SUPPORTING SCHEMA REFERENCES IN KEYWORD QUERIES OVER RELATIONAL DATABASES

PAULO RODRIGO OLIVEIRA MARTINS

SUPPORTING SCHEMA REFERENCES IN KEYWORD QUERIES OVER RELATIONAL DATABASES

Dissertation presented to the Graduate Program in Informatics of the Federal University of Amazonas in partial fulfillment of the requirements for the degree of Master in Informatics.

Advisor: Altigran Soares da Silva

Manaus

March 2020

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Martins, Paulo Rodrigo Oliveira
M386s Supporting Schema References in Keyword Queries over Relational Databases / Paulo Rodrigo Oliveira Martins . 2020
58 f.: il. color; 31 cm.

Orientador: Altigran Soares da Silva
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Keyword Search. 2. Relational Databases. 3. Schema References. 4. Candidate Networks. I. Silva, Altigran Soares da. II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Supporting Schema References in Keyword Queries over Relational Databases"

PAULO RODRIGO OLIVEIRA MARTINS

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Altigran Soares da Silva - PRESIDENTE

Prof. Edleno Silva de Moura - MEMBRO INTERNO

Prof. João Marcos Bastos Cavalcanti - MEMBRO EXTERNO

Manaus, 27 de Março de 2020

Abstract

Relational Keyword Search (R-KwS) systems enable naive/informal users to retrieve information from relational databases without any knowledge about schema details or query languages. Roughly, R-KwS systems must determine which pieces of information to retrieve from the database and how to connect them to generate a relevant answer for the user. For instance, the keywords from the query may refer either to instance values or to database schema elements, such as relations and attributes names. Unfortunately, most of the R-KwS systems do not support references to the database schema. In this work, we propose a new method to generate an adequate SQL query from an input keyword query, taking advantage of references to the content of the database as well as references to the database schema. Our experiments indicate that schema references can help to improve the quality of the ranking the answers for the user.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Relational Keyword Search (R-KwS) enables naive/informal users to retrieve information from relational databases without any knowledge about schema details or query languages. The success of search engines demonstrates that untrained users are comfortable using keyword search to find documents of their interest.

However, empowering users to search relational databases using keyword queries is a challenging task considering that the information sought often spans multiple relations and attributes, according to the schema design of the underlying database. Thus, systems that process keyword queries over relational databases face the challenge of automatically determining which pieces of information to retrieve from the database and how to connect them to generate a relevant answer to the user.

In general, the keywords from a query may refer to both database values, such as tuples containing these keywords, and schema elements, such as relation and attribute names. For instance, consider the query Q_1 = "*will smith* $films$" over a database on movies. The keywords "*will*" and "*smith*" may refer to values of person names. In contrast, the keyword "$films$" is likely to refer to a schema element, the name of the relation about movies.

Although a significant number of keywords in keyword queries correspond to schema references (Bergamaschi et al., 2011a), most of the previous work in the literature on R-KwS systems associate keywords to instance values only, that is, they do not support references to schema information such as the one in query Q_1. Therefore, given the query Q_1, they will look for tuples that contain the keyword "$films$", which is unlikely to produce an useful answer for the user.

In fact, there are only a few work in the literature proposing methods for handling schema references in keywords queries. One of the first systems that support such queries was BANKS (Bhalotia et al., 2002). However, it only considers schema refer-

ences in which keywords exactly match with the names of schema elements. Therefore, it does not support common case such as synonyms. SQAK (Tata e Lohman, 2008) is capable of handling these cases, however it requires an ontology to be supplied for each target database domain. Keymantic (Bergamaschi et al., 2011a) and KEYRY (Bergamaschi et al., 2011b) properly addresses the problem of schema references support. However, they both require user feedback to select the relevant answers. Also, they do not consider database instances and, thus, they cannot benefit from valuable instance-related features exploited by other state-of-the-art R-KwS systems.

In this work we study new techniques for supporting schema references in keyword queries over relational databases. Specifically, we proposed Lathe, a new method to generate an adequate SQL query from an input keyword query, considering that keywords may refer either to instance values or to database schema elements, i.e., relations and attributes.

Lathe follows the so-called *Schema Graph* approach for R-KwS. Given a keyword query Q, this approach consists of first generating relational algebra expressions called *Candidate Networks* or *CNs*, which are likely to express the user intention when formulating the original query Q. Then, the generated CNs are *evaluated*, that is, they are translated in SQL queries, which are executed by a DBMS, and the resulting tuples are collected and supplied to the user. Several systems also propose strategies to rank the resulting tuples, so that tuples that are more likely to fulfill the user intentions are ranked higher. In the literature, the most well-known algorithm to generate CNs is CN-Gen, which was proposed by the pioneer Schema Graph R-KwS systems DISCOVER (Hristidis e Papakonstantinou, 2002). This algorithm is used as default in most of the methods proposed in the literature for CN evaluation and ranking of resulting tuples, such as Efficient (Hristidis et al., 2003), SPARK (Luo et al., 2007), CD (Coffman e Weaver, 2010b).

As the problem of evaluating CNs is known to be very costly, recent proposals have appeared in the literature to make it more efficient. KwS-F (Baid et al., 2010) imposed a time limit for CN evaluation, returning possibly partial results and a summary of the CNs yet to be evaluated. CNRank (Oliveira et al., 2015) proposed a ranking of CNs, so that only the top-ranked CNs need to be evaluated. MatCNGen (Oliveira et al., 2018) proposed a novel method for generating candidate networks, wherein the possible matches for the query in the database are efficiently enumerated at first. These *Query Matches* or *QMs* are then used to guide the CN generation process, further decreasing the number of generated CNs.

Our method Lathe is, to the best our knowledge, the first method to address the problem of generating and ranking Candidate Networks considering queries with

keywords that refer to both schema elements and instance values.

Roughly, Lathe matches the keywords from the query to both schema elements and tuples from the database that contains these keywords. Next, we combine the keyword-matches in order to gather all the pieces of information that covers the keyword query. These combinations, which represent query matches, are then ranked. Then, we generate the candidate networks, which connect all the necessary pieces of information for the keyword query. After that, the candidate networks are ranked and evaluated. Finally, the results from the candidate networks evaluation are delivered to the user.

To verify the effectiveness of Lathe we performed several experiments. First, we evaluated the ranking of Candidate Networks using the Mean Reciprocal Rank (MRR) and the Precision at position K (P@K). Regarding performance, we evaluate the average elapsed time to generate the candidate networks. Next, we also evaluate the impact of instance-based pruning in the generation and ranking of candidate networks using MMR and P@K. Lastly, we analyzed the ranking of *Query Matches* using P@K. The performed experiments indicate our support to schema references can help to improve the ranking of candidate networks and disambiguate queries.

Our key contributions are as follows: (i) we propose a novel method for generating and ranking candidate networks with support for schema references. (ii) we propose the first algorithm for Ranking of Query Matches, which prunes the processing of less likely answers. (iii) we present several pruning techniques for the generation and ranking of candidate networks. (iv) we propose a simple but effective ranking of candidate networks which exploits the ranking of query matches. (v) we defined keyword matches, query matches and candidate networks which supports schema references.

The remainder of this dissertation is organized as follows: Chapter 2 reviews the related literature in the field of Relational Keywords Search Systems based on Schema Graphs and Support to Schema References. Chapter 3 overviews all the phases from our method, which are detailed in Chapters 4-6. Chapter 7 reports the results of the experiments we have conducted. Finally, Chapter 8 presents the conclusions we have reached and outlines our future work.

Chapter 2

Background and Related Work

In this chapter we discuss the background and related work on the problem of Keyword Search Systems over Relational Databases and the Support to Schema References in R-KwS systems.

2.1 Keyword Search Systems over Relational Databases

Current R-KwS systems fall in one of two distinct categories: systems based on Schema Graphs and systems based on Data Graphs. Systems in the first category are based on the concept of Candidate Networks (CNs), which are networks of joined relations that are used to generate SQL queries whose results provide an answer to the input keyword query. This approach was proposed in DISCOVER (Hristidis e Papakonstantinou, 2002) and DBXplorer (Agrawal et al., 2002) and was later adopted by a number of other systems, such as Efficient (Hristidis et al., 2003), SPARK (Luo et al., 2007), CD (Coffman e Weaver, 2010b), KwS-F (Baid et al., 2010), CNRank (Oliveira et al., 2015) and MatCNGen (Oliveira et al., 2018). Systems in this category take advantage of the basic functionality of the underlying RDBMS by producing appropriate SQL join queries to retrieve answers relevant to keyword queries posed by users.

Systems in the second category are based on structures called Data Graphs, whose nodes represent tuples associated with the keywords they contain, and the edges connect these tuples based on referential integrity constraints. In this approach, adopted by several systems, including BANKS (Aditya et al., 2002), Bi-directional (Kacholia et al., 2005), BLINKS (He et al., 2007) and Effective (Liu et al., 2006), results of keyword queries are computed by finding subtrees in a data graph that minimizes the

distance between nodes matching the given keywords.

2.2 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 a RDBMS capable of processing SQL queries.

The most well-known algorithm to generate candidate networks is CNGen, which was proposed in DISCOVER (Hristidis e Papakonstantinou, 2002). This algorithm was later adopted as default in most of the R-KwS systems proposed in the literature.

However, CNGen often too many times generates a large number of CNs, resulting in a costly process of CN evaluation. Mainly to avoid the multi-query optimization problem that arises when all CNs are to be evaluated, several work were proposed for ranking of resulting tuples of CNs, such as Efficient (Hristidis et al., 2003), SPARK (Luo et al., 2007) and CD (Coffman e Weaver, 2010b).

In contrast, KwS-F Baid et al. (2010) addressed the efficiency and scalability problems in CN evaluation in a different way. Their approach consists of two steps. First, a limit is imposed on the time the system spends evaluating CNs. After this limit is reached, the system must return a (possibly partial) top-K JNTs result. Second, if there are CNs yet to be evaluated, these CNs are presented to the user by means of query forms, so the user can select one of the forms and the system evaluates the corresponding CN.

CNRank (Oliveira et al., 2015) presented an approach to reduce the cost of the CN evaluation by ranking candidate networks. This ranking is based on the probability of CNs to produce relevant answers to the user. Specifically, CNRank presented a probabilistic ranking model that uses a Bayesian belief network to estimate the relevance of a Candidate Network given the current state of the underlying database. A score is assigned to each generated candidate networks so that only a few CNs with the highest scores are evaluated.

MatCNGen (Oliveira et al., 2018) proposed a match-based approach for generating CNs. MatCNGen enumerates the possible ways that the query keywords can be matched in the database beforehand, to generate query answers. Thereafter, for each of these query matches, MatCNGen generates a single CN, which drastically reduces the time required to generate CNs. In addition, MatCNGen assumes that answers must contain all the query terms, it follows that every keyword must appear in at least one element of a candidate network. Thus, as the generation process avoids generating

too many combinations of keyword occurrences, a smaller but better set of CNs is generated.

