Ranking Candidate Networks of Relations to Improve Keyword Search over Relational Databases

Pericles de Oliveira, Altigran da Silva, Edleno de Moura
Instituto de Computação – Universidade Federal do Amazonas
Manaus, Brazil
{pericles,alti,edleno}@icomp.ufam.edu.br

***Abstract*—Relational keyword search (R-KwS) systems based on schema graphs take the keywords from the input query, find the tuples and tables where these keywords occur and look for ways to "connect" these keywords using information on referential integrity constraints, i.e., key/foreign key pairs. The result is a number of expressions, called Candidate Networks (CNs), which join relations where keywords occur in a meaningful way. These CNs are then evaluated, resulting in a number of join networks of tuples (JNTs) that are presented to the user as ranked answers to the query. As the number of CNs is potentially very high, handling them is very demanding, both in terms of time and resources, so that, for certain queries, current systems may take too long to produce answers, and for others they may even fail to return results (e.g., by exhausting memory). Moreover, the quality of the CN evaluation may be compromised when a large number of CNs is processed. Based on observations made by other researchers and in our own findings on representative workloads, we argue that, although the number of possible Candidate Networks can be very high, only very few of them produce answers relevant to the user and are indeed worth processing. Thus, R-KwS systems can greatly benefit from methods for accessing the relevance of Candidate Networks, so that only those deemed relevant might be evaluated. We propose in this paper an approach for ranking CNs, based on their probability of producing relevant answers to the user. This relevance is estimated based on the current state of the underlying database using a probabilistic Bayesian model we have developed. Experiments that we performed indicate that this model is able to assign the relevant CNs among the top-4 in the ranking produced. In these experiments we also observed that processing only a few relevant CNs has a considerable positive impact, not only on the performance of processing keyword queries, but also on the quality of the results obtained.**

I. Introduction

Systems that process keyword-based queries over relational databases, commonly called *R-KwS* systems, aim at allowing naive/informal users to retrieve information from relational databases without any knowledge about schema details and query languages. These systems face the task of automatically determining, from a handful of keywords without explicit structure, which pieces of information to retrieve from the database and how these pieces can be combined to provide relevant answers to the user. This is quite challenging, since the information sought often spans multiple tables and tuples, but queries posed by users do not provide any hints as to their location on the elements of the database schema. In the last decade, many researchers have proposed methods to enable R-KwS. These methods often address one of two distinct types of R-KwS systems: those based on *Data Graphs* and those based on *Schema Graphs*. Systems in the first category are based on structures called *Data Graph*, whose nodes represent tuples associated with the keywords they contain, and edges connect these tuples based on referential integrity constraints. In these systems, results of keyword queries are computed by finding subtrees in a data graph that minimizes the distance between nodes matching the keywords from the input query. Examples of methods and systems in this category are BANKS [1], Bi-directional [2], BLINKS [3] and Effective [4]. As data graphs do not represent schema information, they can be also used with other data models besides the relational model. Systems in the second category are based on the concept of *Candidate Networks (CNs)*, which are networks of joined database relations used to generate SQL queries whose results provide an answer to the input keyword query. This approach was proposed in DISCOVER [5] and DBXplorer [6], and was later adopted by a number of other systems, such as Efficient [7], SPARK [8], CD [9], Min-cost [10], S-KwS [11] and KwS-F [12]. Systems in this category take advantage of the basic functionality of the underlying RDBMS by producing appropriate SQL joint queries to retrieve answers relevant to keyword queries posed by users.

The focus of this paper is on systems from this second category. In general, R-KwS systems based on Schema Graphs take the keywords from the input query, find the tuples and tables where these keywords occur and look for ways to "connect" these keywords using information of referential integrity constraints, i.e., key/foreign key pairs, provided in the database schema. The result is a number of expressions that join relations where keywords occur in a meaningful way. These join expressions are the Candidate Networks (CNs). The generated CNs are then *evaluated*, resulting in a number of joint networks of tuples (JNTs) presented to the user as answers to the query. As there can be many JNTs, they are often ranked by their relevance to fulfil the user needs.

Depending on the query and the database, there can be a large number of Candidate Networks. For instance, the experimental query workload we use in this paper includes queries from which a well-known CN generation algorithm [5] obtains hundreds of CNs. Processing a large number of CNs, is of course, time-demanding and resource-consuming. In fact, it is known that, for certain queries, current systems can take too long to provide answers, and for others they may even fail to return results (e.g., by exhausting memory). Moreover, the quality of the answers produced by CN evaluation and JNT ranking algorithms may be compromised when a large number of CNs is processed.

We propose that, although the number of possible Candidate Networks can be very high, only very few of them produce answers relevant to the user and are indeed worth processing. This claim is in line with observations made by other researchers who found that the number of relevant answers to keyword queries is often very small and that, in many cases, there is only one relevant answer to return [8], [12]–[14]. It follows that, if only a few answers are relevant, then only a few CNs need to be evaluated to produce them. We also observed this trend in queries of different workloads we used in experiments we carried out and report in this paper. This is significant, since these workloads have been proposed and used in previous studies on R-KwS in the literature. In fact, we verified that, in all queries in these workloads, no more than two Candidate Networks are needed to produce relevant answers. This is a drastic reduction, if we consider that the number of CNs generated often ranges from tens to many hundreds. Making this claim explicit and showing experimental data to support it is the first contribution we made in this paper.

An implication of this claim is the need for methods to assess the relevance of Candidate Networks, so that only those deemed relevant might be evaluated. With this goal in mind, we propose in this paper an approach for ranking Candidate Networks, based on their probability of producing relevant answers to the user. Specifically, we present 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 Network so that only a few CNs with the highest scores are evaluated. In addition, we also show how this ranking process can be carried out efficiently using a simple inverted index. This approach, the model and the ranking algorithm comprise the second contribution we offer in this paper.

Using the proposed approach, we performed a comprehensive set of experiments using query workloads also used in R-KwS experiments previously presented in the literature. By comparing our results with those obtained with other representative methods on the same tasks, we could observe that our approach had a considerable positive impact, not only on the performance of processing keyword queries, but also on the quality of the answers produced by CN evaluation and JNT ranking algorithms. For instance, when we coupled our CN ranking algorithm with two well-known CN evaluation algorithms, namely, Hybrid [7] and Skyline Sweeping [8], the results they deliver were twice as precise, according to widely-accepted metrics, in compared with the results they provide without our algorithm. As these evaluation algorithms received less CNs to process, they also run much faster. In addition, we have experimentally shown that our ranking model is very precise: for all the queries we tested, it was able to place the relevant CNs among the top-4 in the ranking produced. Showing experimental evidences of the impact of our approach in the performance and the quality of the answers produced by R-KwS systems is our third contribution in this paper.

The remainder of the paper is organized as follows. Section II discusses the problems of generating and evaluating Candidate Networks in R-KwS systems. It also reviews the related literature, notation and terminology used in this field. Section III overviews the process of ranking CNs and shows how it fits in the typical architecture of R-KwS systems. In Section IV, we present the details of our ranking model and introduce our ranking algorithm. Section V reports the results of experiments we conducted to show the effectiveness of our ranking model and its impact on the performance and the quality of typical R-KwS systems. Finally, Section VI presents conclusions we have reached and outlines some directions for future work.

II. Background and Related Work

A. Schema Graph-Based R-KwS

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. Thus, we focus on systems adopting such a model throughout the paper. Fig. 1 presents the main architecture and functioning of a typical R-KwS System based on schema graphs, which we describe below.

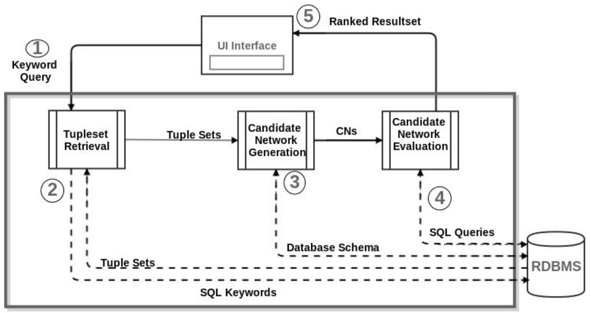


Fig. 1. A typical Schema Graph R-KwS System.

