# OBDBSearch: A Keyword-based Semantic Search for Databases

Simone Miraglia*, Clarissa Molfino† and Costanza Romano‡
DIBRIS - Università degli Studi di Genova
Genoa, Italy
Email: *simone.miraglia@gmail.com, †clarissa.molfino@gmail.com, ‡costanza.romano.cr@gmail.com

***Abstract*—In this paper, we consider a semantic approach to information retrieval in structured data. We designed an ontology-based database search tool, namely OBDBSEARCH, that exploits ontologies to add semantics to traditional database-style searching. Our goal is to retrieve all database information somehow related to a set of keywords and to have them sorted by relevance. A heuristic method to calculate relevance has been devised. In order to apply and test our algorithm, we considered an application to Key Performance Indicators (KPIs), i.e., quantifiable and strategic measurements for an organization. We observed consistency between experimental results and those expected, even if testing on large amounts of data is still missing.**

***Keywords–Keyword-based search; Semantic search; Information retrieval; Ontologies; Database access.***

## I. INTRODUCTION

*Information retrieval concerns with all the activities related to the organization of, processing of, and access to, information of all forms and formats. The objective of an information retrieval system is to enable users to find relevant information from an organized collection of documents* [1]. As pointed out in [2], in response to various challenges of providing information access, the field of information retrieval evolved to give principled approaches to search various forms of content. Information retrieval is fast becoming the dominant form of information access, overtaking traditional database-style searching. Since semantic web [3] is a growing field and information retrieval is still evolving - but database-style searching has not been overtaken yet - we sensed the need for a semantic approach to information retrieval in structured data.

This paper addresses the problem of retrieving information using ontologies to add semantics to traditional database-style searching. In particular, we are given a database, an ontology describing the underlying domain of the database and a set of query keywords provided by users. We are interested in retrieving and sorting by relevance all the database records which are related to all the keywords.

We developed an ontology-based database search algorithm, namely OBDBSEARCH, suitable for databases with textual information. It easy to integrate and provides an original heuristic ranking system to sort records. A case study has been devised to apply and test such algorithm, investigating the context of Key Performance Indicators (KPIs), i.e., quantifiable and strategic measurements that reflect an organizations critical success factors [4]. The main contributions of our work are:

- A novel keyword-based semantic search algorithm for databases.
- An easy to integrate and database schema-independent algorithm.
- An innovative heuristic ranking system to sort records by relevance.

This paper is structured as follows. Section II reviews related work, pointing out differences with our contribution. Section III of this paper gives a brief overview of ontologies. Section IV describes our case study, introducing what KPIs are and how our database and ontology have been designed. This should improve the understanding of our algorithm. In Section V, we introduce the problem we are tackling. Modeling and notation formalisation are given in Section VI. Section VII is devoted to the presentation of our algorithm. Section VIII is dedicated to the performance evaluation of our algorithm. Section IX summarizes the results of our work.

## II. RELATED WORK

Semantic search, as defined in [5], is a search paradigm that makes use of explicit semantics to solve core search tasks, i.e., to use semantics for interpreting query and data, matching query against data and ranking results. As shown in [5], search is commonly seen as an user-oriented application, which uses user-friendly interfaces instead of complex structured queries. The success of commercial search engines shows that users are more comfortable with keyword-based interfaces.

We focus on approaches which express the information needs as keywords. Several approaches in semantic search are based on translating a given set of query keywords into a set of conjunctive queries, which are evaluated with respect to an underlying knowledge base. Examples of this approach can be found in engines such as Hermes [6], SemSearchPro [7], SPARK [8], AVATAR [9], CIRI [10] or in [11]. The semantic approach we propose differs from the above engines since we aim to retrieve data from databases instead of knowledge bases and no conjunctive queries are formulated. We sense that a semantic approach in database search is missing and our work goes in that direction.

A semantic approach closed to ours is the one proposed in [12]. The authors developed a semantic search engine based on query refinement powered by ontology navigation for traditional information retrieval purposes. The ontology is regarded purely as a taxonomy, where focalization and generalization are used to refine queries. Furthermore, results are ranked by relevance according to standard information retrieval techniques, such as tf/idf, while our approach uses a heuristic method suitable for textual records.

YACOB [13] is a tool that uses domain knowledge in the form of concepts and their relationships for formulating and processing queries. The idea behind this approach is similar to ours.

Substantial work has been done in the field of ontology-based database access [14], such as [15] or [16]. Our work differs from these approaches because databases and ontologies are used togheter to search, but they are clearly separated and database access is performed in a traditional fashion via SQL

 ISBN: 978-1-61208-408-4 

queries. This should ease integration with existing databases.

