Keyword Search over Relational Databases: A Metadata Approach

Sonia Bergamaschi
University of Modena and Reggio Emilia, Italy
sonia.bergamaschi@unimore.it

Elton Domnori
University of Modena and Reggio Emilia, Italy
elton.domnori@unimore.it

Francesco Guerra
University of Modena and Reggio Emilia, Italy
francesco.guerra@unimore.it

Raquel Trillo Lado
University of Zaragoza, Spain
raqueltl@unizar.es

Yannis Velegrakis
University of Trento, Italy
velgias@disi.unitn.eu

ABSTRACT

Keyword queries offer a convenient alternative to traditional SQL in querying relational databases with large, often unknown, schemas and instances. The challenge in answering such queries is to discover their intended semantics, construct the SQL queries that describe them and used them to retrieve the respective tuples. Existing approaches typically rely on indices built a-priori on the database content. This seriously limits their applicability if a-priori access to the database content is not possible. Examples include the on-line databases accessed through web interface, or the sources in information integration systems that operate behind wrappers with specific query capabilities. Furthermore, existing literature has not studied to its full extend the inter-dependencies across the ways the different keywords are mapped into the database values and schema elements. In this work, we describe a novel technique for translating keyword queries into SQL based on the Munkres (a.k.a. Hungarian) algorithm. Our approach not only tackles the above two limitations, but it offers significant improvements in the identification of the semantically meaningful SQL queries that describe the intended keyword query semantics. We provide details of the technique implementation and an extensive experimental evaluation.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process; Retrieval models; Query formulation*

General Terms

Algorithms, Experimentation, Performance

Keywords

Semantic Keyword Search, Intensional Knowledge, Relational Databases, Metadata

SIGMOD'11, June 12–16, 2011, Athens, Greece.

1. INTRODUCTION

The more the relational data complexity is increasing and the user base is shifting towards the less technically skilled, the more the keyword searching is becoming an attractive alternative to traditional SQL queries, mainly due to its simplicity. Unfortunately, this simplicity comes with the price of inherent ambiguity. Thus, the challenge of answering a keyword query over a relational database is to discover the database structures that contain the keywords and explore how these structures are inter-connected to form an answer. The discovered structures, alongside their inter-connections, are actually representing in relational terms the semantic interpretation of the keyword query.

Numerous studies and tools can already be found in the scientific literature. They include DISCOVER [16], DBXplorer [2], BANKS [1], SPARK [22], SQAK [31], and many others [21, 29, 27, 33, 34, 38]. Generally, these works consider the database as a network of interconnected tuples, they detect those containing the keywords in the query, they generate connected components based on how these tuples are associated, and they return these connected tuples as an answer to the query. To do so, specialized structures that index the database content [2] are used. By using these indices, they may directly retrieve the tuples of interest, or they may instead construct the queries expressions that retrieve these tuples when evaluated. This is the basic idea followed by the modern commercial database management systems supporting full-text search over their relational database.

Unfortunately, existing techniques suffer from two main limitations. The first is that they require a-priori access to the data instance in order to build the indices that will locate the tuples related to the given keywords at run time. This seriously limits their applicability if such access is not possible. Examples of such situations include databases on the hidden web and sources located behind wrappers in data integration systems [19] that typically expose only their schema information and lack notification mechanisms for their data updates. The second limitation is that no considerable attention has been paid to the inter-dependencies among the query keywords. The likelihood that a specific data structure represent the same semantics as a keyword in a user query does not only depend on the relationship between the keyword and the data structure, but also on the data to which the other keywords in the query are mapped. This is because despite the fact that a keyword query is a flat list of keywords, the meaning of each keyword is not independent of the meaning of the others, but they all collectively represent the intended concepts the user had in mind posing the query. Furthermore, not all the keywords represent instance values. Many are used as meta-data specification of the adjacent

keywords. Although there are already keyword based approaches on relational data that take into consideration metadata [21][31], they provide only partial solutions to the problem, and they only use the metadata as a way to improve their technique.

In this work, we propose a novel technique for answering keyword queries over relational databases. The queries are translated into a number of SQL queries that capture the possible semantics of the keyword query. The generated SQL queries can be evaluated on the database, and their results serve as the answer to the keyword query. One of the novelties of the technique is that it is not based on an a-priori access to the database instances. Moreover, our approach exploits the relative positions of the keywords in the query alongside auxiliary external knowledge in order to make a more educated guess of the semantics that most likely represent those of the keyword query, and then rank them accordingly. The strategy can not only be easily incorporated to many relational database management systems, but it can also be used as an enhancement of existing keyword searching techniques that utilize the database instance, offering them significant improvements over effectiveness and efficiency.

An advantage of our approach is that it can be used to assist users browsing databases with large unknown schemas. It is often the case that users formulate keyword queries without some specific semantics in mind but in an exploratory manner, mainly when they are neither fully aware of the type of information that is stored in a database, nor of the way this information is stored. The possible interpretations of a keyword query according to the schema and domain information of the database will be generated by our approach. In contrast to other keyword searching techniques on relational data that return sets of linked tuples, we can return the interpretations expressed in SQL. The study of these queries can reveal tables, attributes and join paths, providing the user with enough information to understand the kind of data that is stored in the database and the way this data is structured.

Our key contributions are as follows: (i) we formally define the problem of keyword querying over relational databases that lack a-priori access to the database instance; (ii) we introduce the notion of a weight as a measure of the likelihood that the semantics of a keyword are represented by a database structure, i.e., a table, an attribute, or a value. We further distinguish the weights to intrinsic and contextual, to emphasize that this likelihood does not depend only on the meaning of the keyword semantics when the keyword is considered in isolation (intrinsic), but also on the way the semantics of the remaining, especially the neighbouring, keywords are represented in the data (contextual). (iii) we extend and exploit the Hungarian (a.k.a., Munkres) algorithm [7] to develop a technique for the systematic computation of the contextual weights that leads into to the generation and ranking of the different interpretations of a keyword query in terms of SQL; finally, (iv) we experimentally evaluate our approach on real application scenarios.

The remainder of the paper is structured as follows: Section 2 provides a motivating example and Section 3 formally defines the problem. Section 4 describes our technique. Sections 5 - 7 provides details on our technical contributions, i.e. the computation of the intrinsic weights, the contextualization and the selection of the best mappings. The relationship of our approach to the related work is discussed in Section 8 and Section 9 describes our experimental evaluation and discusses our findings. Finally some conclusions are presented in Section 10.

2. MOTIVATING EXAMPLE

Consider the database illustrated in Figure 1, containing information about academic researchers, departments, publications, and publication databases, all modeled by the respective tables. The tables `Affiliated` and `Author` materialize many-to-many relationships between `Person-Department` and `Person-Publication`, respectively. Let "`Date Database`" be a keyword query posed over this database. To answer this query we need to understand the meaning of each keyword and build an SQL query that offers an interpretation of the keyword query semantics in terms of the relational database.

The first problem of this translation is to decide *what* role each keyword plays in the query, i.e., it represents a value or describes some meta-information (specification) about another keyword or the element of interest. In the above query, the keyword `Date` may represent the value of a name, and the keyword `Database` the value of a research area. In that case, a possible interpretation of the keyword query is information about a person called `Date` that has done work in the `Database` area. If the intended meaning of the keyword query was instead to find the online databases containing `Date`'s publications, then the keyword `Database` is actually representing meta-information about the element of interest.

Knowing *what* kind of information each keyword represents, the next critical step is to decide *which* part of the database actually models the intended keyword meaning. Consider the keyword query "`Director Watson Address`" and assume that the first and last keyword represent meta information while the second represents a value. The intended information of the keyword `Director` is most probably the one modeled by the attribute with the respective name. It is not clear, however, whether the keyword `Address` refers to the homonym attribute in the table `Person` or to the one in the table `Department`. Choosing one over another leads to completely different interpretations of the keyword query. Furthermore, the keyword `Watson`, even though we know it is a value, might be the name of a director, the name of a street, or even a value of some other attribute.

Deciding *which* database structures model the meaning of the different keywords in the query is not enough. It is also important to decide *how* these structures relate to each other. In relational databases, such relationships are expressed either through the table-attribute-value hierarchy or through join paths, i.e., chains of key/foreign key relationships. Between two database structures there may be multiple different join paths. Each such path leads to a different interpretation of the query keywords. For instance, consider the keyword query "`email CS`", and assume that the keyword `email` corresponds to attribute `Email` and keyword `CS` to a value of the attribute `DName`. Between `DName` and `Email`, there are two different paths, one that goes through the table `Affiliated` and one through the `Director`. If the semantics of the keyword query were to find emails of the CS affiliated persons, then the database query describing this semantics is one that uses the first path. If instead the semantics were to find the email of the CS department director, the query is one that uses the second path.