Lastly, Coffman e Weaver (2010a) proposed a framework for evaluating R-KwS systems and reported the results of applying this framework over three representative standardized datasets they built, namely Mondial, IMDb and Wikipedia, along with respective query workloads. The authors compare nine R-KwS systems, evaluating them in many aspects related to their effectiveness and performance. These resources built in this framework were also used in the experiments of several work, such as Coffman e Weaver (2012), Luo et al. (2007) and Oliveira et al. (2015, 2018).

2.3 Support to Schema References in R-KwS Systems

In fact, there are only a few works in the literature proposing methods for handling schema references in keywords queries. One of the first system that supports such queries was BANKS (Bhalotia et al., 2002). Although, in that work, the identification of schema references considered only the exact matches between a keyword and a schema element. Therefore, misspellings, abbreviations and synonyms of schema elements were ignored. Besides, BANKS is a R-KwS system based on Data Graphs, which is out of the scope of this work.

SQAK (Tata e Lohman, 2008) presented more robust support to schema references by using an ontology-based normalization and an edit distance based measure. In fact, the focus of SQAK was on handling more complex keyword queries, allowing the usage of aggregation functions. For that reason, SQAK noted that aggregation functions were generally succeeded by their targets: schema elements. For instance, in the query "students average grade", the average function targets the attribute grade. Interestingly, SQAK showed us that keywords are not independent of one another.

Keymantic (Bergamaschi et al., 2011a) was the first work to specifically address the problem of schema references support. In addition to the mapping of keywords to either schema elements or instance values, Keymantic also looked for inter-dependencies between query keywords, that is, how the keywords from the query affect one another. For instance, in the query "actor denzel washington", we can exploit the mapping of "actor" to the relation person to increase the likelihood of "washington" be associated to a person name, instead of movie title. Furthermore, Keymantic also used a combination of different string similarity metrics and the semantic dictionary WordNet(Miller, 1998) in the keyword mappings.

KEYRY (Bergamaschi et al., 2011b) further exploited the inter-dependencies among keywords using a *Hidden Markov Model*. On the other hand, KEYRY also needed to exploit user feedback to improve the results. As a consequence, there is often a need to have the user to manually select the more appropriate answers. Unfortunately, both Keymantic and KEYRY do not consider database instances and, thus, they cannot benefit from most of the other contributions in the literature.

Chapter 3

Lathe Overview

In this chapter, we present an overview of our system Lathe for generating SQL queries given a keyword query with references to the database schema

We begin by presenting a simple example of the task carried out by the method. For this, we use the sample movie database we illustrate in Figure 3.1. This database is actually a simplified excerpt of the well-known IMDB[1].

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

MOVIE

	ID	Title	Year
t_6	6	Men in Black	1997
t_7	7	I am Legend	2007
t_8	8	Harry Potter and the Sorcerer's Stone	2001
t_9	6	The Lord of the Rings: The Fellowship of the Ring	2001
t_{10}	10	The Lord of the Rings: The Return of the King	2003
t_{11}	11	Silent Hill	2006

CHARACTER

	ID	Name
t_{12}	12	Agent J
t_{13}	13	Robert Neville
t_{14}	14	Marcus Flint
t_{15}	15	Minerva McGonagall
t_{16}	16	Boromir
t_{17}	17	Frodo Baggins
t_{18}	18	Christopher da Silva

ROLE

	ID	Name
t_{19}	19	Actor
t_{20}	20	Actress
t_{21}	21	Producer
t_{22}	22	Writer
t_{23}	23	Director
t_{24}	24	Editor

CASTING

	ID	Person_ID	Movie_ID	Char_ID	Role_ID
t_{25}	25	1	6	12	19
t_{26}	26	1	7	13	19
t_{27}	27	2	8	14	19
t_{28}	28	3	8	15	20
t_{29}	29	4	9	16	19
t_{30}	30	4	10	16	19
t_{31}	31	4	11	18	19
t_{32}	32	5	9	17	19
t_{33}	33	5	10	17	19

Figure 3.1: A sample movie database taken from IMDB

Consider that a user inputs the keyword query Q="*will smith films*", and assume that she wants the system to list the movies in which Will Smith appears. Notice that, in this query, terms "*will*" and "*smith*" are likely to match the contents of some relation in database, while the term "films" is likely to match to the name of a relation or attribute.

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

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

(a)

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

(b)

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

(c)

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

(d)

Figure 3.2: SQL queries generated for the keyword query "*will smith movies*" and their returned results.

As other methods previously proposed in the literature, such as CNGen (Hristidis e Papakonstantinou, 2002) and MatCNGen (Oliveira et al., 2018) the main goal of Lathe is, given a query such as Q, generating a SQL query that, when executed, fulfills the user's information need. The difference between Lathe and the previous methods is that they are not able to handle references to schema elements, such as "films" in Q.

For query Q, two of the possible SQL queries that would be generated are queries S_1 and S_2, presented in Figures 3.2 (a) and (b), respectively. The respective results of these queries for the database of Figure 3.1 are presented in Figures 3.2(c) and (d). Query S_1 retrieves the movies in which Will Smith is an actor, and thus, corresponds to the original user intent. On the other hand, query S_2 retrieves movies in which two different persons whose names include the terms "*will*" and "*smith*" are actors.

As this simple example shows, there may be several of plausible SQL, queries given a keyword query. Thus, it is necessary to decide which of these alternatives are more likely to fulfill the user need. This task is also carried out by Lathe.

In the next section, we present an overview on the components and the functioning of Lathe.

3.1 System Architecture

Now we present the overall architecture of Lathe. We base our discussion on Figure 3.3, which illustrates the main phases that comprise the method's functioning.

The process begins with an input keyword query posed by a user. The system then tries to associate each of the keywords from the query with either instance values or schema elements from the database. This phase, called *Keyword Matching* (1),

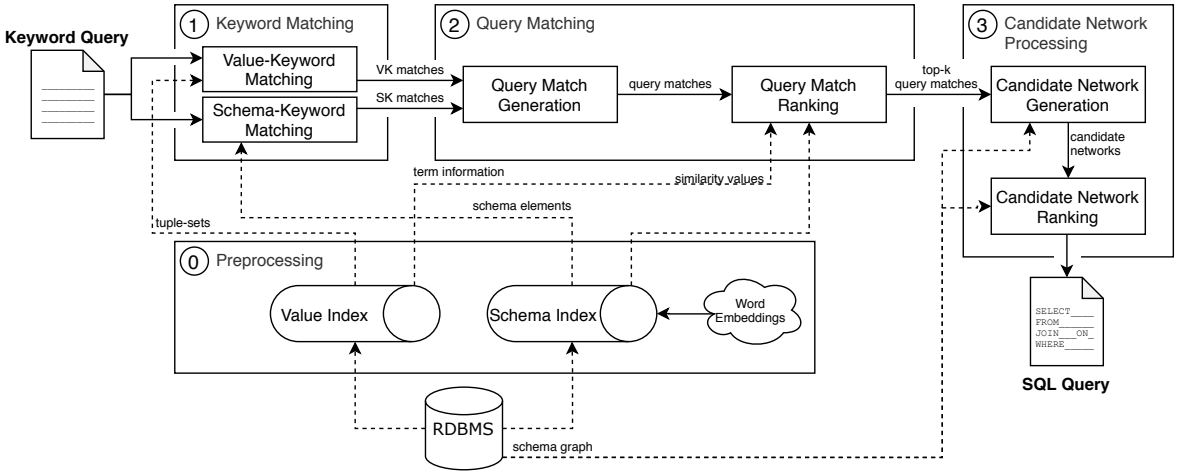

Figure 3.3: Lathe's main phases

generates sets of *Value-Keyword Matches*, which associate keywords with sets of tuples whose attribute values contain these keywords, and *Schema-Keyword Matches*, which associate keywords with names of relations or attributes deemed as similar to these keywords.

In the next phase, *Query Matching* (2), keyword matches are combined so that their combined keywords cover the input query in a *total* and *minimal* way. This means that all keywords from the query must be included by the keyword matches and no keyword match is redundant. These combinations of keyword matches are called *Query Matches*. Since there may be a large number of such combinations, the method may often generates many query matches. However, only a few of them are in fact useful to produce plausible answers to the user. For this reason, we propose the first algorithm for *Ranking Query Matches*. This ranking scores query matches based on the likelihood that they fulfill the user needs when formulating the keyword query. Thus, the system only outputs a few top-ranked query matches to the next phases. By doing so, it avoids having to process less likely query matches.

Once we have query matches, the system tries to connect each piece of information from them in the *Candidate Network* (3) phase. That is, the system search for interpretation for the keyword query. The candidate network are based on relational algebra, which can be directly translated into SQL queries. Next, we take advantage of the ranked query matches to generate a *Candidate Network Ranking*. Roughly, this ranking favors candidate networks that are more concise in the number of relation it uses. Once we have the most likely candidate networks, we perform a translation of candidate networks into SQL queries.

During the process of generating SQL queries, 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 stores the occurrences of keyword in the database, indicating the relations and tuples a keyword appear and which attributes are mapped to that keyword. These occurrences are retrieved in the Query Matching phase. In addition, the Value Index is also used to calculate *term frequencies* for the Ranking of Query Matches. The Schema Index is an inverted index that stores information about the database schema and statistics about ranking of attributes, which is also used in the Query Matches Ranking.

In the following chapters we present each of the phases of Figure 3.3, describing their steps, definitions, data structures, the algorithms used and discussing the main project decisions that we made. We begin by presenting Keyword Matching process in Chapter 4, describing the value-keyword and schema-keyword matches and their respective generation algorithms *VKMGen* and *SKMGen*. After that, we present the Query Matching process in Chapter 5, describing the conditions for combining keyword matches into query matches, which is acomplished by the algorithm QMGen. We also present QMRank, the first algorithm for ranking of query matches. Then, in Chapter 6, we present the generation of candidate networks, we also present a straightforward ranking of candidate networks, which takes advantage of the previous ranking of query matches, and several pruning techniques to decrease the number of candidate networks.

Chapter 4

Keyword Matching

The first phase in the method, *Keyword Matching* associates the keywords from the query to either instance values or schema elements, which respectively generate *Value-Keyword Matches*, and *Schema-Keyword Matches*.

4.1 Value-Keyword Matching

We associate the keywords from the query to the tuples that contain these keywords by means of *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 $\exists |K_i| \geq 1$, $W(t[A_i])$ returns the set of words in t for attribute A_i, K_i is the set of keywords from Q that are associated to the attribute A_i and V denotes a match of keywords to the database values.