If we left out semantics, our approach would be similar to the one of DBXplorer [17], a system that enables keyword-based search in relational databases. Given a set of keywords, DBXplorer returns all rows, either from single tables or joined tables, such that all the rows contain all the keywords. They introduced a symbol table, which is used at search time to determine the locations of query keywords in the database. Results are ranked by the number of joins involved.

## III. ONTOLOGIES

According to [18], ontologies are means to formally model the structure of a certain system we are interested in. From its observation, relevant entities and relations could emerge. Entities are organized in *concepts* and *relationships*. The backbone of an ontology is the generalization/specification hierarchy of concepts, i.e., a taxonomy.

Ontologies could be defined as *formal, explicit specification of a shared conceptualization* [19].

A conceptualization is an *abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly* [20]. In other words, a conceptualization is a list of concepts relevant to the description of our domain. Such conceptualization should be expressed in a computer readable *formal* language, with proper definitions. *Shared* means that several parties should agree with the formal conceptualization. This helps large-scale interoperability.

One of the most popular languages for defining ontologies is OWL 2 (Web Ontology Language Ver. 2) described in [21] and supported by many tools. It points out that there are two main types of relationship, *is-a* and *object property*. The former implements the generalization/specification hierarchy, while the latter allows any other type of relationship between instances of two concepts, such as *has* or *is related to*. One of the many features of [21] is the usage of *Annotation properties*, useful to add literals, i.e., strings, to concepts.

## IV. CASE STUDY

Now, we will briefly give an overview of KPIs and then we will explain the design process of both the database and ontology.

### A. KPIs background

KPIs are defined as *quantifiable and strategic measurements that reflect an organization's critical success factors*. KPIs are very important for understanding and improving manufacturing performance; both from the lean manufacturing perspective of eliminating waste and from the corporate perspective of achieving strategic goals [4]. Several standards exist, but we focused on just two: ISO 22400-2 [4] and SCOR rev.11 [22]. Both describe KPIs for manufacturing operations management.

In more detail, ISO 22400-2 defines KPIs by a standard schema that contains: formula, time behaviour, unit/dimension, rating, the user group where the KPIs are used and to what production methodology they fit, as shown in [4].

SCOR is a reference model that links business process, metrics, best practices and technology into a unified structure to support communication among supply chain partners and to improve the effectiveness of supply chain [22]. Therefore, metrics are related to processes. Processes are organized in a hierarchical structure with three levels, each one more specialized than the previous. KPIs are also organized in a hierarchical structure with several levels, i.e., KPIs residing at a given level are calculated through KPIs from lower levels. These are the main features of SCOR, others can be found in [22].

### B. Database design

In order to design a database to manage KPIs described by different standards, such as SCOR and ISO 22400-2, a common structure for KPIs has been designed. This common structure contains the most important fields of the two standards (KPI Code, Name, Description, Formula, Unit of Measure, Trend and Timing) and a note field for various other information.

Database design is oriented towards SCOR since it has additional features compared with ISO 22400-2. This means that the hierarchical structure of both processes and KPIs has been preserved, although in different ways. Process hierarchy has been directly maintained in our database design, i.e., three tables have been created, one for each hierarchical level. Conversely, since KPI hierarchy means that an upper level KPI is calculated through lower levels KPIs, we decided to have a single table with a recursive relationship to ensure links between KPIs.

Regardless of their level, KPIs can refer to processes of any level. Instead of linking the KPI table with all processes tables, for the sake of simplicity, we decided to link just KPIs and third level processes. In this way, we would preclude the possibility of linking first and second level processes to corresponding KPIs. To ensure that first and second level processes are linked to KPIs anyway, we introduced *default processes*. They are factitious processes which stand for corresponding processes of superior levels. This way, instead of linking KPIs with first or second level processes, we link KPIs to third level default processes.

Figure 1 shows the E/R diagram of our database, pointing out our design choices. We will now describe all our entities.

`standards` - this table allows us to consider several standards of KPIs, with the ID, name and description.

`performances` - this table contains performances, with their ID, name and description.

`kpi` - this table keeps track of KPIs, with their name, ID, formula, description, unit of measure, trend, code, timing, standard and performances.

`kpi_hierarchy` - this table shows the recursive relationship that allows us to define the KPI-hierarchy, listing the IDs of KPI-parents and KPI-sons

`processes_lv1` - this table keeps track of processes of the first level, with their name and description.

