joining networks from CJNGen and CJNInter.

REFERENCES

[1] S. Bergamaschi, E. Domnori, F. Guerra, R. T. Lado, and Y. Velegrakis, "Keyword search over relational databases: A metadata approach," in *Proc. 2011 ACM SIGMOD Int. Conf. Manag. Data*, 2011, pp. 565–576.

[2] P. Oliveira, A. da Silva, E. de Moura, and R. Rodrigues, "Match-based candidate network generation for keyword queries over relational databases," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 1344–1347.

[3] J. Coffman and A. C. Weaver, "A framework for evaluating database keyword search strategies," in *Proc. 19th ACM Int. Conf. Inf. Knowl. Manag.*, Oct. 2010, pp. 729–738.

[4] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword search in relational databases," in *Proc. 28th Int. Conf. Very Large Databases*. Amsterdam, The Netherlands: Elsevier, 2002, pp. 670–681.

[5] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A system for keyword-based search over relational databases," in *Proc. 18th Int. Conf. Data Eng.*, 2002, pp. 5–16.

[6] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-style keyword search over relational databases," in *Proc. 29th Int. Conf. Very Large Data Bases*, 2003, pp. 850–861.

[7] Y. Luo, X. Lin, W. Wang, and X. Zhou, "Spark: Top-k keyword query in relational databases," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Jun. 2007, pp. 115–126.

[8] J. Coffman and A. C. Weaver, "Structured data retrieval using cover density ranking," in *Proc. 2nd Int. Workshop Keyword Search Structured Data*, Jun. 2010, pp. 1–6.

[9] A. Baid, I. Rae, J. Li, A. Doan, and J. Naughton, "Toward scalable keyword search over relational data," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 140–149, Sep. 2010.

[10] P. de Oliveira, A. da Silva, and E. de Moura, "Ranking candidate networks of relations to improve keyword search over relational databases," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Apr. 2015, pp. 399–410.

[11] P. S. de Oliveira, A. da Silva, E. de Moura, and R. de Freitas, "Efficient match-based candidate network generation for keyword queries over relational databases," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 4, pp. 1735–1750, Apr. 2022.

[12] S. Bergamaschi, F. Guerra, M. Interlandi, R. Trillo-Lado, and Y. Velegrakis, "QUEST: A keyword search system for relational data based on semantic and machine learning techniques," *Proc. VLDB Endowment*, vol. 6, no. 12, pp. 1222–1225, Aug. 2013.

[13] K. Affolter, K. Stockinger, and A. Bernstein, "A comparative survey of recent natural language interfaces for databases," *VLDB J.*, vol. 28, no. 5, pp. 793–819, Oct. 2019.

[14] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, and S. Sudarshanxe, "BANKS: Browsing and keyword searching in relational databases," in *Proc. 28th Int. Conf. Very Large Databases*. Amsterdam, The Netherlands: Elsevier, 2002, pp. 1083–1086.

[15] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *Proc. 31st Int. Conf. Very Large Data Bases*, 2005, pp. 505–516.

[16] H. He, H. Wang, J. Yang, and P. S. Yu, "BLINKS: Ranked keyword searches on graphs," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Jun. 2007, pp. 305–316.

[17] F. Liu, C. Yu, W. Meng, and A. Chowdhury, "Effective keyword search in relational databases," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Jun. 2006, pp. 563–574.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.

[19] M. A. P. de Cristo, P. P. Calado, M. D. L. Da Silveira, I. Silva, R. Muntz, and B. Ribeiro-Neto, "Bayesian belief networks for IR," *Int. J. Approx. Reasoning*, vol. 34, nos. 2–3, pp. 163–179, 2003.

[20] J. Coffman and A. C. Weaver, "An empirical performance evaluation of relational keyword search techniques," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 30–42, Jan. 2014.

[21] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," in *Proc. 18th Int. Conf. Data Eng.*, 2002, pp. 431–440.

[22] S. Bergamaschi, F. Guerra, S. Rota, and Y. Velegrakis, "A hidden Markov model approach to keyword-based search over relational databases," in *Proc. Int. Conf. Conceptual Model.* Cham, Switzerland: Springer, 2011, pp. 411–420.

[23] S. Tata and G. M. Lohman, "SQAK: Doing more with keywords," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Jun. 2008, pp. 889–902.

[24] L. Blunschi, C. Jossen, D. Kossmann, M. Mori, and K. Stockinger, "SODA: Generating SQL for business users," *Proc. VLDB Endowment*, vol. 5, no. 10, pp. 932–943, Jun. 2012.

[25] M. S. Ramada, J. C. da Silva, and P. de Sá Leitão-Júnior, "From keywords to relational database content: A semantic mapping method," *Inf. Syst.*, vol. 88, Feb. 2020, Art. no. 101460.

[26] G. A. Miller, *WordNet: An Electronic Lexical Database*. Cambridge, MA, USA: MIT Press, 1998.

[27] T. Pedersen, S. Patwardhan, and J. Michelizzi, "WordNet: Similarity-measuring the relatedness of concepts," in *Proc. Demonstration Papers HLT-NAACL*, 2004, pp. 38–41.

[28] Z. Wu and M. Palmer, "Verbs semantics and lexical selection," in *Proc. 32nd Annu. Meeting Assoc. Comput. Linguistics*, 1994, pp. 133–138.