Notice that, since $W(t[A_i]) \cap Q = K_i$, each tuple from the database can be a member of only one value-keyword match. Therefore, the value-keyword matches 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 value-keyword matches. 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 3.1. The following value-keyword*

matches can be generated for the query "will smith films".

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

Value-keyword matches play a similar role to the tuple-sets from related literature (Hristidis e Papakonstantinou, 2002; Oliveira et al., 2018), however, they are more expressive, since they specify which attribute is associated with each keyword. In contrast, some R-KwS systems, such as MatCNGen (Oliveira et al., 2018), cannot create tuple-sets that span over different attributes. Example 2 illustrates a keyword query that span over more than one attribute.

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

$$MOVIE^V[title^{\{lord,rings\}}, year^{\{2001\}}] = \{t_9\}$$

VKMGen Algorithm

The generation of value-keyword matches is carried out by the Algorithm 1. It takes tuples from the database in which the keywords occur and uses them to form value-keyword matches.

The algorithm directly retrieves the occurrences of each keyword from a structure we call the *Value Index*. This index is built in a preprocessing phase that scans only once all the target relations. This phase precedes the processing of queries and we assume that it does not need to be repeated often. Thus, the answers are generated for each query without any further interaction with the DBMS. The Value Index follows the structure below:

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

which stores the occurrences of keywords in the database, indicating the relations and tuples a keyword appear and which attributes are mapped to the keyword.

Example 3. *The value-keyword matches presented in Example 1 are based on the fol-*

lowing keyword occurrences:.

$$
\begin{aligned}
I_v[will] &= \{PERSON : \{name : \{t_1, t_2\}\}\} \\
I_v[smith] &= \{PERSON : \{name : \{t_1, t_3\}\}\} \\
I_v[smith][PERSON] &= \{name : \{t_1, t_3\}\} \\
I_v[smith][PERSON][name] &= \{t_1, t_3\}
\end{aligned}
$$

In Algorithm 1, once we obtained the occurrences of each keyword, we convert them to the value-keyword structure (Line 7). However, they are not proper value-keyword matches yet, because we need to guarantee that they are disjoint sets.

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
   /* Step 1:  retrieve keyword occurrences */
2  O ← ∅
3  for keyword k_i ∈ Q do
4      if k_i ∈ I_V.keys then
5          for relation R_j ∈ I_V[k_i].keys do
6              for attribute A_k ∈ I_V[k_i][R_j].keys do
7                  R_j^V[A_k^{k_i}] ← I_V[k_i][R_j][A_k]
8                  O.append(R_j^V[A_k^{k_i}])
   /* Step 2:  generate value-keyword matches */
9  VK = VKMIter(O)
10 return VK
```

The task of ensuring that the value-keyword matches are disjoint sets is carried out by Algorithm 2, VKMIter, which is based on the ECLAT algorithm (Zaki, 2000) for finding frequent itemsets.

The algorithm looks for non-empty intersections of the pseudo value-keyword matches recursively until all of them are disjoint sets, and thus, proper value-keyword matches. These intersections are calculated as follows:

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

where $K_{ab,i} = K_{a,i} \cup K_{b,i}$. Then, the algorithm updates the previous value-keyword

Algorithm 2: VKMIter(VK)

Input: A set of value-keyword matches VK

for pair of value-keyword matches $\{KM_a, KM_b\} \in \binom{VK}{2}$ **do**
  $KM_{ab} \leftarrow KM_a \cap KM_b$
  if $KM_{ab} \neq \{\}$ **and** KM_{ab} is valid **then**
    VK.append(KM_{ab})
    $KM_a \leftarrow KM_a - KM_{ab}$
    if $KM_a = \{\}$ **then**
      VK.remove(KM_a)
    else
      VK.update(KM_a)
    $KM_b \leftarrow KM_b - KM_{ab}$
    if $KM_b = \{\}$ **then**
      VK.remove(KM_b)
    else
      VK.update(KM_b)
    VKMIter(VK)

matches, by removing the elements of their intersection from them:

$$R_a^V[A_{a,1}^{K_{a,1}}, \ldots, A_{a,m}^{K_{a,m}}] \Leftarrow R_a^V[A_{a,1}^{K_{a,1}}, \ldots, A_{a,m}^{K_{a,m}}] \setminus R_{ab}^V[A_{ab,1}^{K_{ab,1}}, \ldots, A_{ab,m}^{K_{ab,m}}]$$

$$R_b^V[A_{b,1}^{K_{b,1}}, \ldots, A_{b,m}^{K_{b,m}}] \Leftarrow R_b^V[A_{b,1}^{K_{b,1}}, \ldots, A_{b,m}^{K_{b,m}}] \setminus R_{ab}^V[A_{ab,1}^{K_{ab,1}}, \ldots, A_{ab,m}^{K_{ab,m}}]$$

4.2 Schema-Keyword Matching

Similarly to what we do for references to instance values, our method also addresses references to the schema. Specifically, our method generates *Schema-Keyword Matches*, which associate keywords from the query to schema elements. This is accomplished using similarity metrics, which we will further explain later.

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 k 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 K_i is the set of keywords from Q that are associated to the schema element A_i, S denotes a match of keywords to the database schema and $sim(A_i, k)$ gives the similarity between the name of a schema element A_i and the keyword, which must be

above a threshold ε.

The schema-keyword matches play a different role in our method than the value-keyword matches. The main objective of a schema-keyword matches is ensuring that the attributes of a relation appear in the query results. In our method, they do not denote any selection operation over database relations. This is the reason why they do not "filter'" any of the tuples from the relation.

We generated an artificial attribute called $self$ to be used when a keyword that is matched 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* $\varepsilon = 0.6$.

$$
\begin{aligned}
&MOVIE^S[self^{\{films\}}] &&= \{t_6, t_7, t_8, t_9, t_{10}, t_{11}\}, && sim(movie, films) = 1.00\\
&MOVIE^S[title^{\{will\}}] &&= \{t_6, t_7, t_8, t_9, t_{10}, t_{11}\}, && sim(title, will) = 0.87\\
&PERSON^S[name^{\{smith\}}] &&= \{t_1, t_2, t_3, t_4, t_5\}, && sim(name, smith) = 0.63
\end{aligned}
$$

Similarity Metrics

For the matching of keywords to schema elements, we used two similarity metrics based on the lexical database *WordNet*: the *Path similarity* (Miller, 1998; Pedersen et al., 2004) and the *Wu-Palmer similarity* (Wu e Palmer, 1994; Pedersen et al., 2004). We introduce the WordNet database and the two similarity metrics below.

WordNet Database

WordNet (Miller, 1998) is a large lexical database which superficially resembles a thesaurus, in that it groups words together based on their meanings. One use of WordNet is to represent *word senses*, the many different meanings that a single lemma can have (Keselj, 2009). Thus the lemma "film" can refer to a movie, to the act of recording of to the plastic film.

WordNet also represents relations between senses, such as *synonymy*, *hyponymy* and *hypernymy*. If two word senses have the same meaning, the are synonyms. 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 example, *"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.

Path Similarity

The Path similarity (Miller, 1998; Pedersen et al., 2004) 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:

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

Wu-Palmer Similarity

The Wu-Palmer measure (WUP) (Wu e Palmer, 1994; Pedersen et al., 2004) 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 LCS is the most specific synset c_3 which is ancestor of both synsets c_1 and c_2.The score can never be zero because the depth of the LCS is never 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:

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

SKMGen Algorithm

The retrieval of schema-keyword matches is pretty straightforward, which is detailed in Algorithm 3. First, SKMGen iterates over the relations and attributes using a structure called *Schema Index* (Lines 2-3), which is built in the preprocessing phase. This index stores information about the database schema and statistics about the ranking of attributes, which will be explained in Chapter 5. The stored information follows the structure below:

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

Then, SKMGen calculates the similarity between each keyword and schema element. It only considers the schema-keyword matches whose similarities are above a threshold ε (Line 8).

Once SKMGen find a similarity between a keyword and a schema element above the threshold, it generates a schema-keyword match(Line 9). The keyword match

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.keys do
4          if sim(k_i, R_j) ≥ ε then
5              R_j^S[self^{k_i}] ← tuples(R_i)
6              SK ← SK ∪ {R_j^S[self^{k_i}]})
7          for attribute A_l ∈ I_S[R_j].keys do
8              if sim(k_i, A_l) ≥ ε then
9                  R_j^S[A_l^{k_i}] ← tuples(R_i)
10                 SK ← SK ∪ {R_j^S[A_l^{k_i}]})
11 return SK
```

receives all the tuples from relation R, since schema-keyword matches do not perform any filter operation over the tuples. Notice that SKMGen algorithm generates schema-keyword matches associated with a single keyword.

4.3 Generalization of Keyword Matches

In this chapter, we presented the Definitions 1 and 2 which, respectively, introduces Value-Keyword Matches and Schema-Keyword Matches. We choose to explain the specificity of these concepts separately for the sake of better understanding. Definition 3 formally presents the concept of Keyword Matches. This generalization will be useful when merging value-keyword matches and schema-keyword matches in the next phases.

Definition 3. *Let Q be a keyword query and R be a relation state over the relation schema $R(A_1, \ldots, A_m)$. A* ***keyword match*** *from R over Q is given by:*

$$R^S[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]^V[A_1^{K_1^V}, \ldots, A_m^{K_m^V}] = \{t | t \in R \wedge (\forall k \in K_i^S : sim(A_i, k) \geq \varepsilon) \\ \wedge ((\nexists |K_i^V| \geq 1) \vee (\forall A_i : W(t[A_i]) \cap Q = K_i^V))\}$$

where $W(t[A_i])$ returns the set of words in t for attribute A_i and $sim(A_i, k)$ gives the similarity between the name of a schema element A_i and the keyword, which must be above a threshold ε.

The representations of value-keyword matches and schema-keyword matches in

the general notation are given as follows:

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

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

Chapter 5

Query Matching

This chapter describes the processes of generating and ranking query matches, which are combinations of the keyword matches generated in the previous phases that comprises every keyword from the keyword query.

5.1 Query Matches Generation

The objective of this step is combining the keyword matches generated in the previous phases to create query matches, that is, configurations of keyword matches in which every keyword from the query must appear in at least one of the keyword matches.

Intuitively, there may be many different combinations, however, Lathe considers as query matches only those in which the combinations of keywords correspond to minimal set covers for the query. This restriction ensures that query matches are total and not redundant.