`processes_lv2` - this table keeps track of processes of the second level, with their name and description and the IDs of the processes of the first level. There are also first level default processes.

`processes_lv3` - this table keeps track of processes of the third level, with their name and description and first and second level default processes.

`kpi_processes` - this table realises the many-to-many

relationship between KPIs and processes of third level. It associates a KPI to a process, maintaining their IDs.

`input_processes` - Each process produces documents to be consumed by another one, i.e., a workflow exists. That means that every process becomes input (output) process for other processes. This table keeps track of input processes, with their IDs, documents and IDs of the processes they are input for.

`output_processes` - This table keeps track of output processes, with their IDs and IDs of the processes they are output for.

### C. Ontology for KPIs

Afterwards, an ontology has been implemented in order to represent the domain of KPIs in a supply chain context, considering also the structure of the database. OWL 2 [21] has been used to define the ontology.

Figure 2 shows the ontology, pointing out the structure of KPIs domain. Obviously, it does not represent the whole KPIs domain, it is just a small example. Now, we will describe it in more detail.

*KPI* - this entity refers to the concept of KPI.

*Process* - this concept refers to supply chain processes. Processes are linked to KPIs and thus on the ontology we have a connection between concepts related to them.

*Plan*, *Source*, *Make*, *Deliver*, *Enable*, *Return* - these entities are related to the main processes individuated by the SCOR standard [22]. Therefore, they are linked to processes by an *is-a* relationship.

*Sphere* - this concept refers to the scope of KPIs application. It is linked to KPIs by a *has* relationship.

*Time*, *Cost*, *Quality*, *Energy* - these entities are related to four different scopes. They are useful because some indicators mainly refer to costs, whereas others point out timing aspects of supply chain and so on. They are linked to *Sphere* by an *is-a* relationship.

*Industry* - this concept refers to the industrial sector of KPIs.

*Continuous*, *Discrete* - these entities are related to the two types of industrial process. They are linked to *Industry* by an *is-a* relationship.

*Textile* - this concept refers to a certain kind of discrete industry and is linked to *Discrete* by an *is-a* relationship.

*Pharma*, *Alimentary* - these concepts refer to two types of continuous industrial process and are linked to *Continuous* by an *is-a* relationship.

*Flour*, *Food&Beverages* - these entities are related to two possible types of food industry and are linked to *Alimentary* by an *is-a* relationship.

As shown later in Section V, synonyms need to be managed. In the ontology, for the sake of simplicity, we keep track of synonyms by means of *Annotation properties*, not listed in Figure 2.

## V. The problem

The problem we are addressing is the retrieval of information from a database given a set of keywords. In other words, we want to return a list of information related to *all* those keywords. Furthermore, it is desirable to have such information sorted by relevance. Our algorithm needs an existing database - preferably with textual information - and an ontology describing the underlying domain.

Our aim was to develop a schema-independent algorithm, both from database and ontology point of view. Furthermore, we expect the algorithm to navigate between tables and concepts of the ontology. Those issues reveal a need for abstraction, which is addressed in Section VI.

In order to devise a *flexible* search algorithm, synonyms need to be managed. In fact, a search keyword might not appear in the database as it is, but as one of its synonyms: if they were not tracked, these results would be lost. We can take the keyword *KPI* as an example: this word can be found in the database as it is or perhaps as *Performance Indicator*.

Since our results should be listed sorted by relevance, we should define what relevance means in order to be able to discriminate whether a certain result is more or less relevant than another one. In Section VII-D, we give a thorough description of the issue, devising a solution.

## VI. Modeling and notation

In this section, we want to introduce some notation, useful to describe the algorithm.

We can easily think of a database as a graph, since it is none other than a set of tables linked by relationships. This is true assuming it is well normalized, i.e., all many-to-many relationships between a certain table A and a table B have been transformed with two one-to-many relationships from A to C and from B to C, where C is a newly created table storing at least both primary keys of A and B [23].

Database metadata contain information regarding the structure of the database, i.e., tables, fields, foreign keys constraints. Since they can be queried, we can get all information needed to build straightforwardly the graph. This solves the problem of having a schema-independent database because, once the graph is created, we will only reason in terms of tables and links, forgetting about the actual schema.

A similar approach is also applicable to ontologies, since they are basically a set of concepts linked by relationships. No particular assumptions have to be made to the ontology structure. We can also individuate two types of edges/relationships, i.e., is-a relationships and object properties. Building a graph is easy, since state-of-art programming libraries provide access to ontologies, such as Apache Jena.