Finding the different semantic interpretations of a keyword query is a combinatorial problem which can be solved by an exhaustive enumeration of the different ways that mappings can be associated to database structures and values. The challenging task is to develop a mechanism that is able to significantly reduce the possible associations that most likely represent the intended keyword semantics. One way to do this is to exploit syntactic knowledge or other forms of auxiliary information. For instance, a keyword "`(320)463-1463`" is more likely to represent a phone number due to its format, while a keyword `Everest`, given the publicly available geo-name knowledge, is very likely to represent the mount Everest. Similarly, the keyword "`article`" is likely to cor-

Person

Name	Area	Phone	Address	Email
Watson	Database	(320) 4631234	30 Bloor	watson@aaa.bb
Lenzerini	Database	(390) 6987654	Ariosto 25	lenzerini@bbb.cc
Date	Database	(817) 1937842	107 GACB	date@ccc.dd
Hunt	Inf. Systems	(343) 2920812	17 Helix	Hunt@ddd.ee

Affiliated

Professor	Department
Watson	x123
Lenzerini	cs34
Date	cs34
Hunt	m111

Department

id	DName	Address	Director
x123	CS	25 Blicker	Watson
cs34	IE	15 Tribeca	Hunt
ee67	EE	5 Charles	Date
m111	ME	2 Cottle	Hunt

Author

Name	Publication
Lenzerini	Data Integration
Date	Foundation Matters

Publication

Title	Year	Resource
Data Integration	2002	ACM DL
Foundation Matters	2002	DBLP

Database

Name	Address
DBLP	http://www.informatik.uni-trier.de
ACM DL	http://portal.acm.org/dl.cfm

Figure 1: A fraction of a database schema with its data.

respond to the table `Publication`, based on some lexical database knowledge like WordNet[1].

The quest for the database structure that a keyword corresponds should not be an isolated task. Keywords are not independent entities, but it should be seen in the *context* of the presence of the other keywords in the query. To illustrate this fact, consider the keyword query "`Name Date Database`". A possible interpretation of the query is that the user is looking for the person called Date who works in the area of databases. This means that keywords `Name` and `Date` may correspond to the attribute `Name` of the table `Person` and one of its values, respectively, while the keyword `Database` to one of the values of the attribute `Area`. Consider now the query "`Name Date Database DBLP`". In this case, a possible interpretation is that the user is looking for data items related to Date in the DBLP database, hence, the meaning of the keyword `Database` is represented by the table `Database` and not by some value of the `Area` attribute.

A natural and challenging question that comes to mind is which of the different semantic interpretations of a query should be considered as the correct one. It is a well-known and documented fact that keyword queries are under-specified queries [18]. As such, any of the possible interpretations may be correct, and all have to be considered. However, based on some known patterns of human behavior [18], we know that the order of keywords is important and that correlated keywords are typically close. This means that certain interpretations of a keyword query are actually more likely than others. Thus, instead of returning a flat list of possible interpretations to the user, one can return a ranked list based on the likelihood that they represent the intended keyword semantics.

3. PROBLEM STATEMENT

A database D is a collection of relational tables. A relational table is denoted as $R(A_1, A_2, \ldots, A_n)$, where R is the name of the table and $A_1, A_2, \ldots, A_n$ its attributes. The *vocabulary* of the database D, denoted as V_D, is the set $V_D=\{X \mid \exists R(A_1, A_2, \ldots, A_n)\in D$, s.t. $X=R \vee X=A_k \vee X=Dom(A_k)$ with $1\leq k\leq n\}$. In other words, the vocabulary of a database is the set of all its relation names, their attributes and their respective domains. A *database term* is a member of its vocabulary.

A keyword query q is an ordered list of keywords $\{k_1, k_2, \ldots, k_n\}$. Each keyword is a specification about the element of interest. The specification may have been modeled in the database as a relational table, an attribute, or a value of an attribute. A configuration is a mapping function that describes a specification for each query keyword in terms of database terms.

DEFINITION 3.1. *A* configuration C *of a keyword query* q *on a database* D *is an injective mapping from the keywords in* q *to database terms in the vocabulary of* D.

[1]wordnet.princeton.edu

We have made the natural assumption that each keyword cannot have more than one meaning in the same configuration, i.e., it can be mapped to only one database term. Furthermore, we have assumed that no two keywords can be mapped into the same database term based on the fact that overspecified queries is only a small fraction of the queries that are typically met in practice [18]. However, our technique can be extended to work without this assumption with only minor modifications. We have also made the natural assumption that every keyword plays some role in the query, i.e., there are no unjustified keywords.

Answering a keyword query over a database D means finding a set of SQL queries, with each of these queries using all the database elements that belong to the range[2] of a specific configuration. Such an SQL query is referred to as an *interpretation* of the keyword query, since it provides a possible meaning of the keyword query in terms of the database vocabulary. In the current work, we consider only select-project-join (SPJ) interpretations, that are typically the queries of interest in similar works [2, 16]. Nevertheless, interpretations involving aggregations [31] are also in our list of future extensions.

DEFINITION 3.2. *An* interpretation *of a keyword query* $q=\{k_1, k_2, \ldots, k_n\}$ *on a database* D *using a configuration* C *is an SQL query:* `select` $A_1, A_2, \ldots, A_j$
`from` R_1 JOIN R_2 JOIN ... JOIN R_n
`where` $A'_1=v_1$ AND $A'_2=v_2$ AND ... AND $A'_m=v_m$
such that $\forall k\in q$ *one of the following is true:*

- $C(k)\in\{R_1, R_2, \ldots, R_n\}$
- $C(k)\in\{A_1, A_2, \ldots, A_j\}$
- $C(k)=dom(A'_i)\in\{dom(A'_1), dom(A'_2), \ldots, dom(A'_j)\}$ *and* $\langle A'_i, k\rangle \in \{\langle A'_1, v_1\rangle, \langle A'_2, v_2\rangle, \ldots, \langle A'_m, v_m\rangle\}$

To eliminate redundancies, we require any use of a database term in an interpretation to be justified either by belonging to the range of the configuration, or by participating in a join path connecting two database terms that belong in the range of the configuration. Note that even with that restriction, due to the multiple join paths in a database D, it is still possible to have multiple interpretations of a given keyword query q and a configuration C of it. We will use the notation $\mathcal{I}(q, C, D)$ to refer to the set of these interpretations. and $\mathcal{I}(q, D)$ for the union of all these sets for all the possible configurations of the query q.

EXAMPLE 3.1. *Consider the keyword query "email CS" over the database of Figure 1, and a configuration that maps* `email` *to* `Email` *and* `CS` *to* `DName`. *At least two different interpretations can be generated from this. One is the:*

`select` *Email*
`from` *Person P* `JOIN` *Affiliated A* `ON` *(P.name=A.professor)*
`JOIN` *Department D* `ON` *(A.Department=D.id)*
`where` *D.DName='CS'* AND *D.id=A.Department*
AND *A.Professor=P.Name*

[2]Since a configuration is a function, we can talk about its range.

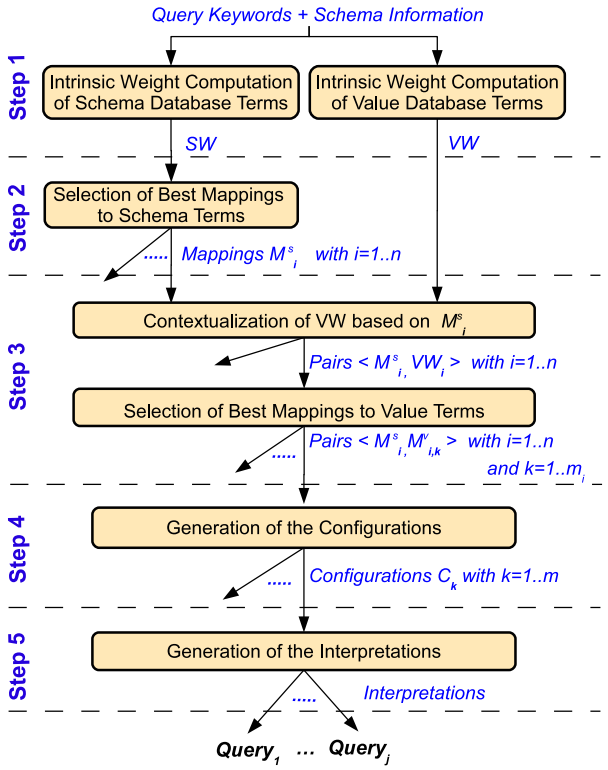


Figure 2: Overview of the keyword query translation process

and the other is the:

`select` *Email*
`from` *Person P* `JOIN` *Department D* `ON` *(P.person=D.Director)*
`where` *D.DName='CS'* AND *D.Director=P.Name*

DEFINITION 3.3. *An* answer *to a keyword query q over a relational database D is the set $ans(q) = \{t \mid t \in eval^D(q') \wedge q' \in \mathcal{I}(q, D)\}$, where $eval^D(q')$ denotes the evaluation of the relational query q' on the database D.*

Note that since there is no restriction on the number of attributes in the select clause of an answer, the answer set of a keyword query will naturally be heterogeneous.