Definition 4. *Let Q be a keyword query. Let VK be the set of all value-keyword matches and SK be the set of all schema-keyword matches for Q in a certain database instance I. Consider a subset M of $VK \cup SK$ of the form*

$$\begin{aligned} M = \{ & R_1^S[A_{1,1}^{K_{1,1}^S}, \ldots, A_{1,m_1}^{K_{1,m_1}^S}]^V[A_{1,1}^{K_{1,1}^V}, \ldots, A_{1,m_1}^{K_{1,m_1}^V}], \\ & \ldots, \\ & R_n^S[A_{n,1}^{K_{n,1}^S}, \ldots, A_{n,m_n}^{K_{n,m_n}^S}]^V[A_{n,1}^{K_{n,1}^V}, \ldots, A_{n,m_n}^{K_{n,m_n}^V}]\} \end{aligned}$$

*Also, let $C = \bigcup_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m_i \\ X \in \{S,V\}}} K_{i,j}^X$ be the set of all keywords associated with the keyword matches in M. M is called a **query match** for Q if, and only, if C forms a **minimal***

set cover *of the keywords in Q. That is, $C = Q$ and $C \setminus \{K^X_{i,j} | 1 \leq j \leq m_i \wedge X \in \{S, V\}\} \neq Q$, for any $1 \leq i \leq n$.*

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

$M_1 = \{PERSON^V[name^{\{will,smith\}}], MOVIE^S[self^{\{films\}}]\}$
$M_2 = \{PERSON^V[name^{\{will\}}], PERSON^V[name^{\{smith\}}], MOVIE^S[self^{\{films\}}]\}$
$U_1 = \{PERSON^V[name^{\{will\}}], PERSON^V[name^{\{smith\}}]\}$
$U_2 = \{PERSON^V[name^{\{will,smith\}}]\}$
$U_3 = \{PERSON^V[name^{\{will,smith\}}], PERSON^V[name^{\{smith\}}], MOVIE^S[self^{\{films\}}]\}$

The sets M_1 and M_2 are considered query matches. In contrast, the sets of keyword matches U_1, U_2 and U_3 are not query matches. While U_1 and U_2 do not include all query keywords, U_3 is not minimal, that is, it has unnecessary keyword matches.

QMGen Algorithm

The generation of query matches is carried out by Algorithm 4, QMGen, which preserves the same proposed idea as in Oliveira et al. (2018), but our method adapts it to keyword matches instead of tuple-sets. Let V and S be respectively sets of value-keyword matches, and schema-keyword matches previously generated. The algorithm looks for combinations of keyword matches in $V \cup S$ 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 $V \cup S$ to obtain the complete set of query matches. However, it can be shown that any minimal cover of a set of n elements has at most n subsets (Hearne e Wagner, 1973). Therefore, no match for a query Q can be formed by more than $|Q|$ keyword matches.

For this reason, Algorithm 4 iterates over all the subsets of $V \cup S$ whose size is less or equal than the size of the query, and checks which of them form a minimal cover for the query. The evaluation of minimal cover is carried out by Algorithm 5.

The Algorithm 5 first analyzes if a given set of keyword matches U covers the query Q, ensuring that U is total. Then, the algorithm checks whether U continues to be total if any one of its elements is removed. Lastly, Algorithm 6 looks for possible mergings of the keyword matches from the query matches. Considering that schema-keyword matches do not denote any selection operation over a database relation, they can be merged with other keyword matches from the same relation.

Algorithm 4: QMGen(Q, VK, SK)

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

```
1 H = VK ∪ SK
2 QM ← {}
3 for i ∈ {1, ..., |Q|} do
4     let H[i] be set of subsets of size i of H
5     foreach U ∈ H[i] do
6         if MinimalCover(U, Q) then
7             MergeKeywordMatches(U)
8             QM.append(U)
9 return QM
```

Algorithm 5: MinimalCover(Q, U)

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

```
1  Cover_U ← {}
2  for i ∈ {1, ..., n} do
3      for j ∈ {1, ..., m_i} do
4          Cover_U ← Cover_U ∪ {K_{i,j}}
5  if Cover_U ⊉ Q then
6      return False
7  for i ∈ {1, ..., n} do
8      Cover_{R_i} ← {}
9      for j ∈ {1, ..., m_i} do
10         Cover_{R_i} ← Cover_{R_i} ∪ {K_{i,j}}
11     if Cover_U \ Cover_{R_i} ⊉ Q then
12         return False
13 return True
```

5.2 Query Matches Ranking

As described in Chapter 3, Lathe performs a ranking of the query matches generated in the previous step. This ranking is necessary because often many query matches are generated, yet, only a few of them are in fact useful to produce plausible answers to the user.

Algorithm 6: MergeKeywordMatches(Q, U)

Input: The set of keyword matches U
Output: The set o keyword matchs U with a lower size, if possible

```
1 foreach R_a^S[A_{a,1}^{K_{a,1}}, ..., A_{a,m_a}^{K_{a,m_a}}] ∈ U do
2     foreach R_b^S[A_{b,1}^{K_{b,1}^S}, ..., A_{b,m_b}^{K_{b,m_b}^S}]^V[A_{1,b}^{K_{1,b}^V}, ..., A_{b,m_b}^{K_{b,m_b}^V}] ∈ U do
3         if a ≠ b and R_a = R_b then
4             for i ∈ 1, ..., m_b do
5                 K_{b,i}^S ← K_{b,i}^S ∪ K_{a,i}
6             U ← U − {R_a^S[A_{a,1}^{K_{a,1}}, ..., A_{a,m_a}^{K_{a,m_a}}]}
```

Lathe estimates the relevance of query matches based on the current state of the underlying database using a *Bayesian Belief Network* model. In practice, this model considers two different types of relevancy score for the query matches to be ranked. The *value-based score* is calculated according to the TF-IDF model, which adapts the traditional vector space model to the context of relational databases, similarly as done in LABRADOR (Mesquita et al., 2007) and CNRank (Oliveira et al., 2015). On the other hand, the *schema-based score* is obtained by estimating the similarity between keywords and the names of schema elements.

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

Belief Bayesian Network

In this section, we describe the Bayesian Belief Network Model we used for the *Ranking of Query Matches*. We adopt the Bayesian framework proposed by Ribeiro e Muntz (1996) and de Cristo et al. (2003) for modeling distinct IR problems. This framework is simple and allows combining features of distinct models into the same representational scheme. In addition, it also has been used by other keyword search systems, such as LABRADOR (Mesquita et al., 2007) and CNRank (Oliveira et al., 2015).

In our model, we interpret the query matches as documents, which are ranked for the keyword query. Figure 5.1 illustrates an example of the Bayesian Network we adopt. The Query Side, at the top of the network, contains the nodes that represent the keyword query. The Database Side, at the bottom of the network, contains the nodes that represent the query match that will be scored. The middle of the network is present in both sides and it is composed by two set of keywords: the set V of all

terms present in the values of the database, and the set S of the names of all schema elements.

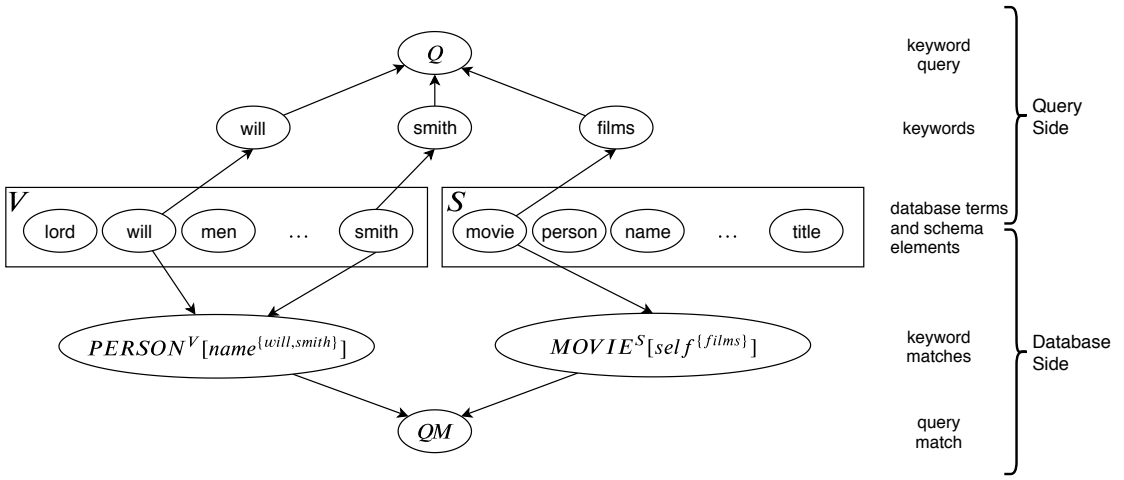

Figure 5.1: Bayesian network corresponding to the query $Q = \{will, smith, films\}$

In our Bayesian Network, we rank query matches 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:

$$P(QM|Q) = \mu P(QM \wedge Q)$$

where $\mu = 1/P(Q)$ is a normalizing constant, as used in Pearl (2014).

We define a random binary variable associated with each keyword from the sets V and S, which indicates whether the keyword was observed. Since these random variables are the roots of our Bayesian Network, all of the probabilities of the other nodes are dependent of 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 event c as follows:

$$P(c) = P(c|v, s) \times P(v) \times P(s)$$

Since there is no preference for any particular set of terms, all possibilities of v and s are equally likely a priori, then we can calculate each probability as $P(v) = (1/2)^{|V|}$ and $P(s) = (1/2)^{|S|}$.

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 splited between the probability of each of its keywords:

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

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.

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

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

$$P(QM|v,s) = \prod_{1 \leq i \leq |QM|} P(R_i^S[A_{i,1}^{K_{i,1}^S}, \ldots, A_{i,m_i}^{K_{i,m_i}^S}]^V[A_{1,i}^{K_{1,i}^V}, \ldots, A_{i,m_i}^{K_{i,m_i}^V}]|v,s)$$

We compute the probability of keyword matches in two different ways: the probabilities of schema-keyword matches are based on the same similarities used in the generation of schema-keyword matches and, the probabilities of value-keyword matches are based on a vector model (Baeza-Yates e Ribeiro-Neto, 2008; Salton e Buckley, 1988), using the cosine similarity.

$$P(R_i^S[A_{i,1}^{K_{i,1}^S}, \ldots, A_{i,m_i}^{K_{i,m_i}^S}]^V[A_{1,i}^{K_{1,i}^V}, \ldots, A_{i,m_i}^{K_{i,m_i}^V}]|v,s) = \begin{cases} \prod\limits_{1 \leq j \leq m_i} cos(\overrightarrow{A_{i,j}}, \overrightarrow{v \cap K_{i,j}^V}) \\ \prod\limits_{\substack{1 \leq j \leq m_i \\ t \in s \cap K_{i,j}^S}} sim(A_{i,j}, t) \end{cases}$$

It is important to distinct the documents from the Bayesian network model and the Vector Model. While the documents of the Bayesian Network are the query matches and the query is the keyword query itself. The documents of the Vector model are the attributes from the database and the query is the set of keywords associated with the keyword match.