A graph structure is useful for navigation between nodes, both database tables and ontology concepts, thanks to many accomplished algorithms, such as Depth First Search (DFS) and Breadth First Search (BFS)[24].

Therefore, we can define a database $\mathcal{DB}$ and an ontology $\mathcal{O}$ as

$$\mathcal{DB} := (\mathcal{T}, \mathcal{L}) \quad \mathcal{O} := (\mathcal{C}, \mathcal{E}) \tag{1}$$

where $\mathcal{T}$ is a set of tables and $\mathcal{L}$ a set of links between tables, while $\mathcal{C}$ is a set of concepts and $\mathcal{E}$ a set of relations among concepts. $\mathcal{E}$ can be subdivided in two disjoint sets $\mathcal{H}$ - is-a relationships - and $\mathcal{P}$ - object properties.

A link $l \in \mathcal{L}$ is defined just as pair of tables and a unique name. It is a matter of fact that multiple links between the same pair of tables could exist. As shown in Figure 1, `kpi_hierarchy` and `kpi` are connected twice. The unique name is useful to discriminate links between the same pair of

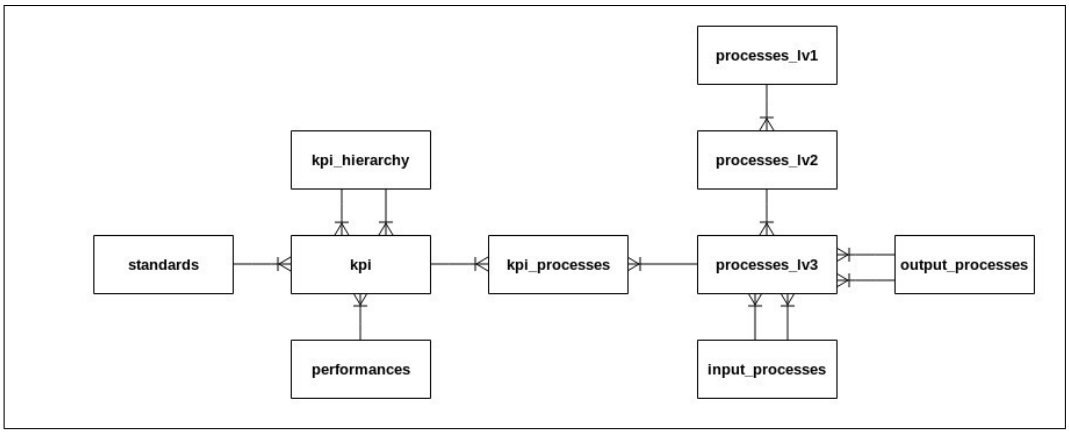


Figure 1. E/R diagram for KPIs. Boxes are entities, i.e., database tables, while lines between entities represent one-to-many relationships. Trebled end point indicates where the *many* is.

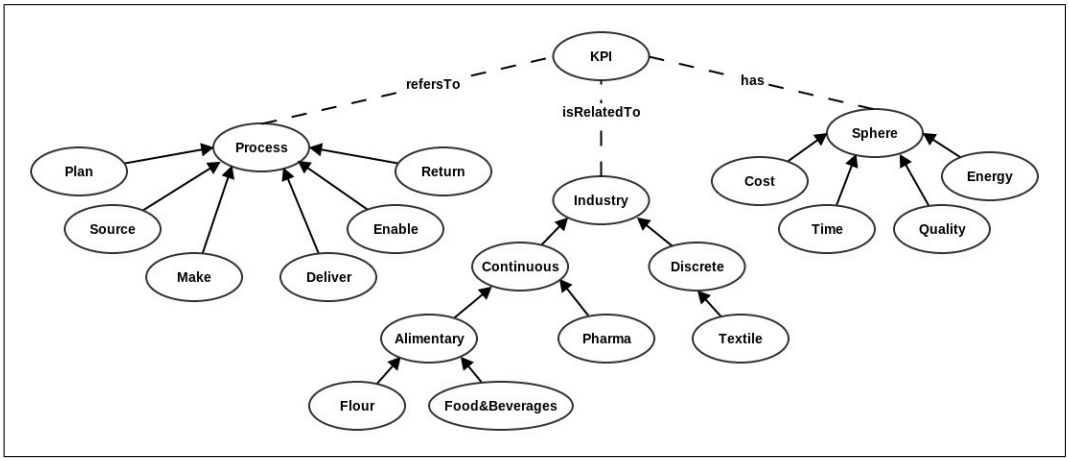


Figure 2. Designed ontology for the KPI context. Arrow lines refer to is-a relationships, e.g., *Plan* is-a *Process*, while dotted lines indicate object properties.

tables.

$$l := (\text{name}, t_1, t_2) \tag{2}$$

Let $t \in \mathcal{T}$ be a table, defined as follows

$$t := (\text{name}, R) \tag{3}$$

where name is table name and $R$ a set of records.