Initially, an unstructured keyword query is submitted by an user, typically using a single text box (1). A keyword query is posed by a user who searches for interconnected tuples that contain the given keywords. A tuple contains a keyword if a text attribute of the tuple contains the keyword. With the keywords in hand, the system looks for subsets of relations that contain the keywords from the queries. These subsets, called *Tuple Sets*, are then retrieved from the database (2); Next, these tuple sets are used to generate *Candidate Networks* [5], [6] (CNs) (3). CNs are relational algebra expressions that join relations whose tuples contain the keywords beign sought. In other words, each CN describes how to produce potential answers to the keyword query entered. Besides the tuple sets generated in the previous step, generating CNs also requires information on referential integrity constraints taken from the database schema. Since there are many different ways of joining relations that store the tuples containing the keywords, many different CNs can potentially be generated. In practice, however, only a few of them are indeed useful to produce plausible answers. The most well-known algorithm for generating CNs, called *CNGen*, have been proposed in [5]. In the next step (4), the CNs generated are evaluated in order to provide answers from the database to fulfil the input query. In this context, answers are *Joining Networks of Tuples* (JNTs) [5], that is, trees composed of joined tuples that either contain the keywords or associate tuples that contain keywords. Many different algorithms have been proposed to evaluate CNs [5], [6], [11]. In particular, in state-of-the-art systems [7]–[9], only top-K JNTs are retrieved, which requires them to be ranked using IR style score functions. The top-K answers are then presented to the user (5).

The main motivation for ranking JNTs in systems like Efficient [7], SPARK [8], and CD [9] to avoid the multi-query optimization problems arising when all CNs are to be evaluated, as occurs in DISCOVER [5]. The efficiency and scalability problem in CN evaluation is addressed in a different way in KwS-F [12]. Its 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. In [12], the authors report a number of experimental findings on the effectiveness and feasibility of the proposed approach, yet, unfortunately, they did not report on the quality of the results obtained.

In [11], the authors present a strategy for ordering the internal nodes of the CNs generated by CNGen aiming at detecting possible duplicate CNs. This can reduce the number of CNs handled by the systems, but still requires that all CNs, duplicated or not, be generated.

In [13], Coffman et. al. proposed a framework for evaluating R-KwS systems and reported the results of applying this framework over three representative standardized datasets they constructed, namely *Mondial*, *IMDb* and *Wikipedia*, along with respective query workloads. The authors compare 9 state-of-the-art R-KwS systems, evaluating them in many aspects related to their effectiveness and performance. An important finding they report is that, in terms of effectiveness, no single system they tested performed best across all datasets/query sets. So far, this is the only study in the literature to provide qualitative evaluations of R-KwS systems. In our experimental evaluation we use datasets, query sets and results provided in this paper.

B. Concepts and Terminolgy

We rely on the concepts and terminology introduced in [5]. In particular, we use the definitions of *Joining Network of Tuples*, *Minimal Total Joining Network of Tuples*, *Tuple Sets*, *Joining Network of Tuple Sets* and *Candidate Network* . For the sake of convenience, some of these definitions are stated below with slight modifications with respect to [5].

As in [5], consider a schema graph G that represents a relational schema where edges correspond to relations and arcs correspond to referential integrity constraints between relations. For the discussion that follows, the directions of referential integrity constraints is not important, so we consider an undirected version G_u of G[1].

Definition 1: A **Joining Network of Tuples (JNT)** j is a tree of tuples where for each pair of adjacent tuples t_i, $t_j \in j$, where t_i and t_j are tuples of relations R_i and R_j, respectively, there is an edge $\langle R_i, R_j \rangle$ in G_u and $(t_i \bowtie t_j) \in (R_i \bowtie R_j)$.

Definition 2: Given a set $W = \{w_1, \ldots, w_n\}$ of keywords, a JNT j is a **Minimal Total JNT (MTJNT)** for W if it is both **total**, that is, every keyword w_i is contained in at least one tuple of j, and **minimal**, that is, any JNT j' that results from removing any tuple from j is not total.

Definition 3: A **Keyword Query** is a list Q of keywords whose result is either: (i) the set T of tuples from some relation R, such that each of its tuples contains all keywords from Q; or (ii) the set M of all possible MTJNT for the keywords in Q over some set of relations $\{R_1, \ldots, R_m\}$.

Definition 4: Let Q be the set of keywords in a keyword query and let K be a subset of Q. Also, let R_i be a relation. A **Tuple Set** from R_i over K is given by

$$R_i^K = \{t | t \in R_i \wedge \forall k \in K, k \in \mathcal{W}(t) \wedge \forall \ell \in Q - K, \ell \notin \mathcal{W}(t)\},$$

where $\mathcal{W}(t)$ gives the set of terms (words) in t. If $K = \emptyset$, the tuple set is said to be a *free tuple set* and it is denoted by $R_i^{\{\}}$.

According to Definition 4, R_i^K contains the tuples of R_i that contain all terms of K and no other terms from Q.

Definition 5: Given a set of keyword query $Q = \{k_1, \ldots, k_n\}$, a **Candidate Network** C is a tree of tuple sets, where for each pair of adjacent tuple sets R_i^K , R_j^M in C there is an an edge $\langle R_i, R_j \rangle$ in G_u. Further more, C must be *total*, that is, each keyword in the Q must be contained in at least one tuple set of C, and *minimal*, that is C is not total if any tuple set is removed.

The procedure for generating CNs for a given keyword query consists of first finding the non-free tuple sets that contain the keywords of the query and then traversing the schema graph trying to "connect" these tuple sets. Each CN is thus a tree in which leaves are non-free tuple sets and internal nodes are free tuple sets connecting them. The best know algorithm for generating CN is CNGen, proposed in [5], to which we refer the reader for details on the generation procedure.

III. Ranking CNs in R-KwS Systems

Each Candidate Network represents a distinct way of arranging the keywords of a given query among the relations of the database, so that each keyword is present in at least one tuple of the answer, and, that these tuples are joined according to the logical constraints of the database to form a meaningful answer. Depending on such factors as the number of keywords, the size of the database and the distribution of keywords among the tuples, the expected number of Candidate Networks that

[1]Without loss of generality, we also assume the same simplifications as [5]; that is, the attributes involving referential integrity constraints have the same name; there are no self loops or parallel edges in the schema graph, and no set of attributes of any relation is both a primary key and a foreign key for two other relations

can be derived from a query is quite high. If this is the case, the evaluation of the generated CNs may be very demanding in terms of time and resources needed. Indeed, it is a known fact that current R-KwS systems can take too long to deliver answers for certain queries, and for others they may even fail to return results (e.g., by exhausting memory).

However, a fact that has been overlooked in the literature so far is that just a few of theses many CN indeed lead to the correct answer sought by the user, i.e., tuple networks that fulfill users' informational needs. For instance, consider the query "gangster denzel washington". For this query, the CNGen algorithm [5] generates about 50 CNs, considering the IMDB dataset we used in our experiments. However, there is just a single CN that returns the answer, a movie entitled "American Gangster" starred by the famous actor Denzel Washington. This CN is the one which assigns the keyword "gangster" to the relation that stores movie titles and the pair of keywords "denzel" and "washington" to the relation that stores person names. This CN also includes a relation that "connects" these two relations through referential integrity constraints.

As another example, in the Coffman query set [13], 55 out of the 150 proposed queries have a single network of tuples as the correct answer. As pointed out in [14], keyword queries returning one single answer as results are very common. As a consequence, queries for which just one single CN is relevant are also very common.

It follows that being able to select a few plausible CNs and discard a large number of spurious ones is important for saving resources needed to process CNs and to improve the quality of the results presented to users.

Our motivation for this work is, thus, developing a method to select a few relevant CN from the many other possible CNs that may exist. By doing so, we want to ultimately achieve two goals. First, we want to save time and resources that would otherwise be spent evaluating many CNs. Second, we want to improve the final results produced by CN evaluation methods by providing them only with CNs that are likely to produce relevant results.

The hypothesis that in fact only a few CNs produce relevant answers was verified in a workload of several queries coming from many distinct sources used in our experiments. Our empirical findings are reported along with our experimental results in Section V. Thus, while we will provide evidence to better support this hypothesis in Section V, we will assume it to be the case for the discussion henceforth, and, ultimately, throughout the paper.