Once we know the document and the query of the vector model, we can derive the cosine similarity from the inner product of the document and the query. The cosine

similarity formula is given as follows:

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

where $\alpha = 1/\sqrt{\sum_{t \in V} w(\overrightarrow{v \cap K^V_{i,j}}, t)^2}$ 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 specificness in the collection. For the term frequency, we used the Augmented Normalized Term Frequency (Salton e Buckley, 1988), which is given as follows:

$$tf(X, t) = \left(0.5 + 0.5 \times \frac{freq_{X,t}}{\max_{l \in V} freq_{X,l}}\right)$$

where $X \in \{cos(\overrightarrow{A_{i,j}}, \overrightarrow{v \cap K^V_{i,j}}\}$ can be either the document or the query. In case of 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 case of 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 use the Inverted Document Frequency(IDF)Salton e Buckley (1988); Baeza-Yates e Ribeiro-Neto (2008) for measuring the specificness of terms, which is given by the formula below:

$$idf(t) = \log\left(\frac{N_A}{n_t}\right)$$

where N_A is the number of attributes in the database and n_t is the number of attributes which are mapped to the occurrences of the term t.

We chose the Normalized form of the Term Frequency suggested in Salton e Buckley (1988) because our vector model does not benefit much from the Inverted Document Frequency. This happens because generally a word occurs in all of the text attributes from the database or it does not occur in any of them at all.

QMRank Algorithm

The ranking of query matches is carried out by Algorithm 7. The score of a query match is based on the score of its keyword matches. Thus, the algorithm iterates over each keyword match from each query match, scoring them.

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**
  $value\_prod \leftarrow 1$, $schema\_prod \leftarrow 1$,
  $vk\_count \leftarrow 0$, $sk\_count \leftarrow 0$
  foreach $R_i^S[A_{i,1}^{K_{i,1}^S}, \ldots, A_{i,m_i}^{K_{i,m_i}^S}]^V[A_{1,i}^{K_{1,i}^V}, \ldots, A_{i,m_i}^{K_{i,m_i}^V}] \in M$ **do**
    for $j \in \{1, \ldots, m_i\}$ **do**
      if $\exists |K_{i,j}^V| \geq 1$ **then**
        $vk\_count \leftarrow vk\_count + 1$
        $(norm, maxfreq) \leftarrow I_S[R_i][A_j]$
        /* where $norm = |A_j|$ and $maxfreq = \max_{t \in S} freq_{A_j,t}$ */
        $value\_sum \leftarrow 0$
        for $t_k \in K_{i,j}$ **do**
          $num\_occurrences \leftarrow |I_V[t_k][R_i][A_j]|$
          $tf \leftarrow 0.5 + 0.5 \times num\_occurrences/maxfreq$
          $value\_sum \leftarrow value\_sum + tf \times iaf(t_k)$
        $value\_prod \leftarrow value\_prod \times value\_sum/norm$
      if $\exists |K_{i,j}^S| \geq 1$ **then**
        $sk\_count \leftarrow sk\_count + 1$
        $schema\_sum \leftarrow 0$ **for** $t_k \in K_{i,j}$ **do**
          if $A_j = self$ **then**
            $schema\_sum \leftarrow schema\_sum + similarity(R_i, t_k)$
          else
            $schema\_sum \leftarrow schema\_sum + similarity(A_j, t_k)$
        $schema\_prod \leftarrow schema\_prod \times schema\_sum$
  $final\_score \leftarrow 1$
  $value\_score \leftarrow 0$
  $schema\_score \leftarrow 0$
  if $vk\_count > 0$ **then**
    $value\_score \leftarrow value\_prod$
    $final\_score \leftarrow final\_score \times value\_score$
  if $sk\_count > 0$ **then**
    $schema\_score \leftarrow schema\_prod$
    $final\_score \leftarrow final\_score \times schema\_score$
  RQM.append($(M, final\_score, value\_score, schema\_score)$)
RQM.sortDescBy($final\_score$)
return RQM

In case of the schema-keyword matches, the score is based on the word similarity functions presented in Chapter 4. In case of the value-keyword matches, the score is based on the cosine similarity is calculated be using TF-IDF weights. To compute the IDF, the algorithm obtains the maximum frequencies in a attribute and its norm directly from the Schema Index. Similarly, the algorithm obtains the Term Frequencies from the Value Index. Both of these indexes are presented in Chapter 4.

Once the algorithm aggregates the scores of keyword matches to generate the score of query matches, the final step is to sort them in descending order.

Chapter 6

Candidate Networks Generation

In this chapter we present the details on our approach for generating Candidate Networks (CNs), which represent possible interpretations for the keyword query.

6.1 Concepts

The generation of CNs 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 PK/FK relationships in the database schema. In practice, this graph is built in a preprocessing phase using the database schema.

Definition 5. *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:*

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

*where $RIC(R_x, R_y)$ indicates that there exists a Referential Integrity Constraint from a relation R_x to a relation R_y. We say that a **schema graph** is an ordered pair $G_S = \langle \mathcal{R}, E\rangle$, where $\mathcal{R}$ are the vertices (nodes) of G_S, and E are the edges of G_S.*

Example 6. *Considering the sample movie database introduced in Figure 3.1, our method generates the schema graph bellow.*

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

In Figure 6.1 we represent a graphical illustration of G_S.

$$\begin{array}{ccccc} & & ROLE & & \\ & & \uparrow & & \\ PERSON & \leftarrow & CASTING & \rightarrow & MOVIE \\ & & \downarrow & & \\ & & CHARACTER & & \end{array}$$

Figure 6.1: A schema graph for the sample movie database of Figure 3.1

Another concept require for the generation of CNs are *keyword-free matches*, which are described in Definition 6. They are used as intermediate nodes in candidate networks.

Definition 6. *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, $\nexists K_i^S \neq \{\} \wedge \nexists K_i^V \neq \{\}$.*

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

Once we defined the schema graph and keyword-free matches, we can properly introduce candidate networks. Intuitively, a candidate network *CN* contains every keyword match from a query match M. Also, for the sake of connectivity, *CN* may contain some free-keyword matches. Lastly, the *CN* is a connected graph structured according to the schema graph G_S. The definition of candidate networks 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. Consider a graph of keyword matches $CN = \langle \mathcal{V}, E \rangle$. We say that CN is a **candidate network** of keyword matches from M over G_S if the following conditions hold:*

$$\begin{aligned} &i)\ \ \mathcal{V} = M \cup F \\ &ii)\ \forall KM_i \in \mathcal{V} : \exists \langle KM_a, KM_b \rangle \in E | KM_i = KM_a \vee KM_i = KM_b \\ &iii) \forall \langle KM_a, KM_b \rangle \in E \implies \exists \langle R_a, R_b \rangle \in G_S \end{aligned}$$

For the sake of simplifying the notation, we will use a graphical illustration to represent candidate networks.

Example 7. *Considering the query match M_1 previously generated in Example 5, the following candidate networks can be generated:*

$$CN_1 = PERSON^V[name^{\{will,smith\}}] \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}]$$

$$CN_2 = PERSON^V[name^{\{will,smith\}}] \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}]$$
$$\downarrow$$
$$CHARACTER$$

The candidate networks CN_1 and CN_2 cover the query match M_1. The interpretation of CN_1 looks for the movies *of the* person *will smith. CN_2 looks for the* movies *of the* person *will smith and which* character *will smith played in this movies.*

Notice that a candidate network might have unnecessary information for the keyword query, which was the case of CN_2 presented in Example 7. One approach to avoid generating unnecessary information is to generate minimal candidate networks, which are addressed in Definition 8. Roughly, a minimal candidate network 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. Let $CN = \langle \mathcal{V}, E \rangle$ be a candidate network from M over G_S . We say that CN is* ***minimal*** *if, and only if, the following condition holds:*

$$\forall KM_a \in \mathcal{V}(\ |\{\ \{KM_a, KM_b\}\ |\langle KM_a, KM_b\rangle \in E\}| = 1 \implies KM_a \neq R_a^S[\ ]^V[\ ]\ )$$

Example 8. *Considering the query match M_2 previously generated in Example 5, the following minimal candidate network can be generated:*

$$CN_3 = PERSON^V[name^{\{smith\}}] \leftarrow CASTING \rightarrow PERSON^V[name^{\{will\}}]$$
$$\downarrow$$
$$MOVIE^S[self^{\{films\}}]$$

Another problem a candidate network might have is representing an inconsistent interpretation. For instance, it is impossible to the CN_3 presented in Example 8 return any results from the database. By Definition 1, the value-keyword matches $PERSON^V[name^{\{will\}}]$ and $PERSON^V[name^{\{smith\}}]$ are disjoint. However, a tuple from $CASTING$ cannot refer to two different tuples of $PERSON$. Thus CN_3 is inconsistent.

One approach to prevent inconsistency was presented in Hristidis e Papakon-

stantinou (2002). It consists of verifying whether a candidate network contains an inconsistent sub-graph. Definition 9 formalizes this approach.

Definition 9. *Let G_S be a schema graph. Let CN be candidate network given by $CN = \langle \mathcal{V}, E \rangle$. We say that CN is a **sound** if, and only if, the following condition holds:*

$$\forall \{KM_a, KM_b, KM_c\} \subseteq \mathcal{V} | R_a = R_c \implies \nexists \{\langle KM_b, KM_a \rangle, \langle KM_b, KM_c \rangle\} \subseteq E$$

Example 9. *Considering the query match M_2 previously generated in Example 5, the following sound candidate network can be generated:*

$$CN_4 = MOVIE^S[self^{\{films\}}] \leftarrow CASTING \rightarrow PERSON^V[name^{\{will\}}]$$
$$\uparrow$$
$$CASTING \rightarrow PERSON^V[name^{\{smith\}}]$$

The candidate networks CN_4 covers the query matches M_2. CN_4 is a minimal and sound candidate network. The interpretation of CN_4 looks 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 candidate network CN_4.

6.2 Candidate Network Generation

In this section, we present CNKMGen, our proposed algorithm for generating candidate networks from the top-K query matches previously generated.

Roughly, for each query match, CNKMGen uses a *Breadth-First Search* approach (Cormen et al., 2009) to expand partial trees of keyword matches until they become candidate networks for this query match. Then, CNKMGen returns all the candidate networks for all the query matches.

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

In Algorithm 9, we present CNKMIter. This algorithm takes as input a query match M and the schema graph G_S. Then, it algorithm chooses and arbitrary keyword match from the query match as a starting point, resulting in an unitary graph (Lines 2-5). If the query match M has only one element, we already generated the one possible candidate network (Line 7).