We use a function GETLINKS $(t)$: given a table, returns a set of links involving $t$. In addition, we define another function GETLINKEDTABLE $(l, t)$: given a table and a link containing that table, returns the linked table.

A record $r \in \mathcal{R}$ is defined as follows

$$r := (\mathcal{F}) \tag{4}$$

where $\mathcal{F} = \{f_1, \ldots, f_m\}$ is a set of fields.

A field $f_i$ can be defined as

$$f_i := (\text{name}, \text{type}, \text{key}, \text{value}) \tag{5}$$

where name and type are trivial to understand, key indicates whether $f_i$ is a primary key field or not and value is the actual value of the field.

We can further individuate a subset $\mathcal{LT} \subset \mathcal{T}$ containing all tables added to the database while transforming all many-to-many relationships into two one-to-many relationships.

Similarly, let $c \in \mathcal{C}$ be a concept, defined as follows

$$c := (\text{name}, \mathcal{S}) \tag{6}$$

where name is concept name and $\mathcal{S} = \{s_1, \ldots, s_k\}$ a set of synonyms.

We use a function GETNODE($\mathcal{O}$, name): given an ontology and a name, returns a concept $c$ such that $c$.name is equal to name.

A relationship $e \in \mathcal{E}$ is defined just as a pair of concepts

$$e := (c_1, c_2) \tag{7}$$

Furthermore, we can use three functions GETPARENTS($c$), GETCHILDREN($c$) and GETHORIZLINKS($c$): given a concept, they return a set of parent concepts, a set of children concepts and a set of concepts linked to $c$ by object properties, respectively.

## VII. THE ALGORITHM

Given a set of keywords $\mathcal{K}$, our goal is to retrieve a list of *results* relevant to all $k_i \in \mathcal{K}$ and have this list sorted by relevance. For the sake of simplicity, we assume that a *valid keyword* $k$ must be a name of any concept of the ontology,

 ISBN: 978-1-61208-408-4 

not one of the synonyms. In this section, we discuss how we achieved such goal.

First, we define a *result* as a record $r \in \mathcal{R}$ of any tables of $\mathcal{DB}$ such that is *relevant* to a certain keyword. We are looking for all those results relevant to all $k_i \in \mathcal{K}$. We further discuss in depth the relevance, but now say that a record is relevant to a certain keyword $k$ if $k$ or one of its synonyms is somehow contained in at least one of the fields of the record. This means that we should only consider fields $f_i : f.\text{type} \in \{\text{varchar}, \text{text}\}$.

More formally, given a table $t \in \mathcal{T}$, a record $r$ and a concept $c$ linked to $k$, we say that $r$ is relevant to $k$ iff $\exists\, f_i \in \mathcal{R}$ s.t.

$$k \in f_i.value \ \vee\ \exists\, s \in c.\mathcal{S} \ :\ s \in f_i.value \tag{8}$$

Let $RS_i$ be the set of *heterogeneous* results related only to $k_i$. We say *heterogeneous* since two results can come from two different tables and thus their structure be different.

One of the cornerstones of our algorithm is the following lemma

**Lemma 1.** *A result $r$ is relevant both to $k_i$ and $k_j$ iff $r \in RS_i \wedge r \in RS_j$*

This gives us a way to design our algorithm. Precisely, we can find a list of results relevant to all $k_i \in \mathcal{K}$ by finding all $RS_i$ for each $k_i$ separately and then intersect all those result sets. Let $RS$ be the final set of results given by

$$RS = RS_1 \cap RS_2 \cap \cdots \cap RS_{|\mathcal{K}|} \tag{9}$$

Since it is a heterogeneous set, we assign an unique identifier to each result in order to perform intersection and union.

From now on we focus on the problem of finding all the results relevant to a single keyword $k$.

We describe the algorithm starting from an example. Suppose our keyword is *Textile*. As shown from the ontology of Figure 2, *Textile* is a *Discrete* type of *Industry* for *Kpi*. This means that we are looking for all KPIs related to textile industry, and linked information. The keyword gives us hints where to look, i.e., in which table (or tables) to start our search. In fact, we are primarily looking for KPIs with certain features, not for processes or anything else.