[29] V. Keselj, *Speech and Language Processing*, D. Jurafsky and J. H. Martin, Eds. Upper Saddle River, NJ, USA: Prentice-Hall, 2009.

[30] F. Mesquita, A. S. da Silva, E. S. de Moura, P. Calado, and A. H. F. Laender, "LABRADOR: Efficiently publishing relational databases on the web by using keyword-based query interfaces," *Inf. Process. Manag.*, vol. 43, no. 4, pp. 983–1004, Jul. 2007.

[31] B. A. N. Ribeiro and R. Muntz, "A belief network model for IR," in *Proc. 19th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 1996, pp. 253–260.

[32] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Amsterdam, The Netherlands: Elsevier, 2014.

[33] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval: The Concepts and Technology Behind Search*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2008.

[34] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manag.*, vol. 24, no. 5, pp. 513–523, Jan. 1988.

[35] W. May, "Information extraction and integration with Florid: The Mondial case study," Univ. Freiburg, Institut für Informatik, Freiburg im Breisgau, Germany, Tech. Rep. 131, 1999. [Online]. Available: http://dbis.informatik.uni-goettingen.de/Mondial

[36] A. Afonso, P. Martins, and A. da Silva, "SEREIA—Busca por palavras-chave em document stores," in *Proc. 36th Simpósio Brasileiro de Bancos de Dados*, 2021, pp. 133–144.

[37] N. Yaghmazadeh, Y. Wang, I. Dillig, and T. Dillig, "SQLizer: Query synthesis from natural language," *Proc. ACM Program. Lang.*, vol. 1, pp. 1–26, Oct. 2017, doi: 10.1145/3133887.

[38] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, Apr. 2007, pp. 836–845.

[39] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum, "STAR: Steiner-tree approximation in relationship graphs," in *Proc. IEEE 25th Int. Conf. Data Eng.*, Mar. 2009, pp. 868–879.

[40] P. Martins, A. Afonso, and A. Da Silva, "PyLatheDB—A library for relational keyword search with support to schema references," in *Proc. IEEE 39th Int. Conf. Data Eng. (ICDE)*, Apr. 2023, pp. 3627–3630.
[41] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, "From word embeddings to document distances," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 957–966.
[42] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[43] M. Trabelsi, J. Cao, and J. Heflin, "SeLaB: Semantic labeling with BERT," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8.
[44] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018, *arXiv:1802.05365*.
[45] P. Yin, G. Neubig, W.-T. Yih, and S. Riedel, "TaBERT: Pretraining for joint understanding of textual and tabular data," 2020, *arXiv:2005.08314*.
[46] M. Trabelsi, Z. Chen, S. Zhang, B. D. Davison, and J. Heflin, "StruBERT: Structure-aware BERT for table search and matching," 2022, *arXiv:2203.14278*.
[47] S. Idreos, O. Papaemmanouil, and S. Chaudhuri, "Overview of data exploration techniques," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, May 2015, pp. 277–281.
[48] P. Schirmer, T. Papenbrock, I. Koumarelas, and F. Naumann, "Efficient discovery of matching dependencies," *ACM Trans. Database Syst.*, vol. 45, no. 3, pp. 1–33, Sep. 2020.
[49] C. Chanial, R. Dziri, H. Galhardas, J. Leblay, M.-H.-L. Nguyen, and I. Manolescu, "Connectionlens: Finding connections across heterogeneous data sources," *Proc. VLDB Endowment*, vol. 11, no. 12, pp. 2030–2033, Aug. 2018.
[50] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000.
[51] T. Hearne and C. Wagner, "Minimal covers of finite sets," *Discrete Math.*, vol. 5, no. 3, pp. 247–251, Jul. 1973.

PAULO MARTINS received the M.Sc. degree in informatics from the Institute of Computing, Universidade Federal do Amazonas (IComp/UFAM), Brazil, in 2020, where he is currently pursuing the Ph.D. degree in informatics. His research interests include relational databases, keyword query procedures, and information retrieval systems.

ALTIGRAN SOARES DA SILVA received the Ph.D. degree from Universidade Federal de Minas Gerais (UFMG), in 2002. He is currently a Full Professor with the Institute of Computing, Universidade Federal do Amazonas (IComp/UFAM). He has also been the co-founder of successful technology ventures. He has published more than 130 scientific publications in high quality venues and served on committees in Brazil and abroad. His research interests include data management and information retrieval.

ARIEL AFONSO received the M.Sc. degree in informatics from the Institute of Computing, Universidade Federal do Amazonas (IComp/UFAM), Brazil, in 2020, where he is currently pursuing the Ph.D. degree in informatics. His research interests include relational databases, document store exploration, and keyword query procedures.

JOÃO CAVALCANTI received the Ph.D. degree from The University of Edinburgh, in 2003. He is currently a Professor with the Institute of Computing, Universidade Federal do Amazonas (IComp/UFAM). He has also been the co-founder of successful technology startups. His research interests include databases, information retrieval, and applications in multimedia retrieval and computational photography.

EDLENO DE MOURA received the Ph.D. degree in computer science from Universidade Federal de Minas Gerais (UFMG), Brazil, in 1999. He is currently a Full Professor with the Institute of Computing, Universidade Federal do Amazonas (IComp/UFAM), where he heads the Database and Information Retrieval Group. He is the author of several articles in journals and conference proceedings covering topics related to information retrieval, such as efficiency issues, text indexing, ranking algorithms, text classification, text compression, and image search.