Algorithm 8: CNKMGen(RQM, G_S)

Input: The set of ranked query matches RQM
The schema graph G_S
Output: The set of candidate networks CNs

```
1  CNs = {}
2  foreach query match M ∈ RQM do
3      CN_M ← CNKMGenPerQM(M, G_S)
4      CNs ← CNs ∪ CN_M
5  return CNs
```

Algorithm 9: CNKMInter(M, G_S)

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

```
1  let G_S^U be the undirected version of G_S
2  let KM be an element from M
3  CN ← [ ]
4  C ← graph()
5  Add KM to C.V
6  if |M| = 1 then
7      return {C}
8  D ← queue()
9  D.enqueue(C)
10 while D ≠ {} do
11     C ← D.dequeue()
12     foreach keyword match KM_u ∈ CN.V do
13         let R_u be the relation of KM_u
14         foreach relation R_a adjacent R_u in G_S^U do
15             foreach keyword match KM_v ∈ M \ CN.V | R_v = R_a do
16                 C' ← C
17                 Expand C' with KM_v joined to KM_u
18                 if C' ∉ CN and C' is sound then
19                     if C'.V ⊇ M then
20                         CN.append(C')
21                     else
22                         D.enqueue(C')
23             C' ← C
24             Expand C' with the keyword-free match R_a^S[ ]^V[ ] joined to KM_u
25             D.enqueue(C')
26 return CN
```

Next, the CNKMIter initializes a queue D, which will be used to store the partial trees (Lines 8-9). In Loop 10-25, CNKMIter takes one partial tree C from the queue and tries to expand it with keyword matches. Notice that C can be expanded with incoming and outgoing neighbors, therefore it uses an undirected schema graph G_S^U(Line 14). Also, non-free keyword matches can only be added once in a partial tree.

The expansion of C results in a new partial tree C' (Lines 16-17). Then , CNKMIter verifies whether C' was not generated. Also, the algorithm check if C'

is *sound*, according to Definition 9. If the new partial tree C' fails to meet these two conditions it is pruned (Line 18).

If C' was not pruned, it will either be added in the deque D or it already is a candidate networks. C' is considered a candidate network if it covers the query match M(Lines 19 -22). At the end of the procedure, CNKMIter returns a set of candidate networks CN(Line 26).

For better understanding, we present part of the execution of the CNKMIter algorithm in Figure 6.2. In this case, the execution looks for the candidate networks for the query match M_1 from Example 5, which is given as follows:

$$M_1 = \{PERSON^V[name^{\{will,smith\}}], MOVIE^S[self^{\{films\}}]\}$$

I#	Queue	Operations
T_1	$PERSON^V[name^{\{will,smith\}}]$	initialize, enqueue 0
T_2	$PERSON^V[name^{\{will,smith\}}] \leftarrow CASTING$	expand 0, enqueue 1
T_3	$PERSON^V[name^{\{will,smith\}}] \leftarrow CASTING$ $\uparrow$ $CASTING$	expand 1, enqueue 2
T_4	$PERSON^V[name^{\{will,smith\}}] \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}]$	return generated CN

Figure 6.2: Execution of CNKMIter algorithm for generating the candidate networks for the query match M_1

Initially, the arbitrary keyword match chosen was $PERSON^V[name^{\{will,smith\}}]$ and the unitary partial tree T_1 generated. Next, since there is no keyword match from M_1 whose relation is neighbor to $PERSON$ in the schema graph, T_1 is not expanded with any non-free keyword match. In this same iteration, the algorithm expands T_1 with the keyword-free match, $CASTING$, resulting in partial tree T_2

In the last iteration, CNKMIter selects T_2 and it is expanded twice. First, the join tree is expanded with another keyword-free match from relation $CASTING$, resulting in join tree T_3. Then, CNKMIter expands the current partial tree with $CASTING$ joined to $PERSON^V[name^{\{will,smith\}}]$.

The number of candidate networks generated can be further reduced by the pruning and ranking candidate networks. In Section 6.3, we present a ranking of the candidate networks returned by CNKMGen. In Section 6.4, we present pruning techniques for the generation of candidate networks from CNKMGen and CNKMIter.

6.3 Candidate Network Ranking

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

We present in Section 5.2 a ranking of query matches that advances most of the features present in the ranking of candidate networks of other proposed systems, such as CNRank (Oliveira et al., 2015). Thus, we can exploit the scores of query matches to rank the candidate networks. For this reason, CNKMRank provides a simple yet effective ranking of candidate networks.

CNKMRank is described in Algorithm 10. Roughly, it uses the ranking of query matches adding a penalization for large candidate networks. Therefore, the score of a candidate network CN_M from a query match M is given by:

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

To ensure that CNs with the same score are placed in the same order that they where generated in the CNKMIter algorithm, in Line 6, we used a stable sorting algorithm (Cormen et al., 2009).

Algorithm 10: CNKMRank(QM)

Input: A set of candidate networks CN
Output: The set of candidate networks RCN

```
1 RCN ← [ ]
2 for CN_M ∈ RCN do
3 |   let M be the query match used to generate CN_M
4 |   cn_score = score(M)/ |CN_M|
5 |   RCN.append( (CN_M, cn_score) )
6 RCN.sortDescBy(cn_score)
7 return RCN
```

6.4 Candidate Network Pruning

In this section, we present several pruning techniques for candidate networks. These pruning techniques can be divided into schema-based and instance-based pruning techniques.

6.4.1 Schema-based Pruning

The schema-based pruning techniques aim to decrease the number of candidate networks using the maximum values for the number of generated CNs or using maximum values for CN properties, such as number of leaves or size. We incorporated all the schema-based pruning techniques in Lathe by default.

Top-K CNs per QMs

This pruning technique considers only the first K candidate networks generated for each query match. Therefore, CNKMIter returns the set of candidate networks when it reaches the maximum size of K.

Top-k CNs

This pruning technique considers only the top-K of the ranked candidate networks. Therefore, CNKMRank, after the sorting or CNs by score, returns a list of the first K elements.

Maximum Number of Leaves

As the leafs in minimal candidate networks must be keyword matches from the query match, the candidate networks generated by CNKMIter must have at most $|QM|$ leaves. Therefore, we present a prune technique for the CNKMIter Algorithm that prunes all the partial trees that contain more than $|QM|$ leaves.

Maximun CN Size

This pruning technique consider only the candidate networks whose size are below a specific number T_{max}. Therefore, CNKMIter prunes all the partial trees whose size is above T_{max}.

Maximum Number of Keyword-free Matches

This pruning technique is also based on the Maximum CN Size mentioned earlier. As the size of a candidate network CN_M is given by:

$$|CN_M| = |M| + |F|$$

where F is a set of keyword-free matches. If we consider a maximum CN size of $|CN| \leq T_{max}$, we can define a maximum number of keyword-free matches, which is

given by:

$$|F| \leq T_{max} - |M|$$

Therefore, CNKMIter Algorithm can prune all the partial trees whose number of free-keyword matches are above $T_{max} - |M|$.

6.4.2 Instance-based Pruning

The instance-base pruning techniques aim to decrease the number of candidate networks retrieving information from the database instance. As we cannot guarantee that the SQL query associated with a candidate networks will return any tuples, there might exist CNs that are not be useful for returning an answer for the user. These techniques are not incorporated in Lathe by default.

Applying instance-based pruning techniques incur in a unavoidable increase in the generation time. This is due to the queries we need to issue to the DBMS. Techniques similar to those used in by SQL query optimizers can be used to mitigate this issues. Currently, we left this issue to be addressed as a future work.

Pos-Prunning CNs with Empty Results

This pruning technique evaluates the ranked candidate networks in a sequential order, pruning the CNs whose SQL queries do not return any tuples. As this technique prunes CNs after the generation and ranking of candiadte networks, we call this technique Pos-Pruning CNs with Empty Results, or simply CN-Pos.

Pre-Prunning CNs with Empty Results

This pruning technique evaluates candidate networks as soon as they are generated, pruning the ones whose SQL queries do not return any tuples. In this technique CNKMIter immediately evaluates the CNs before adding to the list of returned CNs. Therefore, this technique prunes CNs before the generation of all candidate networks. For this reason, we call this technique Pre-Prunning CNs with Empty Results, or simply CN-Pre.

Chapter 7

Experiments

In this chapter, we report a set of experiments we performed with our method. For a given keyword query, our method aims at generating a ranking of Candidate Networks (CNs), so that the best ranked CNs are more likely to correctly represent the user intent when formulating the query. Thus, in the experiments we report here, our ultimate goal is to evaluate the quality of this ranking. In addition, we evaluate the performance of our method for generating candidate networks. We also evaluate the impact of instance-based pruning in generating and ranking CNs. Lastly, as the quality of a CN is highly influenced by the quality of the Query Matches (QM) that generates this CNs, we evaluated, as an intermediate result, the ranking of the QMs our method generates.

7.1 Experimental Setup

Datasets

For all the experiments, we used two datasets, *IMDb* and *MONDIAL*, which were previously used for the experiments performed by Coffman e Weaver (2010a, 2012), Luo et al. (2007), Oliveira et al. (2015, 2018), among others.

The IMDb dataset is a subset of the well-known Internet Movie Database (IMDb)[1], which comprises information related to films, television shows and home videos – including actors, characters, etc.

The MONDIAL dataset (May, 1999) comprises geographical and demographic information from the well-known *CIA World Factbook*[2] , the International Atlas, the

[1]https://www.imdb.com/

[2]https://www.cia.gov/library/publications/the-world-factbook/

TERRA database, and other web sources.

The two datasets have different features. The IMDb dataset has a larger size, but the MONDIAL dataset is more complex, with more relations and relational integrity constraints (RICs). Table 7.1 summarizes the details of each dataset.

Table 7.1: Datasets we used in our experiments

Dataset	Size(MB)	Relations	RIC	Tuples
MONDIAL	9	29	40	17,115
IMDb	516	5	4	1,673,074

Query Sets

We used in our experiments the query sets provided by Coffman e Weaver (2010a) for the datasets IMDb and MONDIAL.

An importante drawback we notice is that several queries from both query sets do not have a clear intent, compromising the evaluation of the results, for instance, the ranking of candidate networks generated. Therefore, for the the sake of providing a more fair evaluation, we generated new query sets 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, wherein *Saint Kitts* and *Cambodia* are the name of two countries. There exist several interpretations that might change the way tuples corresponding to these countries are connected. For instance, one may be looking for common religions, languages or ethnic groups between the two countries. While all this alternatives would, in principle, make valid interpretations, the relevant interpretation defined by Coffman e Weaver (2010a) in their golden standard indicates that the query looks for organizations which both countries are member of. Thus, in this case, we replaced this query by the query "Saint Kitts Cambodia Organizations" in the new query set we generated.

Table 7.2 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.

Golden Standards

Coffman e Weaver (2010a) provides the set of the relevant interpretations for each query and its relevant SQL results. We used them to evaluate the effectiveness of our

Table 7.2: 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
MOND	MONDIAL	45	7	12
MOND-DI	MONDIAL	45	-	19

ranking of Candidate Networks and the ranking of Query Matches.