Once where to start searching has been found, we have to actually search. Given the keyword, we fetch all the relevant records. It is true that we are looking for *Textile*, but we need to look also for its synonyms. In fact, there could be relevant records that will not be added just because the description uses the word *Clothing* or *Fabric* instead of *Textile*. This shows how synonyms have to be taken into account when fetching records.

Suppose we fetched all records relevant to the keyword and its synonyms. We might be interested in fetching records relevant to generalizations or specifications of the concept linked to the keyword. For instance, we are also interested in records related to *Discrete* and its synonyms, since it is a more general concept of *Textile*. Loosely speaking, the records directly related to the keyword are more relevant than those related to generalizations or specifications.

It is also valuable to find all information connected to fetched records. For instance, if we had a certain KPI relevant to *Textile*, we would like to have all processes related. This could be achieved by moving from one table to another exploiting the database graph.

We will describe the general idea of algorithm and its main issues. Principal loop of the algorithm is shown in Figure 3. We will now describe in detail the main functions of our algorithm.

Figure 3. Ontology-based database search algorithm. $\mathcal{K}$ is a set of keywords.

```
procedure OBDBSEARCH(K)
    for all k_i ∈ K do
        table-list ← SEARCHSTARTINGTABLES(k_i)
        for all t ∈ table-list do
            RS_i ← RS_i ∪ SEARCH(k_i, t)
    return ⋂_i RS_i
```

### A. Search starting tables

As we said in Section VII, the first step is to understand from which table (or tables) we should start our search. This is a key concept because it shows us one of the benefits of using an ontology. Naively, we could use a *brute-force* approach by looking for the keyword in every table of the database. Besides efficiency issues, many unrelated records could be found. In fact, suppose we are looking for *Cost*. Since *Cost* is a feature of *KPI*, we should initially search the word *Cost* into the `kpi` table. If we also looked for the word *Cost* in other tables, such as `processes_lv3` or `standards`, we would find records containing *Cost*, but they would be semantically unrelated. Indeed, a *brute-force* approach would go against a semantic direction.

Ontologies should be exploited in order to understand if a keyword can be a feature or a specification of another concept - meaning there is a sort of container - and thus we expect a database table related with such concept to exists. This way we add semantics to the search.

A modified BFS on the ontology graph is executed in order to find a set of starting tables. We start from the concept directly related to the given keyword. Then we check if there is correspondence between a database table and such concept. If no table is found, we should expand from the main concept to reach its neighbours and, for each neighbour, check the existence of a table with a similar name. Intuitively, if we are looking for a sort of container, there will be no need to expand downwards, i.e., towards specifications of the main concept. We stop this BFS once a table matching with a concept has been found. From the set of candidate tables we should keep out all tables $t \in \mathcal{LT}$, i.e., tables which perform many-to-many relationships. In fact, they are likely to be just a set of primary keys pair, thus quite few textual information could be inferred.

There could be more than one matching table. Suppose we are looking for *Plan*. Since there is no corresponding table, we will reach *Process*. There are at least three matching tables: `processes_lv3`, `processes_lv2` and `processes_lv1` (since `kpi_processes` has been kept out). Each one has to be considered.

We should also deal with the special case in which a matching table is found at the first round. Suppose we are looking for *KPI*. We will find a match with `kpi` table without any other expansions. In that case, fetching all the records connected to the keyword *KPI* from `kpi` table would be quite wrong. Hence, we would fetch all records. Since actual fetching is

performed by a FETCH function in Figure 6, we should be able to discriminate whether to get all the records or not. So, we suppose to have two functions - SETFETCHALL(table) and ISFETCHALL(table) - to indicate whether to fetch all the records from a certain table. The last function will be used inside FETCH.

The algorithm in Figure 4 recaps above features.

Figure 4. Starting tables search algorithm. $k_i$ is the current keyword. $\mathcal{O}$ and $\mathcal{DB}$ are assumed to be global variables.

```
procedure SEARCHSTARTINGTABLES(k_i)
    tables ← ∅
    queue Q ← ∅
    tableFound ← false
    firstTime ← true

    ENQUEUE(Q, GETNODE(O, k_i))
    while Q ≠ ∅ ∧ ¬ tableFound do
        c ← DEQUEUE(Q)
        SETVISITED(c)
        KS ← c.name ∪ c.S
        for all t ∈ DB.T ∧ t ∉ LT do
            for all s ∈ KS do
                if s ∈ t.name then
                    tableFound ← true
                    if firstTime then
                        SETFETCHALL(t)
                    tables ← tables ∪{t}

        if ¬ tableFound then
            for all v ∈ GETPARENTS(c) ∪ GETHORIZLINKS(c) do
                if ¬ ISVISITED(v) then
                    ENQUEUE(Q, v)
        firstTime ← false
    return tables
```

### B. Search concepts

Once the starting tables have been determined, we must search the keyword and its synonyms inside those tables. This is accomplished by the algorithm in Figure 5. As we said before in Section VII, the ontology can also be exploited to search generalizations/specifications of the main concept. Those results should be marked as less relevant than those directly connected to the main concept.