We argue herein that the selection of relevant CNs can be carried out by *ranking* the CNs generated and by selecting just a small subset of the top-ranked CNs to be effectively evaluated. Specifically, the ranking process we propose is based on the estimated likelihood that a CN assigns each keyword to the correct attribute from the database. In our example, the CN that assigns assigns the keyword "gangster" to the atribute representing movie titles and the pair of keywords "denzel" and "washington" to the atribute representing person names must be estimated as the most likely, also considering that these atributes can be joined according to the logical design of the database.

The algorithm we proposed for this, *CNRank*, uses a Bayesian network for assigning a score to each CN generated, which corresponds to the joint probability of all attribute-term assignments described in the CN, considering the current state of the database and the join operations used in the CN.

Based on this idea, we propose a new alternative architecture for R-KwS systems, as shown in Fig. 2 and described below.

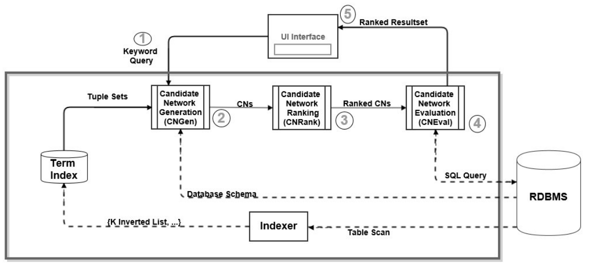


Fig. 2. Including CN Ranking in R-KwS Systems

This architecture involves five main steps and one pre-processing step, as illustrated in Fig. 2. The first two steps are similar to those in typical Schema Graph R-KwS systems (see Fig. 1). That is, in the first step (1), the user inputs an unstructured keyword query. Then (2), the system generates a number of Candidate Networks.

In the next step (3), the CNs generated are ranked using the algorithm *CNRank*, which is presented in Section IV. Next, only a few top CNs from the ranking are evaluated (4). The evaluation process can be carried out by any of the many algorithms proposed for this purpose in the literature; for instance, [5]–[10]. Finally, the ranked tuple networks are presented to the user (5).

The processes of ranking CNs is supported by an inverted index called *Term Index*. In this index, each entry is a term, that is, a word, found in the values of the attributes of the relations over which keyword queries will be issued. To each term t_k, the index associates a list whose elements are pairs of the form $\langle A_j, f_{kj} \rangle$,where A_j is an attribute in a relation, in whose values the term t_k occurs with frequency f_{kj}. The term index is constructed in a preprocessing step that scans once all the tables over which queries will be used. This step precedes the processing of queries, and we assume that it does not need to be repeated often. Under this assumption, CNs are ranked for all queries without further interaction with the DBMS. In experiments that we performed with databases used in previous R-KwS papers in the literature, it was possible to store the term index entirely in the main memory.

While in current systems all CNs generated must be evaluated, in this architecture only a few top CNs in the ranking are evaluated. The experiments we performed and report here show that our CNRank algorithm is higly effective in the task of placing the best CNs, that is, those that fulfill the user's intention, in top positions of the ranking. These experiments also show that the results achieved when evaluating just a few top CNs are at least as good as those obtained with traditional systems that evaluate all CNs. On the other hand, the overall system performance is considerably improved by introducing CNRank between the generation and evaluation of CNs. For our experimental evaluation we instantiated this architecture with implementations of well-known algorithms for generating and evaluating CNs.

Note that our approach resembles the one adopted by LABRADOR [15]. However that system cannot deal with networks of relations such as CNs. It is limited in dealing with relations that are directly connected by natural joins.

We now present details on our CN ranking algorithm.

IV. CNRank

Given a set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of Candidate Networks generated for a keyword query U, we want to assign to each CN a score value that estimates the likelihood of this CN representing the user intention when formulating U.

Consider a CN $C = \langle \mathcal{R}, \mathcal{E} \rangle$, where $\mathcal{R}=\{R_1^{K_1},\ldots,R_n^{K_n}\}$ $(n>0)$ is its set of tuple sets and $E=\{\langle R_a^{K_a}, R_b^{K_b}\rangle |$ there is a join between $R_a^{K_a}$ and $R_b^{K_b}$ in $C\}$ $(1 \leq a, b \leq n$ and $a \neq b)$.

In our work, the score of C is computed as the joint probability of the keywords in each K_j compose of values of some attribute of R_j, considering the current state of the database. The computed CN scores are then used to rank the CNs based on the belief that they correctly represent the keyword query posed by the user. This process is called *Candidate Network Ranking*.

To estimate this joint probability, the individual probabilities involving the keywords K_j and the relation R_j are combined by using a Bayesian network model [16] we will present later. We first need to introduce an alternative algebraic representation for Candidate Networks.

A. Algebraic Representation of CNs

We recall from [5] that there are two types of tuple sets. Let $R_j^{K_j}$ be a tuple set. If $K_j \neq \{\}$, it is called a *non-free* tuple set, and is composed of a subset of the tuples from R_j that contain a subset K_j of the keywords in the input query. On the other hand, if $K_j = \{\}$, then we have a *free* tuple set, and it contains all tuples from relation R_j. Intuitively, the role of *free* tuple sets in a CN is to "connect" non-free tuple sets.

Now, consider a CN C_i defined as above. In such a CN, any tuple set $R_j^{K_j}$ can be represented by a relational algebra expression of the form $\sigma_{\alpha_j}(R_j)$, where α_j is a predicate. The form of α_j depends on the type of the tuple set. If $K_j \neq \{\}$, that is, if $R_j^{K_j}$ is a non-free tuple set, then α_j is a predicate of the form $A_j \supseteq K_j$, where A_j is an attribute of R_j. This predicate is true for tuples of R_j in which the value of A_j contains all terms of K_j. Otherwise, if $K_j = \{\}$, that is, if $R_j^{K_j}$ is a free-tuple set, then α_j is a tautology. This means that all tuples from R_j will be included in the tuple set.

Likewise, we can use a algebraic representation of Candidate Networks using a relational algebra expression of the form:

$$\sigma_{\alpha_1}(R_1) \underset{\Theta_{1,2}}{\bowtie} \sigma_{\alpha_2}(R_2) \underset{\Theta_{1,3}}{\bowtie} \ldots \underset{\Theta_{n-1,n}}{\bowtie} \sigma_{\alpha_n}(R_n)$$

where each $\Theta_{i,i+1}$ $(1 \leq i \leq n-1)$ is a join condition that implements the edge $\langle R_i^{K_i}, R_{i+1}^{K_{i+1}} \rangle$ from the CN.

For convenience, we apply a simple algebraic transformation over the expression above as follows:

$$\underset{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\sigma} (R_1 \underset{\Theta_{1,2}}{\bowtie} R_2 \underset{\Theta_{2,3}}{\bowtie} \ldots \underset{\Theta_{n-1,n}}{\bowtie} R_n)$$

Thus, we can represent a CN C as a pair $C = \langle T, P \rangle$, where

$$T = R_1 \underset{\Theta_{1,2}}{\bowtie} R_2 \underset{\Theta_{2,3}}{\bowtie} \ldots \underset{\Theta_{n-1,n}}{\bowtie} R_n \text{ and } P = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$$

where T is called the *base relation* of the CN and P is its set of predicates. Alternatively, this CN can also be represented by $C = \langle T, P' \rangle$, where in P', the tautologies from P are removed.

Example 1: The following example is based on the IMDb database available in [17], whose schema graph is presented in Fig. 3[2]. Relations are represented by rectangles labelled with their respective schema definitions. Referential integrity constraints are represented by arcs with keys/foreign keys as labels in arc ends.

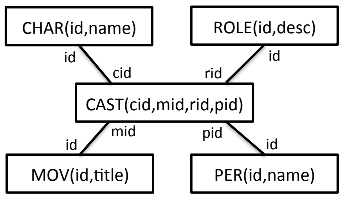


Fig. 3. Schema Graph for the IMDb database.