To obtain the golden standards for CNs and QMs we used the following procedure. For each keyword query, we took the relevant interpretation defined by Coffman e Weaver (2010a). Then, we manually generate the candidate network that represents this relevant interpretation. We define this generated candidate network as relevant. Next, we took the set of nodes from the relevant CN that are not free-keyword matches. This set of keyword matches is a relevant query match.

Metrics

We evaluate the Ranking of Query Matches and the Ranking of Candidate Networks using two metrics: Precision at position ranking K ($P@K$) and Mean Reciprocal Rank (MRR).

Given a keyword query Q , the value of $P_Q@K$ is 1 if the target query for Q appears in a position up to K in the ranking, and 0 otherwise. $P@K$ is the average of $P_Q@K$, for all Q in a query set.

With respect to MRR given a keyword query Q, the value of RR_Q is given by $\frac{1}{K}$, where K is the rank position of the relevant result. Then, the MRR obtained for queries in a query set is the average of RR_Q, for all Q in the query set. Intuitively, the MRR metric measures how close the relevant results are from the first position of the ranking.

7.2 Evaluation of Candidate Network Ranking

In this section, we present two evaluations of the ranking of candidate networks with respect to the relevance and generation time. In both of the performed experiments, we consider only the top-10 query matches, where each query match produces a single candidate network.

In this experiment, we evaluate the ranking of Candidate Networks. We used the metrics MRR and P@K for K up to the 4 rank position to evaluate the quality of the

ranking. Figure 7.1 presents the results obtained for the ranking of candidate networks from the query sets presented in Table 7.2.

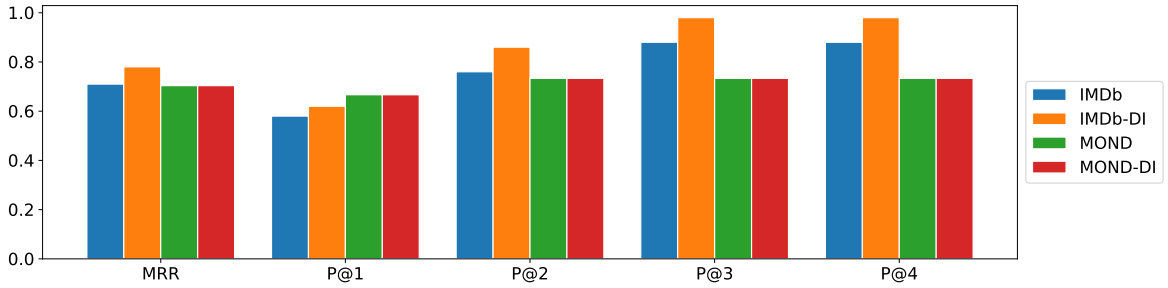

Figure 7.1: Ranking of Candidate Networks

Overall, the query sets obtained an P@3 above 0.7, which indicates that the relevant answer is often found among the top-3 returned candidate network. In addition, all the query sets obtained an MRR above 0.7, which indicates that the relevant candidate network is, on average, found in the first or the second position.

Regarding the disambiguation of queries, IMDb-DI obtained better results than IMDB. However, MOND-DI obtained the same results as MOND. This indicates that adding a schema reference was not enough to disambiguate the queries for MONDIAL dataset, probably due to its large number of RICs and respectively number of possible CNs. However, the addition of schema references did not decrease the quality of results, which indicates that adding schema references can help to improve the quality of the CN ranking.

7.3 Performance Evaluation

In this experiment, we evaluate the time for obtaining the candidate networks given a keyword query. This process includes the phases of Keyword Matching, Query Matching and the Generation and Ranking of Candidate Networks. Figure 7.2 summarizes the execution time for the queries from each query set using box plot graphs.

Box plots are a standardized way of displaying the distribution of data based on a five number summary: “minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum” (Galarnyk, 2018). The median is the middle value of a dataset. The first quartile is the middle number between the smallest number (not the “minimum”) and the median of the dataset. The third quartile is the middle value between the median and the highest value (not the “maximum”) of the dataset. Every value within 1.5 times the interquartile range is considered an outlier. Minimum is the lowest data point excluding any outliers. Maximum is the largest data point excluding any outliers.

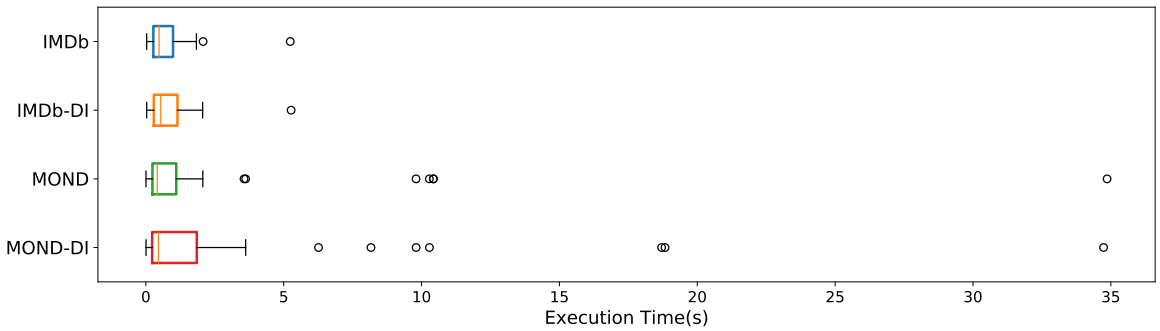

Figure 7.2: Time spent for obtaining Candidate Networks

The whiskers from a boxplot graph represents the range between interquartile and the values of maximum and minimum.

Overall, the results obtained for the third quartile are below 1.84 seconds, which indicates that for the majority of keyword queries, Lathe return the candidate queries in less than 1.84 seconds. Also, all query sets yielded a maximum execution time of 3.62 seconds.

In addition, we noticed several outlier queries, which requires an execution time up to 34.74 seconds. In case of the query sets for the IMDb dataset, the outliers are queries with 5 and 7 keywords, which reasonably might require more time. In the case of the query sets for the MONDIAL dataset, the majority of outliers comes from keyword queries involving the relation *country* twice. This relation has a total of 10 RICs, which directly affects the breadth-first search behavior of the candidate network generation algorithm.

7.4 Impact of Instance-based Pruning on CNs

In this experiment, we evaluated the ranking of candidate networks when using our proposed instance-based pruning techniques introduced in Section 6.4. These techniques which prunes the CNs whose SQL queries results are empty.

We compare three configurations for ranking and generating candidate networks. The first configuration does not perform any pruning technique and we call it the *CN-Std*. The second configuration uses the pos-pruning technique CN-Pos, which applies the pruning after the generation and ranking of CNs. The last configuration uses a the pre-pruning technique CN-Pre, applies the pruning during the generation of CNs and before the ranking of CNs.

We used the metrics MRR and P@K for K up to the 4 rank position to evaluate the quality of the ranking. Also, we consider only the top-10 query matches, where

each query match produces a single candidate network. The results obtained with each of configurations in the IMDb and MONDIAL datasets are respectively presented in Figures 7.3 and 7.4.

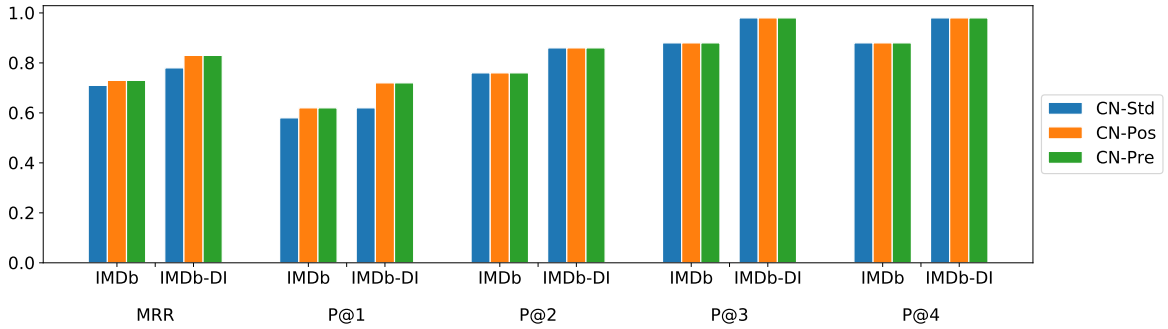

Figure 7.3: Ranking of Candidate Networks with Instance-based Pruning - IMDb

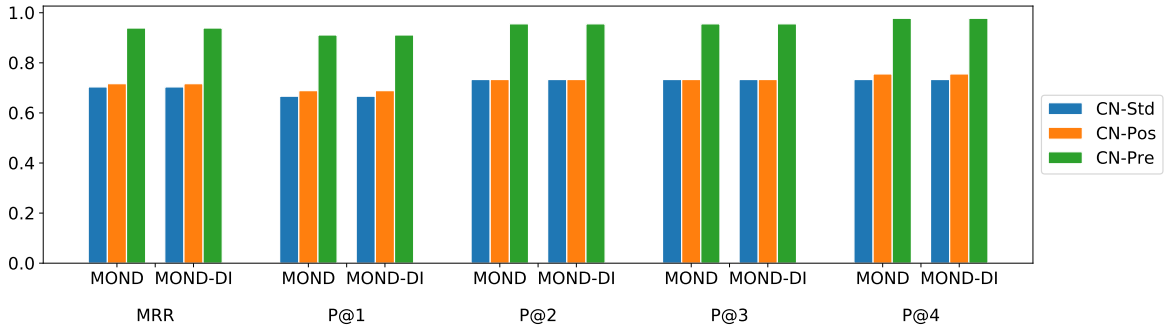

Figure 7.4: Ranking of Candidate Networks with Instance-based Pruning - MONDIAL

Overall, the CN-Std configuration obtained the worst results in both datasets, which indicates that adopting instance-based pruning techniques can help to improve the quality of ranking CNs.

The CN-Pos configuration obtained slightly better results than CN-Std, with an increase of 0.3 in MRR. Considering that CN-Pos only applies the pruning after the generation and ranking of CNs, this configuration is not able to generate the relevant CN if the relevant QM only produces CNs whose SQL queries results are empty.

The CN-Pre configuration obtained the best results overall. As that CN-Pre applies the pruning during the generation of the CNs, it ensures that the relevant QM will produce a CNs whose SQL queries results are not empty.

The large schema graph of MONDIAL database results in a large number of possible candidate networks for a query match. This might lead to QMs producing only CNs whose SQL queries results are empty. The high difference between the results

of CN-Pre and the others indicates for MONDIAL dataset indicates that this is the case.