Since each keyword in a query can be mapped to a relation name, an attribute name or an attribute domain, there are $2*\Sigma_{i=1}^{|D|}|R_i| + |D|$ different configurations, with $|R_i|$ denoting the *arity* of the relation R_i and $|D|$ the number of tables in the database. Based on this, and on the fact that no two keywords can be mapped to the same database term, for N keywords, there are $\frac{|V_D|!}{(|V_D|-N)!}$ possible configurations. Of course, not all the interpretations generated by these configurations are equally *meaningful*. Some are more likely to represent the intended keyword query semantics. In the following sections we will show how different kinds of meta-information and inter-dependencies between the mappings of keywords to database terms can be exploited in order to effectively and efficiently identify these interpretations and present them first. Our work is based on the natural assumption that the intended semantics of the keyword query can be expressed as a query over the relational database. If those semantics cannot be expressed, no answer can be provided.

4. FROM KEYWORDS TO QUERIES

The generation of interpretations that most likely describe the intended semantics of a keyword query is based on semantically

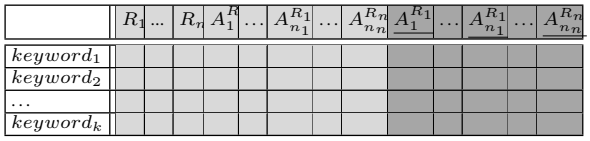


Figure 3: Weight table with its SW (light) and VW (dark) parts

meaningful configurations, i.e. sets of mappings between each keyword and a database term. We introduce the notion of *weight* that offers a quantitative measure of the relativeness of a keyword to a database term, i.e., the likelihood that the semantics of the database term are the intended semantics of the keyword in the query. The sum of the weights of the keyword-database term pairs can form a score serving as a quantitative measure of the likelihood of the configuration to lead to an interpretation that accurately describes the intended keyword query semantics. The range and full semantics of the score cannot be fully specified in advance. They depend on the method used to compute the similarity. This is not a problem as long as the same method is used to compute the scores for all the keywords. This is the same approach followed in schema matching [28] where a score is used to measure the likelihood that an element of a schema corresponds to an element of another.

The naive approach for selecting the best configurations, and, as a consequence, generating the most prominent interpretations of a keyword query, is the computation of the score of each possible configuration and then selecting those that have the highest scores. Of course, we would like to avoid an exhaustive enumeration of all the possible configurations, and compute only those that give high scores. The problem of computing the mapping with the maximum score without an exhaustive computation of the scores of all the possible mappings is known in the literature as the problem of *Bipartite Weighted Assignments* [9]. Unfortunately, solutions to this problem suffer from two main limitations. First, apart from the mutual exclusiveness, they do not consider any other interdependencies that may exist between the mappings. Second, they typically provide only the best mapping, instead of a ranked list based on the scores.

To cope with the first limitation, we introduce two different kinds of weights: the *intrinsic*, and the *contextual* weights. Given a mapping of a keyword to a database term, its intrinsic weight measures the likelihood that the semantics of the keyword is that of the database term if considered in isolation from the mappings of all the other keywords in the query. The computation of an intrinsic weight is based on syntactic, semantic and structural factors such as attribute and relation names, or other auxiliary external sources, such as vocabularies, ontologies, domains, common syntactic patterns, etc. On the other hand, a contextual weight is used to measure the same likelihood but considering the mappings of the remaining query keywords. This is motivated by the fact that the assignment of a keyword to a database term may increase or decrease the likelihood that another keyword corresponds to a certain database term. This is again based on observations that humans tend to write queries in which related keywords are close to each other [18]. As an example, for the keyword query "`Watson Area Database`" expressed on the database in Figure 1, since the keyword "`Database`" is right next to keyword `Area`, mapping the keyword `Area` to the attribute `Area` of the table `Person` makes more likely the fact that the keyword `Database` is an area value, i.e., should be mapped to the domain of the attribute `Area`. At the same time, it decreases its relativeness to the table `Database`. A similar idea has already been exploited in the context of schema matching [24] with many interesting results. To cope with the second limitation, we have developed a novel algo-

	P	D	P.Na	P.Ar	P.Ph	P.Ad	P.Em	D.Id	D.Dn	D.Ad	D.Di	P.Na	P.Ar	P.Ph	P.Ad	P.Em	D.Id	D.Dn	D.Ad	D.Di
$workers$	68	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1
$department$	0	100	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1
CS	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1

Figure 4: Intrinsic Weight SW (light gray) and VW (dark gray) matrix.

rithm for computing the best mappings. The algorithm is based on and extends the Hungarian (a.k.a., Munkres) algorithm [7] and will be described in detail in Section 7.

A visual illustration of the individual steps in the keyword query translation task is depicted in Figure 2. A special data structure, called *weight matrix*, plays a central role in these steps. The *weight matrix* is a two-dimensional array with a row for each keyword in the keyword query, and a column for each database term. The value of a cell $[i, j]$ represents the weight associated to the mapping between the keyword i and the database term j. Figure 3 provides an abstract illustration of a weight matrix. An R_i and $A_j^{R_i}$ columns correspond to the relation R_i and the attribute A_j of R_i, respectively, while a column with an underline attribute name $\underline{A_j^{R_i}}$ represents the data values in the column A_j of table R_i may have, i.e., its domain. Two parts (i.e., sub-matrices) can be distinguished in the weight matrix. One corresponds to the database terms related to schema elements, i.e., relational tables and attributes, and the other one corresponds to attribute values, i.e., the domains of the attributes. We refer to database terms related to schema elements as *schema database terms*, and to those related to domains of the attributes as *value database terms*. In Figure 3, these two sub-matrices are illustrated with different shades of gray. We refer to the weights in the first sub-matrix as *schema weights*, and to those of the second as *value weights*. We also use the notation SW and VW to refer either to the respective sub-matrix, or to their values. The details of the individual steps of Figure 2 are provided next.

Intrinsic Weight Computation. The first step of the process is the intrinsic weight computation. The output is the populated SW and VW sub-matrices. The computation is achieved by the exploitation and combination of a number of similarity techniques based on structural and lexical knowledge extracted from the data source, and on external knowledge, such as ontologies, vocabularies, domain terminologies, etc. Note that the knowledge extracted from the data source is basically the meta-information that the source makes public, typically, the schema structure and constraints. In the absence of any other external information, a simple string comparison based on tree-edit distance can be used for populating the SW sub-matrix. For the VW sub-matrix the notion of *Semantic Distance* [11] can always be used in the absence of anything else. As it happens in similar situations [28], measuring the success of such a task is not easy since there is no single correct answer. In general, the more meta-information has been used, the better. However, even in the case that the current step is skipped, the process can continue with the weight matrix where all the intrinsic values have the same default value. The computation of the intrinsic weights is detailed in Section 5.

Selection of the Best Mappings to Schema Terms. The intrinsic weights provide a first indication of the similarities of the keywords to database terms. To generate the prominent mappings, we need on top of that to take into consideration the inter-dependencies between the mappings of the different keywords. We consider first the prominent mappings of keywords to schema terms. For that we work on the SW sub-matrix. Based on the intrinsic weights, a series of mappings M_1^S,M_2^S, ..., M_n^S, of keywords to schema terms are generated. The mappings are those that achieve the highest overall score, i.e., the sum of the weights of the individual keyword mappings. The mappings are partial, i.e., not all the keywords are mapped to some schema term. Those that remain unmapped will play the role of an actual data value and will be considered in a subsequent step for mapping to value database terms. The selection of the keywords to remain unmapped is based on the weight matrix and some cut-off threshold. Those with a similarity below the threshold remain unmapped. For each of the mappings M_i^S, the weights of its SW matrix are adjusted to take into consideration the context generated by the mapping of the neighborhing keywords. It is based on the observation that users form queries in which keywords referring to the same or related concepts are adjacent [18, 33]. The generation of the mappings and the adjustment of the weights in SW are performed by our extension of the Hungarian algorithm that is described in detail in Section 7. The output of such a step is an updated weight matrix SW_i and, naturally, an updated score for each mapping M_i^S. Given the updated scores, some mappings may be rejected. The selection is based on a threshold. There is no golden value to set the threshold value. It depends on the expectations from the keyword query answering systems. The higher its value, the less the interpretations generated at the end, but with higher confidence. In contrast, the lower the threshold value, the more the mappings with lower confidence.