Consider a keyword query $U = \{$*denzel*,*washington*, *gangster*$\}$. For a given database instance, a possible Candidate Network for this query is given by:

$$\underset{\text{CHAR.name} \supseteq \{\textit{gangster}\} \wedge \text{PER.name} \supseteq \{\textit{denzel,washington}\}}{\sigma} [(\text{CHAR} \underset{\text{id=cid}}{\bowtie} \text{CAST}) \underset{\text{pid=id}}{\bowtie} \text{PER}\,]$$

where the base relation is $\text{CHAR} \underset{\text{id=cid}}{\bowtie} \text{CAST} \underset{\text{pid=id}}{\bowtie} \text{PER}$ and there are two predicates: $\text{CHAR.name} \supseteq \{\textit{gangster}\}$ and $\text{PER.name} \supseteq \{\textit{denzel,washington}\}$

B. Probabilistic Ranking Model

Given the alternative representation of CNs introduced above, we can describe the Bayesian network model [16] we use to compute the score of a Candidate Network. Such a model is illustrated in Fig. 4. Notice that although the figure represents a single CN, it can be easily expanded to many queries.

In a Bayesian network, each node represents a piece of information. At the top of Fig. 4, node C is a Candidate Network and $\alpha_1,\ldots,\alpha_m$ are its predicates. At the bottom, labeled "Database Side", nodes represent the current state of the relations, i.e., the tuple sets, involved in the CN. Specifically, nodes $B_1,\ldots,B_m$ represent the attributes of these relations and T represents base relation of C that joins these relations. Each node b_{ij} represents a term found in the values of B_i in the current database state and $\overrightarrow{b_i}$ is a vector containing all terms

[2]Names of relation and attributes were changed for convenience

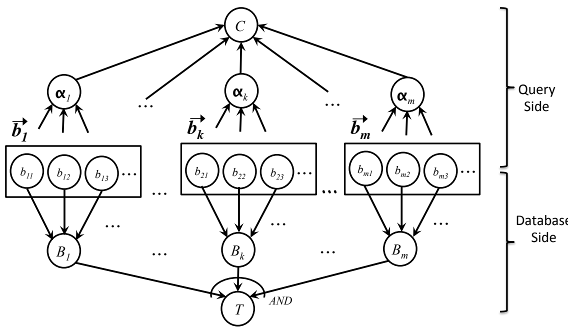


Fig. 4. Bayesian network Model for ranking CNs.

found in the values of B_i. For simplicity, only attributes that use the predicates are illustrated in this figure.

Example 2: To illustrate these nodes and their roles, in Fig. 5, we show an instance of the Bayesian network for the CN of Example 1. In this case, the Candidate Network side

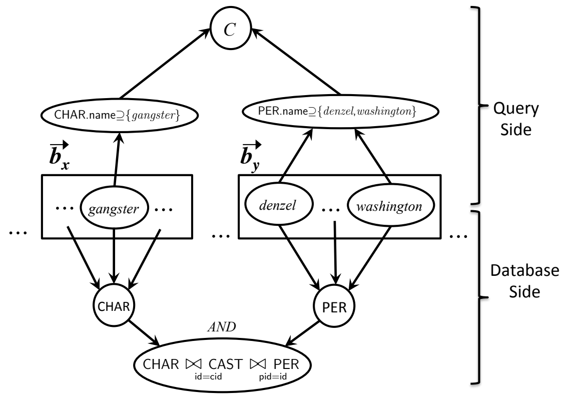


Fig. 5. Example of a Bayesian network for ranking CNs.

would have only two nodes to represent the two predicates of this CN. The first predicate refers to the term "gangster" found in the values of CHAR.name and the second predicate refers to the pair of terms "denzel" and "washington" found in the values of PER.name.

On the database side, we have the attributes name from relation CHAR and name from relation PER joined by the base relation of the CN. Then, vector $\overrightarrow{b_x}$ will have a node corresponding to term "gangster" and vector $\overrightarrow{b_y}$ will have a node corresponding to each term "denzel" and "washington". We would also have arcs from this term-related nodes to the nodes representing the appropriate predicates.

Each node of the network is associated to a binary variable, which takes the value 1 to indicate that the corresponding information will be considered for the ranking computation. In this case, we say that the information was *observed*. The likelihood of a Candidate Network C, given the current state of the base relation T, can be seen as the probability of observing C, given that relation T was observed. This is represented by conditional probability $P(C|T)$.

Analyzing the Bayesian network in Fig. 4, we can derive the following general equation:

$$P(C|T) = \mu \times \sum_{\overrightarrow{b_1},\ldots,\overrightarrow{b_m}} \Big(P(C|\overrightarrow{b_1},\ldots,\overrightarrow{b_m}) \times P(T|\overrightarrow{b_1},\ldots,\overrightarrow{b_m}) \times P(\overrightarrow{b_1},\ldots,\overrightarrow{b_m}) \Big) \quad (1)$$

where μ is a normalizing constant.

The probability $P(C|\overrightarrow{b_1},\ldots,\overrightarrow{b_m})$ of observing the CN C given the state of all attributes it uses, can be expanded as:

$$P(C|\overrightarrow{b_1},\ldots,\overrightarrow{b_m}) = \sum_{\alpha_1,\ldots,\alpha_m} P(C|\alpha_1,\ldots,\alpha_m) \times P(\alpha_1|\overrightarrow{b_1}) \times \ldots \times P(\alpha_m|\overrightarrow{b_m}) \quad (2)$$

In Equation 2, the term $P(C|\alpha_1,\ldots,\alpha_m)$ corresponds to the probability of observing C given the predicate nodes α_1 to α_m. As we consider that the CN C should be active only when all its predicate nodes are also active, this translates to the following equation:

$$P(C|\alpha_1,\ldots,\alpha_m) = \begin{cases} 1 & \text{if } \alpha_1 = 1 \wedge \ldots \wedge \alpha_m = 1 \\ 0 & \text{otherwise} \end{cases}$$

Still in Equation 2, the terms $P(\alpha_1|\overrightarrow{b_1})$ to $P(\alpha_m|\overrightarrow{b_m})$, which correspond to the probability of observing the nodes α_1 and α_m given the state of the attributes B_1 to B_m, respectively, ensure that the active terms in $\overrightarrow{b_j}$ exactly match the terms in the predicate nodes α_j. This translates to the following equation:

$$P(\alpha_j|\overrightarrow{b_j}) = \begin{cases} 1 & \text{if } \forall k, g_k(\overrightarrow{b_j}) = 1 \text{ iff } t_{jk} \text{ occurs in } K_j \\ 0 & \text{otherwise} \end{cases}$$

where $g_k(\overrightarrow{b_i})$ gives the value of the k-th component $\overrightarrow{b_i}$ and t_{jk} is the kth term in B_j, considering $\alpha_j = B_j \supseteq K_j$.

The sum in Equation 1 takes into account all sets of possible active terms in vectors $\overrightarrow{b_1}$ to $\overrightarrow{b_m}$. However, the definition of the probability $P(C|\overrightarrow{b_1},\ldots,\overrightarrow{b_n})$ annuls the probability of any set of active terms, except the set in which the active terms exactly match the query terms referring to attributes B_1 to B_n, respectively. Thus, we can simplify Equation 1 to:

$$P(C|T) = \mu \times P(T|\overrightarrow{b_1},\ldots,\overrightarrow{b_m}) \times P(\overrightarrow{b_1},\ldots,\overrightarrow{b_m}) \quad (3)$$

where the active terms in vectors $\overrightarrow{b_1}$ to $\overrightarrow{b_m}$ are exactly those present in the predicates of C.

In a similar way, we can expand the probability of observing the base relation T given the state of all its attributes $P(T|\overrightarrow{b_1},\ldots,\rightarrow b_m)$ using the probabilities $P(T|B_1,\ldots,B_m)$, $P(B_1|\overrightarrow{b_1})$, $P(B_2|\overrightarrow{b_2})$, etc. Given this and the fact that vectors $\overrightarrow{b_j}$ are independent, Equation 1 can be rewritten as:

$$P(C|T) = \alpha \times P(B_1|\overrightarrow{b_1}) \times \ldots \times P(B_m|\overrightarrow{b_m}) \times P(T|B_1,\ldots,B_m) \times P(\overrightarrow{b_1}) \times \ldots \times P(\overrightarrow{b_m}) \quad (4)$$

where $\overrightarrow{b_j}$ is the state where only the query terms in C referring to attributes B_j are active.