Since we are talking about concepts that are more or less pertinent, the problem of results relevance starts to take shape. This is completely clarified in Section VII-D, but we will start to outline the matter in order to explain this step of the algorithm.

Given a keyword, we can get its related concept and thus actually get results by searching that keyword in the current table - one of the starting tables. See algorithm in Figure 6. With the purpose of getting information related to generalizations or specifications of the principal concept, we perform a modified BFS on the ontology, similar to the one in Figure 4.

Concepts exploration is only upwards, towards generalizations. Downwards exploration, i.e., towards specifications, is useful only when starting from the main concept. In fact, suppose $k$ = *Alimentary*. We can explore upwards and search *Continuous*. Since *Alimentary* is the main concept, we can also go downwards and search both *Flour* and *Food & Beverages*. This would be correct, because we would be looking for specifications of the main concept. On the contrary, if we explored downwards from *Continuous* towards *Pharma*, we would be wrong because *Pharma* is quite not linked with *Alimentary*. We assume we have two functions SETVISITCHILDREN(node) and VISITCHILDREN(node) that allow us to discriminate whether children of node must be visited or not.

In order to measure relevance, and thus differentiate concepts closer from those farther, we assign to each concept a starting ranking. Intuitively, concepts closer will have a starting ranking greater than those farther. How this works is shown in detail in Section VII-D.

Figure 5. Keyword search algorithm. $k_i$ is current the keyword and $t$ a database table in which search $k_i$. $\mathcal{O}$ is assumed to be a global variable.

```
procedure SEARCH(k_i, t)
    RS'_i ← ∅
    queue Q ← ∅
    firstTime ← true

    ENQUEUE (Q, GETNODE(O, k_i), baseRanking)
    while Q ≠ ∅ do
        (c, w) ← DEQUEUE(Q)
        SETVISITED(c)
        KS ← c.name ∪ c.S
        RS'_i ← RS'_i ∪ SEARCHTABLE(KS, t, w)

        if firstTime then
            SETVISITCHILDREN(c)
        if VISITCHILDREN(c) then
            for all u ∈ GETCHILDREN(c) do
                SETVISITCHILDREN(u)
            nodesToVisit ← GETCHILDREN(c) ∪ GETPARENTS(c)
        else
            nodesToVisit ← GETPARENTS(c)

        for all v ∈ nodesToVisit do
            if ¬ ISVISITED(v) then
                ENQUEUE(Q, v, w/10)
        firstTime ← false
    return RS'_i
```

### C. Search table and linked records

Suppose we now have found a keyword to search - regardless of whether it is a principal keyword or another concept - and we want to actually fetch records from a certain database table. This is accomplished by the FETCH function from the algorithm in Figure 6, which is essentially a SQL Select statement with a Like condition. In more detail, we select all the records from the given table, checking if at least one varchar or text type field contains the keyword or one of its synonyms.

We should determine and attribute relevance to all records returned from the FETCH function. The ADDRESULTS function takes care of this issue. Record relevance in connection with a certain keyword is explained in Section VII-D, but, roughly speaking, it depends on the number of occurences of a certain word and in which type of field it occurs.

 ISBN: 978-1-61208-408-4 

Figure 6. Search table algorithm. $KS$ is a set of string composed by the keyword and its synonyms, $t$ is the table in which search and $w$ is the starting ranking.

**procedure** SEARCHTABLE($KS, t, w$)
  $R \leftarrow$ FETCH($t, KS$)
  $RS'_i \leftarrow$ ADDRESULTS($t, KS, R, w$)
  **for all** $r \in R$ **do**
    $RS'_i \leftarrow RS'_i \cup$ LIMDFS($0, w, KS, t, l, r$)
  **return** $RS'_i$

So, we have a set of record $R$ relevant to the keyword, but related just to the starting table. As we said, it is also valuable to get all records linked with those in $R$. In fact, we might be interested in the processes related to a given set of KPIs. The database graph could be exploited in order to get linked results. As a matter of fact, we should be able to move from the current table to those directly connected and, for each $r \in R$, execute a join between those tables. This process should be iterated to reach more tables. Hence, we launch a DFS from each record $r \in R$, in order to get connected records from linked tables.