Contextualization of VW and selection of the Best Mappings to Value Terms. For each partial mapping M_i^S of keyword to schema terms generated in the previous step, the mappings of the remaining unmapped keywords to value terms needs to be decided. This is done in two phases. First, the intrinsic weights of the VW sub-matrix that were generated in Step 1 are updated to reflect the added value provided by the mappings in M_i^S of some of the keywords to schema database terms. This is called the process of contextualization of the VW sub-matrix. It is based on the documented observation that users form queries in which keywords specifying metadata information about a concept are adjacent or at least neighboring [18, 33]. Thus, when a keyword is mapped to a schema term, it becomes more likely that an adjacent keyword should be mapped to a value in the domain of that schema term. The contextualization process increases the weights of the respective values terms to reflect exactly that. For example, in the keyword query "`Name Alexandria`" assume that the keyword `Alexandria` was found during the first step to be equally likely the name of a person or of a city. If in Step 2 the keyword `Name` has been mapped to the attribute `Name` of the table `Person`, the confidence that `Alexandria` is actually the name of a person is increased, thus, the weight between that keyword and the value database term representing the domain of attribute `Name` should be increased, accordingly. In the second phase, given an updated VW_i sub-matrix, the most prominent mappings of the remaining unmapped keywords to value database terms are generated. The mappings are generated by using again the adapted technique of the Hungarian algorithm (ref. Section 7). The result is a series of partial mappings M_{ik}^V, with k=1..m_i, where i identifies the mapping M_i^S on which the computation of the updated matrix VW_i was based. Given one such mapping M_{ik}^V the value weights are further updated to reflect the mappings of the adjacent keywords to value database terms, in a way similar to the one done in Step 2 for the SW sub-matrix. The

outcome modifies the total score of each mapping M_{ik}^V, and based on that score the mappings are ranked.

Generation of the Configurations. As a fourth step, each pair of a mapping M_{ik}^V together with its associated mapping M_i^S is a total mapping of the keywords to database terms, forming a configuration C_{ik}. The score of the configuration is the sum of the scores of the two mappings, or alternatively the sum of the weights in the weight matrix of the elements $[i, j]$ where i is a keyword and j is the database term to which it is mapped through M_{ik}^V or M_i^S.

Generation of the Interpretations. Having computed the best configurations, the interpretations of the keyword query, i.e., the SQL queries, can be generated. The score of each such query is the score of the respective configuration. Recall, however, that a configuration is simply a mapping of the keywords to database terms. The presence of different join paths among these terms results in multiple interpretations. Different strategies can be used to further rank the selections. One popular option is the length of the join path [17] but other heuristics found in the literature [16] can also be used. It is also possible that a same interpretation be obtained with different configurations. A post-processing analysis and the application of data-fusion techniques [6] can be used to deal with this issue. However, this is not the main focus of the current work and we will not elaborate further on it. We adopt a greedy approach that computes a query for every alternative join path. In particular, we construct a graph in which each node corresponds to a database term. An edge connects two terms if they are structurally related, i.e., through a table-attribute-domain value relationship, or semantically, i.e., through a referential integrity constraint. Given a configuration we mark all terms that are part of the range of the configuration as "marked". Then we run a breath-first traversal (that favors shorter paths) to find paths that connect the disconnected components of the graph (if possible). The final SQL query is then constructed using the "marked" database terms, and in particular, the tables for its `from` clause, the conditions modeled by the edges for its `where` clause and the remaining attributes for its `select` clause. Then the process is repeated to find a different interpretation, that will be based on a different join path. The final order of the generated interpretations is determined by the way the different paths are discovered and the cost of the configuration on which each interpretation is based.

It is important to note here that if the thresholds used in the above steps are all brought down to zero, then our technique is able to generate all the possible interpretations that can be defined on a database, even the most unlikely. In that sense, our technique is complete. The thresholds serve only to exclude from the results any interpretation that is not likely to represent the semantics of the keyword query, while the weights are used to provide the basis for a ranking metric.

5. INTRINSIC WEIGHT COMPUTATION

To compute the intrinsic weights, we need to compute the relevance between every query keyword and every database term. Some fundamental information that can be used towards this directions, and that is typically available, is the schema information. It may include the table and attribute names, the domains of the attributes, and very often referential and integrity constraints, such as keys and foreign keys. Syntactic descriptions of the contents of an attribute (e.g., regular expressions) can also lead to a better matching of keywords to database terms since they offer indications on whether a keyword can serve as a value for an attribute or not. There are already many works that offer typical syntax for common attributes such as phone numbers, addresses, etc. [28], and have been used extensively and successfully in other areas. If access to the catalog tables is possible, assertion statements can offer an alternative source of syntactic information. In the same spirit, relevant values [5], i.e., clusters of the attribute domains, are also valuable auxiliary information. Furthermore, there is today a large volume of grammatical and semantic information that is publicly available on the Internet and can be used as a service. Examples include the popular WordNet and the many community specific ontologies.

Algorithm 1: Intrinsic SW Matrix Computation

Input: Q: Keyword Query
T: Schema Database Terms
Output: SW matrix

COMPUTEISW(Q,T)
(1) $SW \leftarrow [0, 0, \ldots, 0]$
(2) $\Sigma \leftarrow \{$ Synonyms(w,t), Hyponyms(w,t), Hypernyms(w,t), StringSimilarity(w,t) $\ldots\}$
(3) **foreach** $w \in Q$
(4) **foreach** $e \in T$
(5) $sim \leftarrow 0$;
(6) **foreach** $m \in \Sigma$
(7) **if** $m(w,e) > sim$
(8) $sim \leftarrow m(w,e)$
(9) **if** $ssim \leq threshold$
(10) $sim \leftarrow 0$;
(11) $SW[w,c] = ssim * 100$

5.1 Weights for Schema Database Terms

Finding matches between the flat list of keywords and the schema terms looks like the situation of schema matching [28] in which one of the schemas is the flat universal relation [23]. We follow a similar approach in which we employ a number of different similarity measurement techniques and consider the one that offers the best result. One of these techniques is the string similarity [12]. For the string similarity we further employ a number of different similarity metrics such as the Jaccard, the Hamming, the Levenshtein, etc., in order to cover a broad spectrum of situations that may occur. Since string similarity may fail in cases of highly heterogeneous schemas that lack a common vocabulary, we also measure the relativeness of a keyword to schema database term based on their semantic relationship. For that we employ public ontologies, such as SUMO[3], or semantic dictionaries such as WordNet, that can provide synonyms, hypernyms, hyponyms, or other terms related to a given word.

Algorithm 1 describes the computation procedure of the intrinsic schema weight matrix SW. The set Σ represents the similarity methods we employ. We have a number of default methods that represent the state of the art in the area, but additional methods can be included. Each such method takes as input two strings and returns their respective similarity in a range between 0 and 1. We trust the method that gives the highest similarity. If a similarity between a keyword and a schema term is found below a specific threshold (that is set by the application) then the similarity is set explicitly to 0. As a result, at the end of the procedure there might be rows in the matrix SW containing only zeros. These rows represent keywords that are not similar enough to any of the schema database terms, thus, they will be considered later as candidates for mapping to value database terms, i.e., domains of the schema attributes. The fact that their rows in the SW matrix are 0 instead of

[3]www.ontologyportal.org

some low value, makes the similarities of the keyword to the value database terms that will be computed in a later step to be the dominating factor determining the guess on the role a specific keyword can play.