We can now use the properties of the Candidate Networks to solve the conditional probabilities appearing in Equation 4, namely $P(B_j|\overrightarrow{b_j})$, $P(\overrightarrow{b_j})$ and $P(T|B_1,\ldots,B_m)$. Since there is no preference for any particular set of terms, the probability of vector $\overrightarrow{b_j}$ is defined as a constant:

$$P(\overrightarrow{b_j}) = \frac{1}{2^{N_j}}$$

where N_j is the total number of distinct terms that occur for values of the attribute B_j. Notice that N_j is taken from the original table where B_j belongs instead of the base relation.

The remaining probabilities are computed using a model we refer to as TF-IAF, which adapts the traditional vector space model [18] to the context of relational databases, similarly as done in [15]. In TF-IAF, the cosine measure is used to estimate the probability of observing the attribute B_j in the database, given the terms indicated by $\overrightarrow{b_j}$. The probability of observing B_j is defined as the cosine of the angle between a vector $\overrightarrow{B_j}$ which represents the values of the attribute B_j stored in the database (also considering its original relation), and vector $\overrightarrow{b_j}$ which represents the values for attribute B_j in the query, i.e.,

$$P(B_j|\overrightarrow{b_j}) = cos(\overrightarrow{B_j},\overrightarrow{b_j}) = \frac{\sum_{t_k \in \overrightarrow{b_j}} w_{jk} \cdot g_k(\overrightarrow{b_j})}{\sqrt{\sum_{t_k \in \overrightarrow{B_j}} w_{jk}^2} \times \sqrt{\sum_{t_k \in \overrightarrow{b_j}} g_k(\overrightarrow{b_j})^2}} \tag{5}$$

where w_{jk} is the weight of the term t_k in attribute B_j. To compute the weights, we use the concepts of *term frequency* (*TF*) and *inverse attribute frequency* (*IAF*) [15].

TF measures the frequency of a given term into the values of an attribute considering all tuples of the relation where it occurs. This is computed by the following formula:

$$tf = \frac{log(1+f_{kj})}{log(1+N_j)} \tag{6}$$

where f_{kj} is the number of occurrences of term k in the attribute B_j and N_j is the total number of distinct terms that occurs in the values of attribute B_j.

IAF is an adaptation of the concept of inverse document frequency (IDF) found in the context of information retrieval [18]. Here, it measures how infrequent the term is among the values of the attributes according to the following formula:

$$iaf = log\left(1 + \frac{N_a}{C_k}\right) \tag{7}$$

where N_a is the total number of attributes in the database and C_k is the number of attributes in whose values the term k occurs. We use IAF as an estimation of the degree of ambiguity of the term with respect to the attributes in the database.

For instance, consider the relation instance shown in Fig. 6. In this table, the term *Washington* is considered ambiguous, since it occurs for both attributes Person and Movie. On the other hand, the term *Godzilla* is typical to the Title attribute, and thus it is unambiguous.

Person	Movie
Albert Finney	*Washington Square*
Kerry Washington	*Against the Ropes*
Juliette Binoche	*Godzilla*
Denzel Washington	*Hurricane*

Fig. 6. A sample relation for illustrating the TF-IAF model.

The weights according to the TF-IAF model are computed by the following formula:

$$w_{jk} = tf \times iaf = \frac{log(1+f_{kj})}{log(1+N_j)} \times log\left(1 + \frac{N_a}{C_k}\right) \tag{8}$$

where f_{kj} is the number of occurrences of the term k in the attribute B_j, N_j is the total number of distinct terms that occur in the values of the attribute B_j, N_a is the total number of attributes in the database and C_k is the number of attributes in whose values the term k occurs.

Final Ranking Equation

Continuing the expansion of the components in Equation 4, since T may be be the result of joining several different relations, we define the probability $P(T|B_1,\ldots,B_m)$ as depending on the number of relations involved in the CN. We would like to prioritize queries in which terms are placed "near" each other, by penalizing queries whose terms are distributed among several relations. We also ensure that the base relations of the CN is observed if all the attributes are observed. Thus, we define this probability as follows:

$$P(T|B_1,\ldots,B_m) = \begin{cases} \frac{1}{|T|} & \text{iff } B_1 = 1 \wedge \ldots \wedge B_m = 1 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where $|T|$ is the number of relations involved in the CN.

Once all conditional probabilities are defined, we can derive the final equation that represents the probability of observing C given the current state of base relation T. As we use TF-IAF to compute the conditional probability $P(B_j|\overrightarrow{b_j})$, we derive the following equation from Equation 4.

$$P(C|T) = \eta \times cos(\overrightarrow{B_1},\overrightarrow{b_1}) \times \ldots \times cos(\overrightarrow{B_m},\overrightarrow{b_m}) \times \frac{1}{|T|} \tag{10}$$

where each $\overrightarrow{b_j}$ is the state where only the query terms referring to attribute B_j are active, $|T|$ is the number of tuple sets in the CN and η accounts for the constants μ, and $P(\overrightarrow{b_1})$ to $P(\overrightarrow{b_n})$.

Equation 10 is applied to each Candidate Network and returns a probabilistic score for each one.

Term Index

As mentioned in Section III, to compute Equation 10 we use an inverted index that associates each term k found in the relations of the database to a list whose elements are pairs of the form $\langle B_j, f_{kj}\rangle$, where B_j is an attribute in a relation, in whose values the term k occurs with frequency f_{kj}. The term index is constructed in a preprocessing step that scans once all the tables over which queries will be used. This step precedes the processing of queries, and we assume that it does not need to be repeated often. Under this assumption, CNs are ranked for all queries without further interaction with the DBMS. In experiments we performed with databases used in previous R-KwS papers in the literature, it was possible to store the term index entirely in main memory. Notice that the query index is important to our method since the information it keeps is not usually available in existing database statistics.

C. Ranking Algorithm

The CNRank algorithm is described in Fig. 7. It is a direct application of the model we derived in the previous section. The algorithm iterates over a set of CNs given as input (Loop 1–17) and computes a score to each CN according to Equation 10. For each CN, the algorithm process each predicate α_j to compute the cosine componentes of this equation (Loop 3–15). In Line 7, the algorithm takes the frequency f_{kj} of term t_k in values of attribute B_j by looking up the entry corresponding to this term in the term index described in Section III. This frequency is the used to computed the $tf \times iaf$ value of term t_k with respect to attribute B_j according to Equation 8 in Line 9. Each cosine from Equation 10 is computed in Line 13 according to Equation 5. Variable $wsum$ is the summation of weights appearing in numerator of Equation 5 and the function **anorm**(B_j) computes the term $\sqrt{\sum_{t_k \in \overrightarrow{B_j}} w_{jk}^2}$. The other term, $\sqrt{\sum_{t_k \in \overrightarrow{b_j}} g_k(\overrightarrow{b_j})^2}$ is the same for all CNs of a same query, thus, it does not need to be computed for ranking purposes and can be, thus, omitted. After the scores corresponding to all CNs are calculated, the ranking $\mathcal{R}$ is built (Line 18).

```
Algorithm CNRank
Input: A set of CNs $\mathcal{C} = \{C_1, \dots, C_n\}$
Output: A rank of CNs $\mathcal{R} = \langle C_{p(1)}, \dots, C_{p(n)} \rangle$
 1: for each $C_i \in \mathcal{C}$ do
 2:     $cosprod \leftarrow 1$
 3:     for each $\alpha_j \in C_i$ do
 4:         Let $\alpha_j = B_j \supseteq K_j$
 5:         $wsum \leftarrow 0$
 6:         for each $t_k \in K_j$ do
 7:             $f_{kj} \leftarrow$ TermIndexLookUp$(t_k, B_j)$
 8:             if $f_{kj} \neq 0$ then
 9:                 $w \leftarrow$ tfiaf$(f_{kj}, t_k, B_j)$
10:                 $wsum \leftarrow wsum + w$
11:             end if
12:         end for
13:         $cos \leftarrow wsum/$anorm$(B_j)$
14:         $cosprod \leftarrow cosprod \times cos$
15:     end for
16:     $score_i \leftarrow \eta \times cosprod \times \frac{1}{|T|}$
17: end for
18: Build $\mathcal{R}$ such that $C_{p(a)} \preceq C_{p(b)}$, iff $score_a \geq score_b$
```

Fig. 7. The CNRank Algorithm

V. Experimental Results

In this section, we report a comprehensive set of experiments performed with CNRank on real case databases. The experiments consist of submitting unstructured queries to CNGen [5], which is used to generate a number of Candidate Networks, and then submitting these CNs to CNRank. In our first experiment, we evaluated the result ranking with respect to its quality; that is, we verified whether our algorithm is able to place the more relevant CNs on the top of the ranking. Next, we submitted a few top CNs to CN evaluation algorithms previously proposed in the literature and compared the obtained results, with those obtained without using CNRank; that is, when all generated CNs are evaluated, as done in current R-KwS systems. For the evaluation of CNs, we used two very well-known algorithms, namely Hybrid [7] and Skyline Sweeping [8]. When possible, we also compared our results with the those obtained by several other R-KwS systems presented in the literature. Finally, we also present a number of results regarding the impact of our algorithm on the performance and scalability of processing keyword queries in R-KwS systems.