We stress that, as commented in Section 6.4, that applying instance-based pruning techniques incur in a unavoidable increase in the generation time. This is due to the queries we need to issue to the DBMS. Techniques similar to those used in by SQL query optimizers can be used to mitigate this issues. Currently, we left this issue to be addressed as a future work.

7.5 Evaluation of Query Matches Ranking

In this experiment, we will evaluate the quality of the ranking of Query Matches according to the metrics MRR and $P@K$. Considering that Lathe generates CNs based on the top-K QMs, it is important to discover in which ranking position K we are able to maximize the quality of ranking CNs. That is, in which K the obtained results for $P@K$ reach their peak and stabilizes. Figure 7.5 presents the results obtained using the metric $P@K$ up to the 10th ranking position.

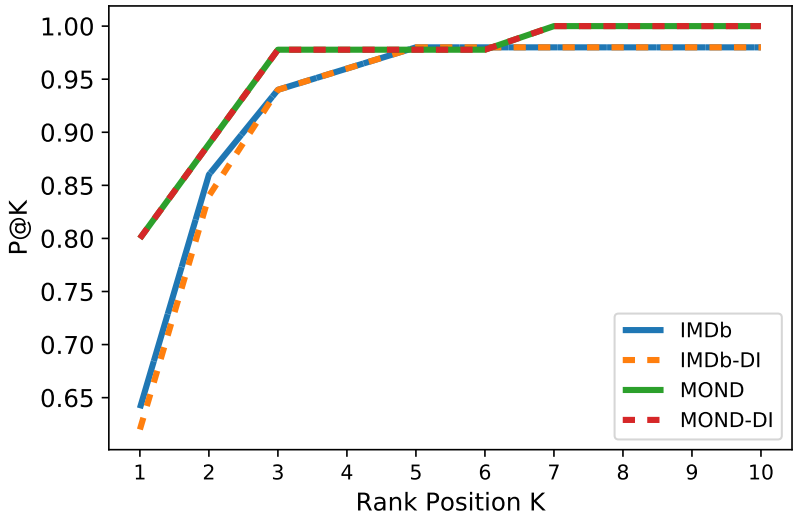

Figure 7.5: Ranking of Query Matches

Overall, the query sets results reached the their peak in P@7. This indicates that we generate the maximum number of relevant QMs in the top-7 query matches. Also, the P@7 of 0.98 obtained for the query sets IMDb and IMDb-DI indicates that these Lathe generated the relevant query match for 49 out of 50 keyword queries.

Chapter 8

Conclusions and Future Work

In this chapter, we review our main contributions, discuss the conclusions we have reached and we outline the future work.

8.1 Main Contributions

In this work, we presented Lathe, a novel method to for generating and ranking Candidate Networks considering queries with keywords that refer to schema elements with support to schema references.

We formally defined Keyword Matches, which associate keywords with schema elements or with sets of tuples whose attribute values contain these keywords.

We also adapted the definitions of Query Matches, Candidate Networks to comprise keyword matches. We adapted the algorithm for Query Match Generation, which not only combines the keyword matches, but also merges them whenever is possible.

We proposed an algorithm for Ranking Query Matches, which serves as an intermediate ranking for candidate networks.

We proposed a simple yet effective algorithm for Ranking Candidate Networks, which takes advantage of the ranking of query matches.

We proposed several Instance-based Pruning Techniques for generating Candidate Networks, such as pruning candidate networks whose SQL results are empty.

We presented an Evaluation of the Candidate Network Ranking, an Evaluation of the Query Matches Ranking, and also an Evaluation of Instance-based Pruning on the Generation and Ranking of Candidate Networks.

Finally, we presented a R-KwS method that can also be used as an underlying framework to different applications, which will be further explained in the next section.

8.2 Future Work

The work we developed raises a number of issues that can further investigated as future work. We list some of these issues below:

Natural Language Interfaces for Databases

Natural Language Interfaces for Databases (NLIDB) aim at enabling naive users to specify complex, *ad-hoc* query intent without training. Typically, these interfaces lack a proper support for mapping elements of the natural language query to the corresponding elements in the database. In our research group, we are currently investigating how our CN generation method can be used to improve this issue. The results we achieve so far are promising, and we will continue to work in this approach in the near future.

Dealing with Database Schema Changes

A common problem in software development is that of adjusting applications' code when attributes of the database are renamed, moved or remove across different schema versions. In our research group, we recently published a framework called LESSQL (Afonso et al., 2020), which used our CN generation method with support for schema references as an underlying framework that generates a corresponding SQL query for the current schema. We will continue to work in this approach in the near future.

Candidate Network Generation and the Steiner-Tree Problem

In attempt to further improve the efficiency of Candidate Network generation, we are currently investigating its relation with the well-known Steiner-Tree problem in graphs. We plan to take advantage of several efficient approximate algorithms that there exists for this problem in the literature and adapted them to the Candidate Generation Problem.

Experiment in more databases

We performed our experiments in two well-known datasets for Keyword Search over Databases. Our experiments showed promising results. We will also test our experiments in several different datasets.

Development of an API

We plan to develop an application programming interface of keyword search over relational databases for other fields of research. In fact, our method is currently being used underlying other two different applications.

Development of a DEMO

For better understanding of our method and to facilitate future comparisons with other methods. We will develop an online DEMO of our work. Also, we aim make our code, query sets, datasets and experiments publicly available.

Bibliography

Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Sudarshanxe, S., et al. (2002). Banks: Browsing and keyword searching in relational databases. Em *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pgs. 1083–1086. Elsevier.

Afonso, A., da Silva, A., Conte, T., Martins, P., Garcia, A., e Cavalcanti, J. (2020). Lessql: Dealing with database schema changes in continuous deployment. Em *Proceedings of the 27 IEEE SANER International Conference on Software Analysis, Evolution and Reengineering*.

Agrawal, S., Chaudhuri, S., e Das, G. (2002). Dbxplorer: A system for keyword-based search over relational databases. Em *Proceedings 18th International Conference on Data Engineering*, pgs. 5–16. IEEE.

Baeza-Yates, R. e Ribeiro-Neto, B. (2008). *Modern Information Retrieval: The Concepts and Technology Behind Search*. Addison-Wesley Publishing Company, USA, 2nd edition.

Baid, A., Rae, I., Li, J., Doan, A., e Naughton, J. (2010). Toward scalable keyword search over relational data. *Proceedings of the VLDB Endowment*, 3(1-2):140–149.

Bergamaschi, S., Domnori, E., Guerra, F., Trillo Lado, R., e Velegrakis, Y. (2011a). Keyword search over relational databases: a metadata approach. Em *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pgs. 565–576. ACM.

Bergamaschi, S., Guerra, F., Rota, S., e Velegrakis, Y. (2011b). A hidden markov model approach to keyword-based search over relational databases. Em *International Conference on Conceptual Modeling*, pgs. 411–420. Springer.

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., e Sudarshan, S. (2002). Keyword searching and browsing in databases using banks. Em *Proceedings 18th International Conference on Data Engineering*, pgs. 431–440. IEEE.

Coffman, J. e Weaver, A. C. (2010a). A framework for evaluating database keyword search strategies. Em *Proceedings of the 19th ACM international conference on Information and knowledge management*, pgs. 729–738. ACM.

Coffman, J. e Weaver, A. C. (2010b). Structured data retrieval using cover density ranking. Em *Proceedings of the 2nd International Workshop on Keyword Search on Structured Data*, pg. 1. ACM.

Coffman, J. e Weaver, A. C. (2012). An empirical performance evaluation of relational keyword search techniques. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):30–42.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2009). *Introduction to algorithms*. MIT press.

de Cristo, M. A. P., Calado, P. P., Da Silveira, M. d. L., Silva, I., Muntz, R., e Ribeiro-Neto, B. (2003). Bayesian belief networks for ir. *International Journal of Approximate Reasoning*, 34(2-3):163–179.

Galarnyk, M. (2018). Understanding boxplots. Technical report, Towards Data Science. Available from `https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51`.

He, H., Wang, H., Yang, J., e Yu, P. S. (2007). Blinks: ranked keyword searches on graphs. Em *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pgs. 305–316. ACM.

Hearne, T. e Wagner, C. (1973). Minimal covers of finite sets. *Discrete Mathematics*, 5(3):247–251.

Hristidis, V., Gravano, L., e Papakonstantinou, Y. (2003). Efficient ir-style keyword search over relational databases. Em *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pgs. 850–861. VLDB Endowment.

Hristidis, V. e Papakonstantinou, Y. (2002). Discover: Keyword search in relational databases. Em *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pgs. 670–681. Elsevier.

Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., e Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. Em *Proceedings of the 31st international conference on Very large data bases*, pgs. 505–516. VLDB Endowment.

Keselj, V. (2009). Speech and language processing daniel jurafsky and james h. martin (stanford university and university of colorado at boulder) pearson prentice hall, isbn 978-0-13-187321-6.

Liu, F., Yu, C., Meng, W., e Chowdhury, A. (2006). Effective keyword search in relational databases. Em *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pgs. 563–574. ACM.

Luo, Y., Lin, X., Wang, W., e Zhou, X. (2007). Spark: top-k keyword query in relational databases. Em *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pgs. 115–126. ACM.

May, W. (1999). Information extraction and integration with FLORID: The MONDIAL case study. Technical Report 131, Universität Freiburg, Institut für Informatik. Available from `http://dbis.informatik.uni-goettingen.de/Mondial`.

Mesquita, F., da Silva, A. S., de Moura, E. S., Calado, P., e Laender, A. H. (2007). Labrador: Efficiently publishing relational databases on the web by using keyword-based query interfaces. *Information Processing & Management*, 43(4):983–1004.

Miller, G. A. (1998). *WordNet: An electronic lexical database*. MIT press.

Oliveira, P., da Silva, A., e de Moura, E. (2015). Ranking candidate networks of relations to improve keyword search over relational databases. Em *2015 IEEE 31st International Conference on Data Engineering*, pgs. 399–410. IEEE.

Oliveira, P., da Silva, A., de Moura, E., e Rodrigues, R. (2018). Match-based candidate network generation for keyword queries over relational databases. Em *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pgs. 1344–1347. IEEE.

Pearl, J. (2014). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier.

Pedersen, T., Patwardhan, S., e Michelizzi, J. (2004). Wordnet:: Similarity: measuring the relatedness of concepts. Em *Demonstration papers at HLT-NAACL 2004*, pgs. 38–41. Association for Computational Linguistics.

Ribeiro, B. A. e Muntz, R. (1996). A belief network model for ir. Em *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pgs. 253–260. Citeseer.

Salton, G. e Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523.

Tata, S. e Lohman, G. M. (2008). Sqak: doing more with keywords. Em *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pgs. 889–902. ACM.

Wu, Z. e Palmer, M. (1994). Verbs semantics and lexical selection. Em *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pgs. 133–138. Association for Computational Linguistics.

Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3):372–390.