EXAMPLE 5.1. *Consider the keyword query "workers department CS" posed on the database of Figure 1. Figure 4 illustrates a fraction of the weight matrix containing the intrinsic weights for the database terms derived from the tables* `Person` *and* `Department`. *Instead of the full names of tables and attributes, only the first letter of the tables and the first two letters of the attributes are used. The schema weights SW are the light gray colored part of the matrix. Note that the keyword* `CS` *has not been mapped to any of the schema terms since all the values of its row in SW are 0.*

5.2 Weights for Value Database Terms

For computing the intrinsic value weights, we mainly exploit domain information, and base our decision on whether a keyword belongs to the domain of an attribute or not. Furthermore, we have adapted the notion of *Semantic Distance* [11] that is based on results retrieved by a search engine in order to evaluate the relatedness of two concepts. In particular, we define the *semantic relatedness* $SR(x,y)$ of two terms x and y as: $SR(x,y)=e^{-2NXD(x,y)}$ where $NXD(x,y)=\{max\{logf(x),logf(y)\}-logf(x,y)\}/\{logN-min\{logf(x),logf(y)\}\}$ with $f(x)$ denoting the number of web documents containing x, and $f(x,y)$ the number of documents containing both x and y, as these numbers are reported by specific search engines such as Google, Yahoo!, Cuil, Excite!, etc. The number N represents the number of documents indexed by the corresponding search engine. For our purpose, we compute the semantic relatedness of every keyword - attribute domain pair and this gives us an indication of the similarity degree between the keyword and the attribute domain. Information about possible values that an attribute can accept is also an important factor. The information is based on the explicit enumeration of values, as in the *Relevant Values* approach [5]. When a keyword is found among (or is similar to) the valid values that an attribute can get, the keyword receives a high weight. Additional comparison techniques include semantic measures based on external knowledge bases.

EXAMPLE 5.2. *For the keyword query introduced in Example 5.1, the intrinsic value weights are indicated in the VW part of Figure 4 i.e., the dark gray-colored part. These weights have been computed by using domain knowledge and regular expressions. Note that these are the value weights, thus, the similarity is not between the keyword and the name of the attribute, but concerns the compatibility of the keyword with the attribute domain.*

6. CONTEXTUALIZATION

The process of contextualization, as previously explained, exploits the interdependencies across mappings of different keywords. There are three different forms of contextualization that we consider. The first one increases the confidence of a keyword corresponding to an attribute (respectively, relation), if an adjacent keyword is mapped to the relation it belongs (respectively, one of the attributes of the relation). This is based on the generally observed behavior that users may sometimes provide more than one specification for a concept in a keyword query. For instance, they may use the keyword `Person` before the keyword `Name` to specify that they refer to the name of a person. The second form of contextualization is similar to the first, but applies on the value weights instead of the schema weights. The third and most important contextualization form is the one that updates the confidence of certain keywords corresponding to value database terms based on the mappings of other keywords to schema terms. The process consists of three phases and takes as input the value weight matrix VW and a partial mapping of keywords to schema database terms, and returns an updated matrix VW. Let K be the ordered list of keywords in a query, M_i^S a partial mapping of keywords in K to schema terms, and K^S the subset of K containing the keywords for which M_i^S is defined, i.e., those that are mapped to some schema terms. We define the notion of *free trailing keywords* of a keyword k, denoted as $T(k)$, to be the maximum set $k_1, k_2, \ldots k_m$, of consecutive keywords in K that are between two keywords k_s and k_e in the query and for which k_s=k, M_i^S is defined for k_s and k_e and undefined for every k_i with i=1..m. The notion of *free preceding keywords* of a keyword k, denoted as $P(k)$, is defined accordingly, with k playing the role of k_e.

As an initialization step, all the weights in the rows of VW corresponding to keywords already mapped to database terms are set to zero. This is done to guarantee that in the three phases that are described next, none of these keywords will be mapped to a value term. In the first phase of the contextualization, for every keyword k mapped to a relation R through M_i^S, the weights of the trailing and preceding keywords $T(k)$ and $P(k)$ for terms representing the domains of the attributes of the relation R are increased by a constant value Δw. The rational of this action is that queries typically contain keywords that are generic descriptions for the values they provide. For instance, "`Person Bill`", "`Department CS`", etc., which means that consecutive keywords may correspond to a relation and a value of one of that relation's attributes. During the second phase, for every keyword k mapped to an attribute A through M_i^S, the weights of the trailing and preceding keywords $T(k)$ and $P(k)$ with the database terms representing domains of attributes in the same relation as A are also increased by a constant value Δw. The rational of this rule is that consecutive keywords may represent value specifications and related values. An example is the query "`Name Bill Databases`" intended to ask about the person Bill who works in the area of databases. In the third phase, if a keyword k is mapped to an attribute A, the weights of the trailing and preceding keywords related to domains of attributes

Algorithm 2: Contextualizing Value Weight Sub-Matrix VW

Input: M : Partial keyword assignment to schema database terms
Q : Keyword Query
SW : Schema intriscic weight sub-matrix
VW : Value intrinsic weight sub-matrix
Output: Updated VW sub-matrix

COMPUTECVW(SW,VW,M)
(1) **foreach** $k \in Q$
(2) $t \leftarrow x$: $\exists\,(k,x) \in M$
(3) **if** $t = null$
(4) **continue**
(5) **if** t is a relation R
(6) **foreach** attribute A of R
(7) **foreach** $k' \in T(k) \cup P(k)$
(8) $VW[k', \underline{R.A}] \leftarrow VW[k', \underline{R.A}] + \Delta w$
(9) **else if** t is an attribute A of a relation R
(10) **foreach** attribute A' of R with $A' \neq A$
(11) **foreach** $k' \in T(k) \cup P(k)$
(12) $VW[k', \underline{R.A}] \leftarrow VW[k', \underline{R.A}] + \Delta w$
(13) **foreach** attribute A' of R' s.t. $\exists$ join path from A' to A
(14) **foreach** $k' \in T(k) \cup P(k)$
(15) $VW[k', \underline{R.A}] \leftarrow VW[k', \underline{R.A}] + \Delta w$

	P.Na	P.Ar	P.Ph	P.Ad	P.Em	D.Id	D.Dn	D.Ad	D.Di
workers	0	0	0	0	0	0	0	0	0
department	0	0	0	0	0	0	0	0	0
CS	30	18	0	18	0	50	75	50	50

Figure 5: Contextualized Value Weights VW

related to A through some join path are increased by the constant value Δw. The rational is that users use keywords referring to concepts that are semantically related, even if these concepts have been modeled in the database in different tables. An example of this situation is the keyword query "`Phone Tribeca`". If `phone` is mapped to the attribute `Phone` of the relation `Person`, then the keyword `Tribeca` most likely represents the address of the department, which is stored in a separate table, and then the keyword query is about finding the phone number of the department that is on Tribeca street. Note that the weight increase Δw can also be a percentage, instead of a constant, but our experimentations have showed no significant differences. The three phases described above are illustrated in pseudocode in Algorithm 2.

EXAMPLE 6.1. *Figure 5 illustrates the VW sub-matrix after the value weights of Figure 4 have been contextualized based on the mapping that maps the keyword* `person` *into* `workers` *and* `department` *to* `department`. *Note that the lines for keywords* `person` *and* `department` *are 0, a result of the initialization step.*

7. SELECTION OF THE BEST MAPPINGS

Given a weight matrix, computing the best possible mapping of keywords to database terms is known as the *assignment problem* [9]. Unfortunately, the traditional solutions return the first best mapping while we would like them all in descending order, or at least the top-k. Furthermore, we need a solution that, during the computation of the best mappings, takes into consideration interdependencies of the different assignments, i.e., the contextualization process. For this reason, we have adapted the popular systematic Hungarian, a.k.a. Munkres, algorithm [7] in order not to stop after the generation of the best mapping but to continue to the generation of the second best, the third, etc. Furthermore, some of its internal steps have been modified so that the weight matrix is dynamically updated every time that a mapping of a keyword to a database term is decided during the computation.

The execution of the algorithm consists of a series of iterative steps that generate a mapping with a maximum score. Once done, the weight matrix is modified accordingly to exclude the mapping that was generated and the process continues to compute the mapping with the second largest score, etc. More specifically, the maximum weight of each row is first identified and characterized as *maximum*. If the characterized as maximum weights are all located in different columns, then a mapping is generated by associating for each of the characterized as maximum weights the keyword and the database term that correspond to its respective row and column. On the other hand, if there is a column containing more than one weight characterized as maximum, all maximums in the column except the one with the maximum value loose their characterization as maximum. This last action means that some of the rows are left without some characterized weight. The values of the weights in these rows are then updated according to a number of contextual rules mentioned in Section 4. This is the effect of the mapped keywords to those that have remained unmapped.

In the sequel, for each row with no characterized weight, the one with the maximum value that belongs to a column corresponding to a database term different from the previous one is selected and characterized as *maximum*. If this leads to a matrix that has each characterized weight in a different column, then a mapping is generated as above or the process of un-characterizing some of the values as previously described is repeated.