A. Experimental Setup

All of the experiments that we performed ran on an AWS Virtual Machine (m1.small, 64-bit, 1.7 GiB RAM, 1ECU, 1vCPU, 160GB of Instance Storage, Network performance low) running on Centos 6.3 Linux. We used PostgreSQL as the underlying RDBMS with a default configuration. All algorithms were implemented using Java with PostgreSQL.

For the evaluation, we used 4 datasets: IMDb, Mondial, Wikipedia and DBLP, which were previously used for the experiments used in [13], [8] and many other studies. The details on these datasets are described in [17], from which we downloaded all relations used. In the case of DBLP, we downloaded the relations from [19]. In Table I we present some details on these datasets, including their size (in MB), the number of relations, the total number of tables and the number of Referential Integrity Constraints (RIC) in their schemas. Further details on IMDb, Mondial and Wikipedia datasets are provided in [17], whereas more details on the DBLP dataset can be found in [19]. Regarding the keyword queries, in an effort to provide a fair comparison with previous work, we used 3 query sets from different sources as described below and summarized in Table II.

TABLE I
Characteristics of the datasets used.

Dataset	Size (MB)	Relations	Tuples	RIC
Mondial	9	28	17,115	104
IMDb	516	6	1,673,074	4
Wikipedia	550	6	206,318	5
DBLP	40	6	878,065	6

TABLE II
Datasets and Query sets used.

Dataset	Query set size			
	Coffman	SPARK	INEX	Total
IMDb	42	22	14	78
Mondial	42	35	-	77
Wikipedia	45	-	-	45
DBLP	-	18	-	18
TOTAL	129	75	14	218

Coffman: Queries used in in [17]. There were 42 to 45 queries targeting datasets IMDb, Mondial and Wikipedia.

SPARK: Queries used in [20]. For the IMDb dataset, 7 out of the 22 queries originally proposed were replaced. This decision was made because these queries refer to a table on film genres, and this table was not available in the sample of IMDb obtained from [17]. We then replaced these queries with equivalent queries that mention person names instead. The remaining 15 queries were used without modification. For the DBLP dataset, 2 out of 18 queries were replaced because they did not generate results in the database. For the Mondial dataset, all 35 original queries were used..

INEX: We used 14 queries originally specified for the INEX 2011 challenge [21] that could be applied to searches in relational databases. The other 29 queries from the challenge were disregarded in our experiments, because they mentioned structural XML elements.

Thus, we experimented with a total of 218 queries. The complete list of all queries used is available in http://shine.icomp.ufam.edu.br/cnrank[3]

Golden Standards

To verify the effectiveness of our CNRank algorithm and its impact on the evaluation of CNs, it was necessary to use the set of relevant CNs and relevant JNTs for each of the tested queries. In the case of the Coffman query set, the set of relevant JNTs is provided for each query in [17]. To obtain the set of relevant CNs we simply took the CN that generates each relevant JNTs. In the case of the SPARK and INEX query sets, we obtained the golden sets using the following procedure. For each query, we first, generated all CNs using the CNGen algorithm [5]. Then, we manually evaluated all CNs to determine the relevant ones, which is how we obtained the CN golden sets. Next, we ran an SQL query based on each CN and obtained the resulting JNTs. The JNT golden set of a query is, then, the set of all JNTs generated by the CNs in its CN golden set. The golden sets we generated and used for JNTs and CNs are available in http://shine.icomp.ufam.edu.br/cnrank[2].

B. Number of Relevant CNs

Our work is based on the hypothesis that the number of relevant CNs per query is typically much lower than the number of all possible CNs. Using the golden sets we generated, we could corroborate this hypothesis.

In Table III we present the number of relevant CNs found per query, considering all queries and each query set individually.

TABLE III
NUMBER OF RELEVANT CNS PER QUERY.

Query set	1 Relevant CN		2 Relevant CNs	
Coffman	129	100%	-	-
SPARK	45	60%	30	40%
INEX	14	100%	-	-
Total	188	86%	30	14%

Note that the maximum number of relevant CNs for any query is only 2. In fact, all queries that have 2 relevant CNs are from the SPARK query set. The vast majority of queries (86% overall) have one single relevant CN. In the case of the Coffman and INEX datasets, all queries had a single relevant CN. In fact, many queries had one single JNT as an answer.

[3]The URL will be available upon publication of the paper.

In Table IV, for each query set used, we compare the number of CNs generated with the number of relevant CNs. In each case, we show the maximum (MAX) number and the average (AVG) number of CNs.

TABLE IV
GENERATED CNS VS. RELEVANT CNS.

Datasets	Generated CNs / Relevant CNs					
	Coffman		SPARK		INEX	
	MAX	AVG	MAX	AVG	MAX	AVG
IMDb	62/1	18.0/1	42/2	25.9/1.5	163/1	33.4/1
Mondial	62/1	16.6/1	820/2	107.7/1.2		
Wikipedia	102/1	23.0/1				
DBLP			1531/2	255.8/1.6		

Notice that number of relevant CNs is indeed much lower than the number of all CNs. There are cases in which the number of relevant CNs is less than 0.1% of all CNs.

C. CN Ranking

In this section we present experimental results on the effectiveness of our algorithms in the task of ranking Candidate Networks given an unstructured query.

In Fig. 8, we present an evaluation of the ranking produced by CNRank using the MRR (Mean Reciprocal Rank) metric. Given an unstructured query U, the value of RR_U is given by $\frac{1}{K}$, where K is the rank position of the first relevant CN in the ranking. Then, the MRR obtained for queries in a query set is the average of RR_U, for all U in the query set. Intuitively, the MRR metric measures how close relevant CNs are from the first position in the ranking generated by CNRank.

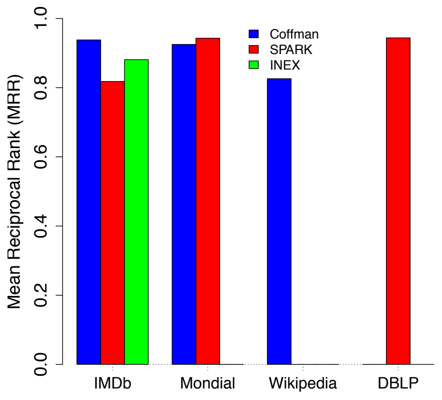


Fig. 8. MRR values achieved by CNRank.

In summary, the results in Fig. 8 show that CNRank is able to place the relevant CNs in the top positions of the rank, provided that CNGen generates this target CN. For 5 of the 7 combinations of query set/datasets, MRR values were above 0.9. In the two remaining cases, MRR was also high: 0.81 for SPARK/IMDb and 0.82 for Coffman/Wikipedia.

Next, we present the precision levels achieved by CNRank using the $P@K$ (precision at position K) metric. Given an keyword query U, the value of $P_U@K$ is 1 if the target query for U appears in a position up to K in the ranking, and 0 otherwise. $P@K$ is the average of $P_U@K$, for all U in a query set. Fig. 9 presents $P@1$ to $P@4$ results for all query sets.

First of all, we observed that in all query sets, the correct answers was always among the top-4 positions in the ranking, since P@4 is 1 in all cases. Considering all queries, on average, $P@1$ is 0.83, which means that the CNRank very often assigns a relevant CN to the top-1 position. Importantly, in all cases, $P@1$ was no lower than 0.63. Also, notice that P@2 is 1 in all queries from the SPARK query set.