Algorithm 3: Keyword to db term mapping selection

Input: $I(i_{ij})$ where $I \equiv SW$ or $I \equiv VW$
Output: $M^I = \{M_1^I, \ldots, M_z^I\}$: Mappings generated by I

MAPPING(I, W_{MAX})
(1) $tempM = \bigcup i_{pt} \leftarrow HUNGARIAN_{Ext} * (I)$
(2) $W \leftarrow \sum i_{pt}$
(3) $M^I \leftarrow tempM$
(4) **if** $(W > c * W_{MAX})$
(5) $W_{MAX} \leftarrow W$
(6) **while** $(W > c * W_{MAX})$
(7) **foreach** $i_{pt} \in tempM$
(8) $i_{pt} \in I \leftarrow -100$
(9) $Mapping(I, W_{MAX})$

Once a mapping of all the keywords has been formed, it is included in the set of mappings that will be returned by the algorithm. Then, the algorithm needs to be repeated to generate additional mappings. To avoid recomputing the same assignments, the algorithm is re-executed cyclically with a new matrix as input. The new matrix is the old one modified to exclude mappings that have already been considered in previous runs of the algorithm. This is done by setting the values of the respective weights to a large negative number, forcing the algorithm to never select them again. This whole process is repeated until the scores of the mappings that the algorithm generates fall below some specific threshold. By construction, the most prominent mapping is the one that is first reported by this task.

The original Hungarian algorithm for rectangular matrices has an $O(n^2 * m)$ complexity [7], where n is the number of keywords and m is the number of databases terms. Extending the algorithm to consider dynamic weights as described above brings the complexity to $O(n^3 * m^2)$ which is due to the fact that a mapping may affect any other mapping, thus, in the worst case, $(n-1)*(m-1)$ weight updates may take place. Nevertheless, this worst case rarely happens since only a subset of the matrix is updated in each iteration, and, due to the threshold, not all the possible updates are evaluated.

Algorithm 3 depicts the overall process of computing the set of most prominent mappings of a set of keywords to database terms, given a weight matrix. The expression $HUNGARIAN_{ext}$ refers to our extended version of the Hungarian algorithm.

EXAMPLE 7.1. *Figure 6 illustrates a VW matrix similar to the one of Figure 5, but with additional keywords to better demonstrate the steps described in this section. The initial version of the matrix is the one composed of the first 7 lines. The lines of the keywords workers and department will remain unchanged since the keywords are mapped to schema terms, and for this reason they are omitted from the subsequent versions. The weights in white cells are those characterized as maximum. Each one is the largest weight in its row. Note that for column* `P.N` *there are more than one weight characterized as maximum. From those, only the largest one is kept, in our case the 46. This leaves the row of keyword* `Hopkins` *with no weight characterized as maximum. The result is the second VW matrix illustrated in Figure 6. The three characterized weights suggest a mapping for the keywords* `CS`, `Mary` *and* `Summerhill`.

	P.N	P.A	P.P	P.Ad	P.E	D.I	D.D	D.A	D.Di
workers	0	0	0	0	0	0	0	0	0
department	0	0	0	0	0	0	0	0	0
CS	30	18	0	18	0	50	75	50	50
Hopkins	45	30	0	35	10	32	40	35	39
Mary	46	20	0	5	7	20	25	30	40
Summerhill	10	7	0	34	22	20	10	32	15

CS	30	18	0	18	0	50	75	50	50
Hopkins	45	30	0	35	10	32	40	35	39
Mary	46	20	0	5	7	20	25	30	40
Summerhill	10	7	0	34	22	20	10	32	15

CS	30	18	0	18	0	50	75	50	50
Hopkins	49	34	0	39	14	34	42	37	41
Mary	50	24	0	9	11	22	27	32	42
Summerhill	14	11	0	38	26	22	12	34	17

CS	30	18	0	18	0	50	75	50	50
Hopkins	49	34	0	39	14	34	42	37	41
Mary	50	24	0	9	11	22	27	32	42
Summerhill	14	11	0	38	26	22	12	34	17

Figure 6: Weight matrix during best mapping computation

Given these mappings, the weights are adjusted to reflect the interdependencies according to the contextual rules. For instance, the mapping of `CS` *to the database term* `D.D`*, triggers an increase in the weights of the database terms on the attributes in the same table. The result of firing the contextual rules is the third matrix in Figure 6. In the updated matrix, the highest value is 49, which is in the column* `P.N`*, but cannot be chosen since the keyword* `Mary` *is already mapped to it. The same applies for the second and third largest weights in the row, which are 42 and 39, respectively. The fourth largest weight, 37, is in a column of a database term that no keyword is mapped to, and it is becoming characterized as maximum. The final outcome is the fourth matrix of Figure 6 and, based on this, the mappings of keywords* `CS`*,*`Hopkins`*,* `Mary` *and* `Summerhill` *to the database terms* `D.D`*,* `D.Di`*,* `P.N` *and* `P.Ad`*, respectively, which is added to the mapping results generated by the algorithm. After that, the mapping is again considered for generating a new input for the algorithm. Four new matrices are derived from it, each one having one of the weights that was in a white cell in the last matrix reduced. For each obtained matrix, the whole process starts again from the beginning.*

8. RELATED WORK

The popularity of keyword searching is constantly increasing. It has been the successful retrieval model in IR for text databases [30] and the web [8] for more than a decade. It has also been adopted by the data management community in the context of XML [13, 20, 32]. To answer a keyword query over XML data, the appearances of the query keywords are first identified within the documents (possibly through some free-text search) and then combined together into Meaningful Lowest Common Ancestor Structures (MLCAS). A score is computed based on this structure and according to this score the respective XML documents containing the MLCAS are ranked and returned to the user. XML benefits from the fact that its basic information model is the "document", which is the same as in IR, thus, many IR techniques can be easily employed in the XML context. On the other hand, keyword search in relational databases is particularly challenging [31], first because the database instances are way larger, and second because the basic model is fundamentally different making hard the identification of the information units that are to be returned. Nevertheless, keyword search over relational data is particularly appealing, and there are already

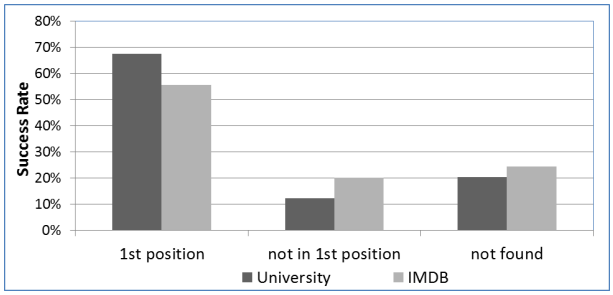


Figure 7: Success in finding the intended query semantics

many interesting proposals in the scientific literature [36, 10]. DISCOVER [16] and DBXplorer [2] have been among the first such systems. The typical approach is to build a special index on the contents of the database and then to use that index to identify the appearances of the query keywords in the attribute values. Many approaches use inverted indexes [1, 16, 29] for that purpose, while others, like DBXplore, use symbol tables. A symbol table is an index consisting of a set of triples $\langle$*value, attribute, relation*$\rangle$, used to map every value to its schema information. Once the keywords are located, the different ways by which their respective tuples are connected are discovered, forming the so-called "joining network" of tuples or tuple trees, that often become the information unit returned to the user. The joining networks are constructed either directly from the instances, or by building query expressions and evaluating them [16]. DISCOVER is interested in finding total and minimal joining networks. A joining network is total if each keyword query is contained in at least one tuple of the network, and minimal if the removal of any tuple makes the network no longer total. More recent approaches [27], are oriented towards reducing the number of tuples that need to be considered in order to improve previous techniques. BANKS [1] follows a similar approach but employs the Steiner tree algorithm to discover how the tuples are associated. SQAK [31] is another system that is based on the same generic principles but focuses on the discovery of aggregate SQL expressions that describe the intended keyword query semantics. Since the set of possible answers may be large, and since the results are already ranked, the above systems typically return the top-k results.

As happened in all the above approaches, we are also trying to identify the possible semantics (i.e., expected answers) to the ambiguous keywords queries, and rank the results. We return the top-k but by bringing the thresholds down to their extreme values we can provide the whole set if needed. Our fundamental difference is that we do not assume a-priori access to the database instance, as our approach do not need pre-build and exploit any index. Unavoidably, we are based on schema and meta-data, i.e., intensional information, which makes our work applicable to scenarios where the above techniques cannot work. Nevertheless, our approach should not be seen as an alternative to the above methods. Since we operate on different information, i.e., the meta-data, our work can be used to enhance these techniques by providing a better exploitation of the meta-information and the relationships among the keywords. The idea of using schema information and keyword semantics has been considered in one of the approaches [21], but this is only limited to word semantics based on WordNet. We go further by combining not only additional semantic similarity techniques, similar to those used in schema matching [28], but also on syntactic and structural information. Data behavior has also been considered [31]. All these works are complementary.

Another distinction of our approach is on the relationship among

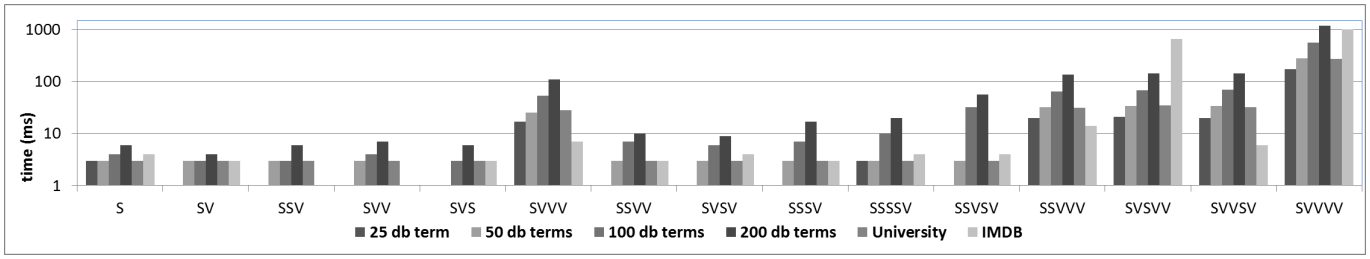


Figure 8: Time performance for different database sizes and different kinds of queries

the keywords. Existing approaches compute the relationship among the keywords by considering the relationships of the data values in the database instance. We believe that the keyword query should be the one driving the translation and not the data. Our approach takes into consideration in a systematic way the position of the keywords in the query itself and the inter-dependencies between the matchings of the keywords to the database structure. Furthermore, we map keywords to schema elements that will form the SQL query, instead of mapping them to the tuples themselves [31]. This allows us, without performing any query evaluation, to present these queries to the user and communicate the considered semantics. Of course, this is an optional step, but in highly heterogeneous environments, this is the norm for matching and mapping techniques [26].

A related field is query relaxation [3], but the main difference is that queries are expected to provide some form of structure specification while keyword queries, which are our interest, are completely flat. The same applies for approximate query answering techniques [14, 15, 20]. The latter may not map all the parts of the query, an assumption that for the moment we also have. Keyword search has also been considered by the Semantic Web community for RDF and linked data [33, 34, 37, 38]. The research issues and the approaches are similar to the keyword search in relational systems, but in the context of the triple-based RDF data.

Our effort can also find applications in the field of graphical tools that assist the user in formulating queries [25]. By finding the different interpretations of a keyword query, we could detect related schema structures, make suggestions and guide the user in the query formulation. Furthermore, in cases of exploratory searches, the user can use the generated interpretations as a way to explore an (unknown) data source and understand better its semantics.

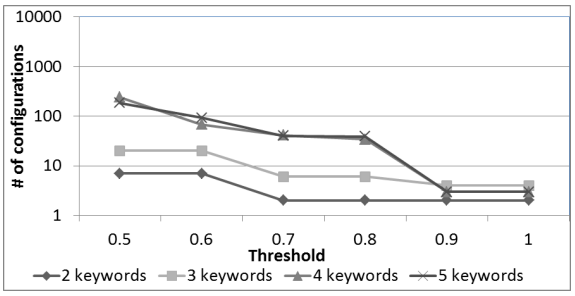
9. EVALUATION

We implemented the proposed technique into the Keymantic system [4] and we run numerous experiments for studying its efficiency and effectiveness. The experiments were performed on a 32bit Centrino Duo vPro Windows machine with 3GB of RAM. Our work is intended for applications that heavily depend on the user semantics, thus, generating synthetic dataset was not an effective alternative. We instead selected for our experiments two real data sets. The first was a university database containing information about courses, professors, students, publications, employees and other academic issues. The second database was a fraction of the popular IMDB service that is publicly available from the respective web site (www.imdb.com). 29 real users were asked to provide a set of keyword queries for these databases alongside a natural language explanation of what they were looking for. The users had no access to schema information, they neither were technical experts. Only a verbal high level explanation of what the database contents were about had been provided to them. A database expert translated each natural language explanation of what the user was looking as it had been provided by her, into SQL. Since many queries were vague, high level and of exploratory nature, some of these queries were translated into more than one SQL query. The answer expected for each keyword query was the result of evaluating the SQL queries that the expert had created, and this was used as a reference to evaluate the results returned by our implementation. We used a total of 99 and 44 queries for the university and the IMDB database, respectively. It is important to note that evaluating a keyword search technique for relational databases is a challenging task, mainly due to the lack of standards [35]. The task is becoming more challenging in our case since existing techniques have different assumptions and goals, making sometimes unfair any direct comparison among them. One of the reasons is that most of these techniques assume the availability of the instance.

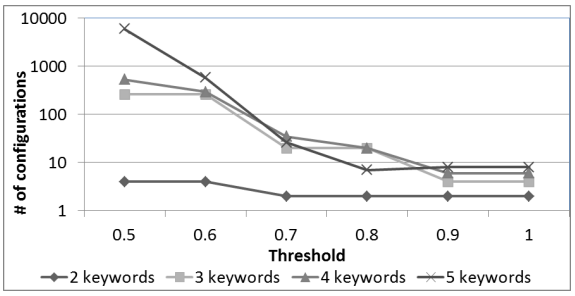
[The importance of Schema Information] Despite the existence of many keyword search techniques, few exploit schema information. We would like to study whether our claim that, a significant number of keywords in keyword queries correspond, not to the data, but to meta-data (schema) information, is true. For that, we studied the SQL answers to the keywords queries of our evaluation dataset, and found that 58% of the keywords were actually referring to schema terms.

[Effectiveness] To measure the effectiveness of our technique we tested whether the queries that generate the intended answers (as specified by the users and created by the expert) were among those our algorithm generates. We also measured how many times the expected results appear in the first place. In case a keyword query had an answer generated by more than one SQL query, we considered in the counting multiple copies of the keyword query each one with only one of these SQL queries. The threshold we used, i.e., the parameter c in Algorithm 3, definitely played an important role. When it was brought to zero, every expected answer to a keyword query was included in the result. The percentage of keyword queries that returned the expected answer in the first position was not much different from the one in the case of a threshold with some default value, thus, the results we report here are for that default value. The results are graphically illustrated in Figure 7. The “1st position" refers to the percentage of the keyword queries in which we found the expected interpretation in the first position. The “not in 1st position" indicates the percentage of the queries in which our algorithm did generated the expected interpretation but did not return it in the first position. Finally, the “not found” indicates the percentage in which the expected interpretation was not returned at all in the results produced by our algorithm. In the IMDB scenario, we obtained worst results than in the university scenario. This was mainly due to the fact that the IMDB schema consists of many nested tables with the same structure, i.e., same name and very similar data-types.