We noticed that in many cases the inherent ambiguity of certain queries was harmful to the evaluation we adopted. For instance, for the query *"fast food"*, the information need statement provided by INEX requires a movie in the answer. For this query, the CNRank algorithm placed the CN that provides the required answer in the second position. However, we found out that the CN assigned by CNRank to the first position, which retrieves a character instead, is also plausible and provides a suitable answer.

D. Impact on CN Evaluation

As we have already shown, by using CNRank, we can drastically reduce the number of CNs that are evaluated. That is, instead of evaluating tens or hundreds of CNs as in current R-KwS systems, only a few top CNs selected using CNRank need to be evaluated. In this section we show experiments we performed to verify the impact of using only a few top CNs in evaluating CNs to produce the JNTs that comprise the answer for an unstructured query.

For this experiment, two distinct evaluation algorithms were used, namely, Hybrid [7] and Skyline-Sweeping [8]. These evaluation algorithms were the best among those proposed in their respective papers and are highly representative of the state of the art in CN evaluation. We compared the results obtained by each algorithm in three different configurations considering the input they receive: (1) all generated CNs, identified respectively as "Hybrid" and "Skyline"; (2) only the top-4 CNs selected with CNRank, identified respectively as "HybRank4" and "SSRank4"; and (3) only the top-1 CN selected with CNRank, identified respectively as "HybRank1" and "SSRank1". The decision of using the top-4 CNs was suggested by the results from Fig. 9, where we have shown that the relevant CNs were among the top-4 CNs ranked by CNRank.

In Fig. 10, we present a comparison of the results obtained with each of these configurations using the MRR metric. The MRR values were calculated in a way similar to that described above for evaluating CNs, but this time we applied it to evaluating the ranking of JNTs.

The graph in Fig. 10 shows that using CNRank yields high MRR values in all query sets. Indeed, these values are much higher than the values obtained without CNRank. This corroborates our claims regarding the positive impact of using CNRank in CN evaluation. This was observed for both Hybrid and Skyline-Sweeping, which suggests that CNRank can improve the results of any CN evaluation algorithm. Indeed,

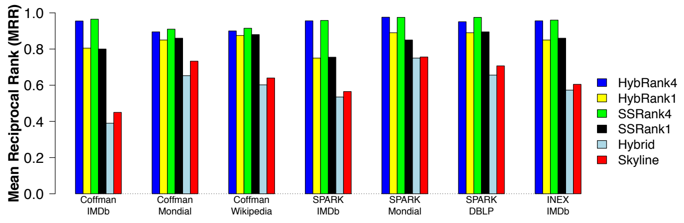


Fig. 10. Effect of CNRank on CN evaluation in terms of MRR.

this occurs due to the fact that now the CN evaluation algorithm receive only those CNs that are likely to produce relevant JNTs.

Notice that the configurations that use top-4 CNs achieved better results than the top-1 configurations. This is expected, since top-1 configurations may miss relevant CNs. Nevertheless, the top-1 configurations performed better than the baselines.

Incidentally, notice that the Skyline-Sweeping performed better than the Hybrid algorithm. However, their counterparts that use CNRank provided very similar results.

In [13], a similar experiment was performed, also using the MRR metric. In this case, however, the authors only used queries that returned a single JNT as results. The goal was to evaluate how well this single-answer JNT is placed in the ranking. We also ran this same experiment with the configurations we implemented. To distinguish this metric, which applies to a single answer, from the one used in Fig. 10, which applies to multiple answers as well, we called this metric *SMRR*. The result of this experiment is presented in Fig. 11.

Performing this experiment allowed us to compare our results with those obtained with a number of other R-KwS systems evaluated in [13], and whose detailed numbers were generously provided by its authors. This includes not only systems based on the Schema Graph approach, i.e., DISCOVER [5], Efficient [7], Bidirectional [2], Effective [4], and CD [9], but also in systems based on the Data Graph approach, i.e., BANKS [1], DPBF [10] and BLINKS [3]. In this case, all results are from the Coffman query set, which was the only one used in the experiments reported in [13].

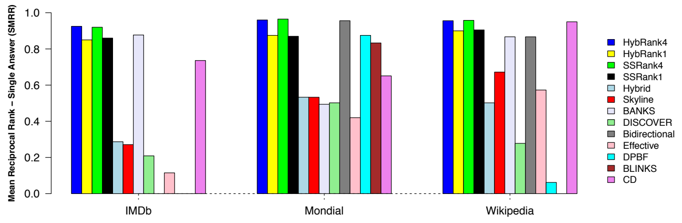


Fig. 11. Impact of CNRank. SMRR on Coffman query set.

Again, the configurations based on CNRank provided superior values of SMRR, with a slight advantage to configurations that receive top-4 CNs. These configurations had results equal to the best system in each dataset, that is, BANKS in IMDb, Bidirectional in Mondial and CD in Wikipedia. It should be noted that none of the tested system performed consistently well across all datasets. However, such a trend was observed in all CNRank-based configurations.

To evaluate the impact of CNRank in the overall quality of

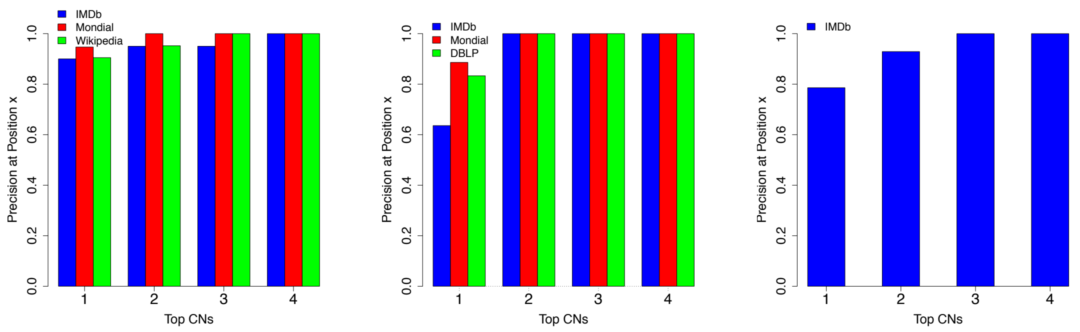


Fig. 9. P@k values achieved by CNRanking on Coffman (left) SPARK (midlle) and INEX (right).

the ranking, we used the MAP (Mean Average Precision). Let A be a ranking of JNTs generated for a given keyword query U. The *Average Precision (AP_U)* of this ranking is the average of the precision values calculated at each position k in which there is a relevant JNT in A. That is, $AP_U = \sum_{k=1}^{n} P(k) \times rel(k)/|R|$, where n is the number of JNT considered from the ranking A (in our case, $n = 1000$), $P(k)$ is the precision at position k, $rel(k)$ is 1 if the answer at position k is relevant or 0 otherwise, and R is the set of known relevant answers for U. Then, the *Mean Average Precision (MAP)* is the average of AP_U, for all U in a query set.

In Fig. 12, we present the MAP values obtained with the configurations we implemented. Again, for the case of the Coffman dataset, it was possible to compare our results with those reported in [13], as shown in Fig. 12 (right).

The MAP results in Fig. 12 reveal that CNRank also affects the global ranking quality in all tested scenarios. Again, the CNRank-based configurations consistently achieved better results across different query sets and datasets, compared to all previously proposed methods considered.

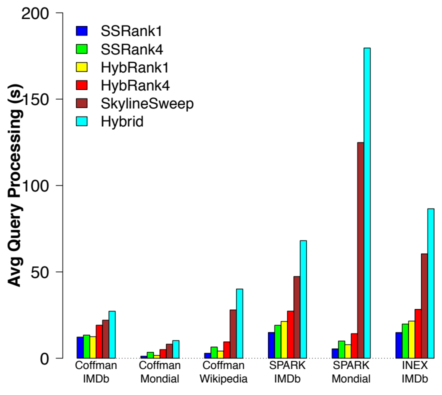
E. Impact on Performance

An important question regarding introducing CNRank in the processing of keyword queries in R-KwS systems is how it affects their performance.