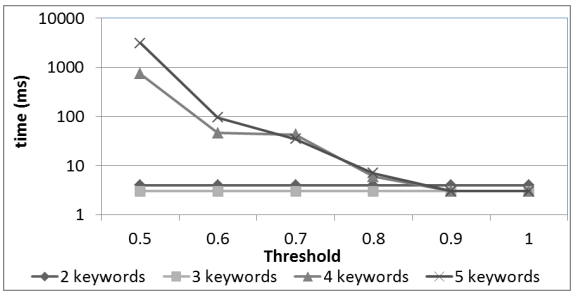
[Efficiency] To measure the efficiency of our system, we studied



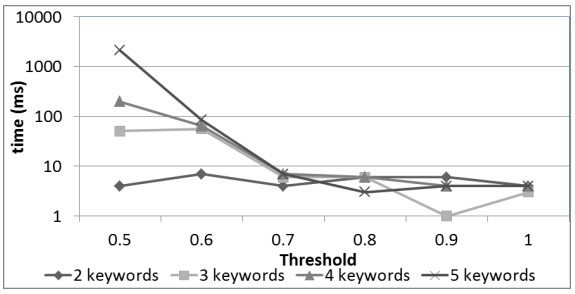
(a) Generated configurations for the University database



(b) Generated configurations for the IMDB database



(c) Time performance for the University database



(d) Time performance for the IMDB database

Figure 9: Experimental results using different thresholds

its time performance with respect to two different parameters: the queries and the size of the database. For the former, we considered queries that differ not only on the number of keywords, but also on their type, i.e., whether the keywords correspond to schema or value terms. For the latter, since our approach assumes no access to the instance, the variation in size was considered in terms of the size of the schema. We created fractions of different sizes from our original evaluation database. Our performance measurements are illustrated in Figure 8. Each query is characterized by a string. Each character of the string corresponds to a keyword in the query and can be S, if the respective keyword is intended to be a schema term, or V, if it is intended to be a value term. For instance, the string SVS represents a keyword query in which the first and last keyword represent schema terms while the middle a value. As expected, the response time increases with the number of keywords, but when there is a prevalence of keywords mapped to schema terms this increase is not dramatic. The interesting result is the increase in time due to the increase of the number of keywords representing value terms. In general, this increase should have been factorial (ref. Section 3), but the experiments showed no such increase, which is because the metadata and the auxiliary information alongside the contextual rules are significantly restricting the number of possible mappings of keywords to database terms. Only in the last case, where there was one keyword referring to a schema element and four keywords referring to data values, the response time increased. The comparison between the IMDB database and the university database showed no significant difference, which means that the nesting level of the database schema did not really affect the time performance. In the experiments above, we did not report the time needed to actually evaluate the interpretations, i.e., queries, to avoid having the query engine performance blurring the results. We did measured, however, the performance as well as the overall time needed to answer a keyword query. We noticed that the overall time was highly related to the number of results retrieved and the number of join operations required. It ranged from some tenths of a second for selective queries involving one table to a few minutes for general queries. For example, in the IMDB scenario, the keyword query "`film`" took 1.9 sec and retrieved 166936 records, while the query "`actor film 2007`" took 1 min and 40 sec and retrieved about 2.7 million tuples.

[Algorithm Configuration] To evaluate the effect of the threshold value in the behavior of our system, we have experimented with the same keyword query but with different threshold values. As expected, a selective, i.e., high, threshold reduces the number of results and the response time. Figure 9 provides an insight on the dependencies between these two factors. A study of the results allows us to observe that different number of configurations can be obtained for the same threshold value. The reason of this discrepancy is mainly due to the different keyword queries with which the scenarios were evaluated and the different schemas of their respective databases. In particular, the university database is composed of few tables with a large number of attributes and the IMDB database contains many related tables with few attributes.

[Qualitative comparison to other approaches] Our approach is the first to operate under the assumption that there is no prior access to the relational database instance. Under such a condition, existing works will return no results since they will have no way to determine where the keywords appear. Even works that take schema information into account [21] do so as an optimization step after they find the appearance of some keywords in the database instance. Thus, a direct comparison in terms of effectiveness and efficiency would be unfair. Nevertheless, to realize a qualitative comparison, we run some experiments using two popular systems for keyword searching in relational databases, DBXplorer [2] and DISCOVER [16], alongside our system. For DISCOVER, we had exactly the same results as in DBXplorer, thus, we report only the former. The first experiment we run was with DBXplorer assuming that the relational database was not allowing access to its instance so that DBXplorer could not create its index. As such, all the queries we executed returned zero results. Then, we allowed such access, and repeated the experiment. We run the same queries

against our system, naturally assuming no access to the instances. The number of queries generated by our system was larger than the one generated by DBXplorer. This was expected since our system was using no information about the existing data values and was making more generic guesses of possible semantically meaningful queries. We also tried what happens if our system did have information about the instance values. This allowed it to identify and eliminate the queries generated by mappings of certain keywords to value terms that were leading to zero results. The number of queries generated by our system was less than the number of queries generated by DBXplorer since in our case some queries that were not very likely to occur semantically were eliminated by the threshold (the parameter c in Algorithm 3). We obtained the same number of queries if the threshold was set to zero.

10. CONCLUSION

We have presented a novel framework for keyword search in relational databases. In contrast to traditional keyword search techniques that require access to the actual data stored in the database in order to build an index, our technique uses intensional knowledge. This allows our approach to be used in a wide range of applications, including, besides databases on the web, information integration systems, where building and maintaining specialized indexes is not a feasible option, because the only service offered by the source is to access the data through wrappers, predefined queries or web forms. For the challenging task of interpreting the keyword queries, we have extended the Hungarian algorithm in order to find the SQL queries the most likely describe the meaning of the keyword queries. We have implemented and evaluated the technique and reported our findings.

REFERENCES

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, and S. Sudarshan. Banks: Browsing and keyword searching in relational databases. In *VLDB*, pages 1083–1086, 2002.

[2] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16. IEEE Computer Society, 2002.

[3] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. FleXPath: Flexible structure and full-text querying for XML. In *SIGMOD*, pages 83–94. ACM, 2004.

[4] S. Bergamaschi, E. Domnori, F. Guerra, M. Orsini, R. T. Lado, and Y. Velegrakis. Keymantic: Semantic keyword-based searching in data integration systems. *PVLDB*, 3(2):1637–1640, 2010.

[5] S. Bergamaschi, C. Sartori, F. Guerra, and M. Orsini. Extracting relevant attribute values for improved search. *IEEE Internet Computing*, 11(5):26–35, 2007.

[6] J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1), 2008.

[7] F. Bourgeois and J.-C. Lassalle. An extension of the Munkres algorithm for the assignment problem to rectangular matrices. *Communications of ACM*, 14(12):802–804, 1971.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[9] R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM Society for Industrial and Applied Mathematics, 2009.

[10] S. Chakrabarti, S. Sarawagi, and S. Sudarshan. Enhancing search with structure. *IEEE Data Eng. Bull.*, 33(1):3–24, 2010.

[11] R. Cilibrasi and P. M. B. Vitányi. The google similarity distance. *IEEE TKDE*, 19(3):370–383, 2007.

[12] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, 2003.

[13] D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into xml query processing. In *BDA*, 2000.

[14] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML joins. In *SIGMOD*, pages 287–298, 2002.

[15] S. Helmer. Measuring the Structural Similarity of Semistructured Documents Using Entropy. In *VLDB*, pages 1022–1032. ACM, 2007.

[16] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[17] Y. Kotidis, A. Marian, and D. Srivastava. Circumventing Data Quality Problems Using Multiple Join Paths. In *CleanDB*, 2006.

[18] R. Kumar and A. Tomkins. A Characterization of Online Search Behavior. *IEEE Data Engineering Bulletin*, 32(2):3–11, 2009.

[19] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246. ACM, 2002.

[20] Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, pages 72–83, 2004.

[21] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.

[22] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD*, pages 115–126. ACM, 2007.

[23] D. Maier, J. D. Ullman, and M. Y. Vardi. On the Foundations of the Universal Relation Model. *ACM Trans. Database Syst.*, 9(2):283–308, June 1984.

[24] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128. IEEE Computer Society, 2002.

[25] A. Nandi and H. V. Jagadish. Assisted querying using instant-response interfaces. In *SIGMOD*, pages 1156–1158. ACM, 2007.

[26] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating web data. In *VLDB*, pages 598–609, 2002.

[27] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: the power of rdbms. In *SIGMOD*, pages 681–694. ACM, 2009.

[28] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[29] A. Simitsis, G. Koutrika, and Y. E. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *VLDB Journal*, 17(1):117–149, 2008.

[30] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *SIGIR*, pages 21–29, 1996.

[31] S. Tata and G. M. Lohman. SQAK: doing more with keywords. In *SIGMOD*, pages 889–902. ACM, 2008.

[32] M. Theobald, H. Bast, D. Majumdar, R. Schenkel, and G. Weikum. TopX: efficient and versatile top-k query processing for semistructured data. *VLDB Journal*, 17(1):81–115, 2008.

[33] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE*, pages 405–416. IEEE Computer Society, 2009.

[34] V. S. Uren, Y. Lei, and E. Motta. Semsearch: Refining semantic search. In *ESWC*, pages 874–878. LNCS Springer, 2008.

[35] W. Webber. Evaluating the effectiveness of keyword search. *IEEE Data Engineering Bulletin*, 33(1):55–60, 2010.

[36] J. X. Yu, L. Qin, and L. Chang. *Keyword Search in Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[37] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl. From keywords to semantic queries-incremental query construction on the semantic web. *Journal of Web Semantics*, 7(3):166–176, 2009.

[38] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. Spark: Adapting keyword query to semantic search. In *ISWC*, pages 694–707, 2007.