In Fig. 13 we present, for each query set/dataset pair, the average time spent processing a keyword query. In the case of ”Hybrid” and ”Skyline” configurations, this time includes the generation and evaluation of CNs. In the case of ”HybRank4”, ”SSRank4”, ”HybRank1” and ”SSRank1” this time also includes the ranking of CNs performed by CNRank. The graph in Fig. 13(top) includes all query set/dataset pairs, except for SPARK/DBLP, whose values required a different scale. We decided to show this result in a separate graph in Fig. 13(bottom).

Fig. 13 shows that, as expected, CNRank also positively affects performance. Obviously, it is due to the fact that CN evaluation algorithms now have much less CNs to handle. Also, these graphs show that the process of ranking CNs introduced between the generation and evaluation of CNs does not imply any significant overhead to the whole process.

We wish to point out that the Skyline-Sweeping algorithm performed better than Hybrid, which is consistent with the results presented in [8]. This trend was also observed, on a



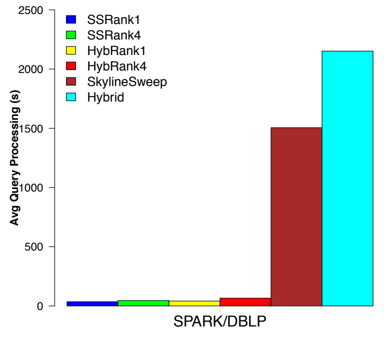


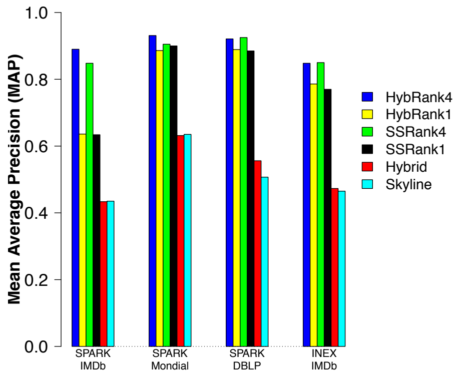
Fig. 13. Impact of CNRank on CN evaluation – Performance.

smaller scale, in the CNRank-based configurations that use each algorithm.

Interestingly, we did not observe a significant advantage, in terms of performance, of the the configurations that use only the top-1 CN over those that use top-4 CNs. This suggests that top-4 CN configurations, which in the previous experiments resulted in a better a ranking in terms of quality, are preferable.

VI. Conclusion

In this paper we propose an approach for ranking Candidate Networks of Relations (CNs), which are relational join expressions adopted by many Relational keyword search (R-KwS) systems to produce answers in the form of join networks of tuples (JNT). Our motivation comes from evidences we collected, and also report here, that, although the number of possible Candidate Networks can be very high, only very few of them produce answers relevant to the user and are indeed worth processing. We experimentally show that our ranking



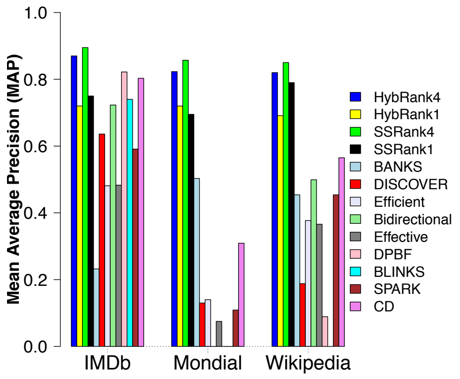


Fig. 12. Impact of CNRank on CN evaluation in terms of MAP – SPARK and INEX (left), Coffman (right).

algorithm is able to place the relevant CNs among the top-4 in the ranking it produces. Our experiments also show that, in comparison to traditional R-KwS systems that process all possible CNs, processing only those top-4 CNs improves not only the time it takes to return answers and but also the quality of the answers retrieved.

Currently, our weighting scheme used for calculating CN scores is based only on features related to the lexical properties of the atributes, i.e., on measuring which terms are more representative for a given attribute. Although we have achieved good results with this scheme, it would be interesting to investigate other weighting schemes based, for instance, on key/foreign key statistics or on user preferences, both explicitly declared or inferred from query logs. In particular, log entries can provide helpful information on the typical positioning and sequencing of values of certain attributes in user queries. Another possible future line of investigation is exploring alternatives to the Bayesian framework for combining the individual probabilities estimated for the attributes. This line is particularly interesting if the number of features to consider increases, as suggested above. In this case, we believe that approaches based on machine learning are worth being considered.

VII. Acknowledgments

Work partially funded by the project TTDSW (PRONEM/FAPEAM/CNPq) and by the authors' individual grants from CNPq. The authors are grateful to the MS Azure Research Pass Grant, for the computational infrastructure used in the experiments, to the authors of DISCOVER [5] and SPARK [8] for making their code available, and to Joel Coffman for sharing his experimental results.

References

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, P. Parag, and S. Sudarshan, "BANKS: Browsing and keyword searching in relational databases," in *Proc. of the 28th Intl. Conf. on VLDB*, 2002, pp. 1083–1086.

[2] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *Proc. of the 31st Intl. Conf. on VLDB*, 2005, pp. 505–516.

[3] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: Ranked keyword searches on graphs," in *Proc. of the 2007 ACM SIGMOD Intl. Conf. on Management of Data*, 2007, pp. 305–316.

[4] F. Liu, C. Yu, W. Meng, and A. Chowdhury, "Effective keyword search in relational databases," in *Proc. of the 2006 ACM SIGMOD Intl. Conf. on Management of Data*, 2006, pp. 563–574.

[5] V. Hristidis and Y. Papakonstantinou, "DISCOVER: keyword search in relational databases," in *Proc. of the 28th Intl. Conf. on VLDB*, 2002, pp. 670–681.

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

[7] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-style keyword search over relational databases," in *Proc. of the 29th Intl. Conf. on VLDB*, 2003, pp. 850–861.

[8] Y. Luo, X. Lin, W. Wang, and X. Zhou, "SPARK: Top-k keyword query in relational databases," in *Proc. of the 2007 ACM SIGMOD Intl. Conf. on Management of Data*, 2007, pp. 115–126.

[9] J. Coffman and A. C. Weaver, "Structured data retrieval using cover density ranking," in *Proc. of the 2nd Intl. Workshop on KwS on Structured Data*, 2010, pp. 1:1–1:6.

[10] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *Proc. of the 23rd Intl. Conf. on Data Engineering*, 2007, pp. 836–845.

[11] A. Markowetz, Y. Yang, and D. Papadias, "Keyword search on relational data streams," in *Proc. of the 2007 ACM SIGMOD Intl. Conf. on Management of Data*, 2007, pp. 605–616.

[12] A. Baid, I. Rae, J. Li, A. Doan, and J. Naughton, "Toward scalable keyword search over relational data," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 140–149, 2010.

[13] J. Coffman and A. C. Weaver, "A framework for evaluating database keyword search strategies," in *Proc. of the 19th ACM Intl. Conf. on Information and Knowledge Management*, 2010, pp. 729–738.

[14] A. Nandi and H. V. Jagadish, "Qunits: queried units in database search," in *CIDR 2009, Fourth Biennial Conf. on Innovative Data Systems Research*, 2009.

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

[16] B. A. N. Ribeiro and R. Muntz, "A belief network model for ir," in *Proc. of the 1996 ACM SIGIR Intl. Conf. on Research and Development in IR*, 1996, pp. 253–260.

[17] J. Coffman and A. C. Weaver, "Relational keyword search benchmark," 2010, http://www.cs.virginia.edu/ jmc7tp/resources.php.

[18] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval - the concepts and technology behind search, Second edition.* Pearson Education Ltd., Harlow, England, 2011.

[19] R. Cyganiak, "Benchmarking D2RQ v0.1," 2007. [Online]. Available: http://wifo5-03.informatik.uni-mannheim.de/bizer/d2rq/benchmarks/index01.html

[20] Y. Luo, X. Lin, W. Wang, and X. Zhou, "SPARK: top-k keyword query in relational databases.technical report unsw-cse-tr-0708," in *Proc. of the 2007 ACM SIGMOD Intl. Conf. on Manag. of data*, 2007.

[21] "INitiative for the Evaluation of XML Retrieval (INEX)," https://inex.mmci.uni-saarland.de/about.html, 2011.