This DFS should not visit tables, but relationships instead. It could seem slightly like a nuance, but in truth it is not. In fact, we are interested in getting connected records, i.e., to perform joins between tables. So, starting from a given table $t_1$, we must perform all possible joins between $t_1$ and all linked tables. Since join conditions are individuated by foreign key constraints and thus by relationships, we have to visit them instead of tables. Suppose we perform a join between $t_1$ and $t_2$, then we must iterate starting from $t_2$.

If we ran a complete DFS, we would explore all links and get a sort of big join among all database tables. This is not advisable, because we cannot lose sight of our goal. Since we are looking for records somehow *semantically* connected to those in $R$, we should limit the depth of the expansions. A heuristic maximum depth of two/three tables seems reasonable. Figure 7 shows the limited DFS algorithm. FETCHLINKEDTABLE processes the foreign key constraint and returns all records linked to the one passed.

### D. Results relevance

Relevance is measured by observing how much and in which type of attribute the keyword appears in each record found through the FETCH operation. Evidently, records related to principal keyword with a relevant number of occurrences should be listed before records related to a generalization/specification of the keyword or, in the same way, related to the principal keyword but with few occurrences. Therefore, we associate a relevance level - starting ranking - to each record found. This relevance level depends on the level of the searched keyword. Later, this level will be increased or decreased using a bonus/penalty system.

In more detail, let $R$ be the record set returned after the FETCH operation for a certain table $t \in \mathcal{T}$. If $r \in R$, it contains at least in one field a certain keyword $k$, as shown by Equation 8. A starting ranking $sr$ is assigned to each $r \in R$. $Sr$ will be different whether $k$ is the principal keyword or a linked concept. For example, say $k =$ *Textile* has $sr = 10.000$, while $k =$ *Discrete* will have $sr = 1.000$. Each jump on the ontology graph entails a division by 10.

Every field $f_i \in \mathcal{F}$ gets a bonus/penalty $w_i$ weighted on $sr$ (e.g., $+20\%$ of $sr$), because we want the ranking to oscillate about the starting value. If $f_i$ is a description field $w_i$ depends on the number of occurrences of $k$, if instead $f_i$ is a string field, e.g., a name or a formula, $w_i$ is bonus if $k$ is contained, no penalties otherwise. More precisely, we recap bonus/penalties values that we heuristically decided to use in Table I.

Figure 7. Limited Depth First search algorithm. $d$ is current depth, $w$ is the starting ranking, $KS$ is a set of string composed by the keyword and its synonyms, $t$ is the incoming table, $l$ is the link we are visiting, $r$ is the current record to link.

**procedure** LIMDFS($d, w, KS, t, l, r$)
  $RS'_i \leftarrow \emptyset$
  **for all** $l \in$ GETLINKS($t$) **do**
    **if** $\neg$ ISVISITED($l$) **then**
      $RS'_i \leftarrow RS'_i \cup$ VISITDFS($0, w, KS, t, l, r$)
  **return** $RS'_i$
**procedure** VISITDFS($d, w, KS, t, l, r$)
  $RS'_i \leftarrow \emptyset$
  **if** $d \leq$ maxDepth **then**
    $t_2 \leftarrow$ GETLINKEDTABLE($l, t$)
    SETVISITED($l$)
    $R \leftarrow$ FETCHLINKEDTABLE($r, l$)
    $RS'_i \leftarrow$ ADDRESULTS($t_2, KS, R, w$)
    **for all** $r_l \in R$ **do**
      **for all** $l_2 \in$ GETLINKS($t_2$) **do**
        **if** $\neg$ ISVISITED($l_2$) **then**
          $d \leftarrow d + 1$
          **return** $RS'_i \cup$ VISITDFS($d, w, KS, t_2, l_2, r_l$)
  **return** $RS'_i$

TABLE I. BONUS/PENALTIES OVERVIEW.

| *Field type* | *Occurrences* | *Bonus/penalty* |
|---|---|---|
| String | at least one | $+25\%$ of $sr$ |
| Description | none | $-20\%$ of $sr$ |
| Description | one | $-15\%$ of $sr$ |
| Description | from 2 to 5 | 0 |
| Description | more than 5 | $+15\%$ of $sr$ |

So, the record relevance is given by the sum of the starting ranking and bonus or penalties associated with it. Negative $w_i$ are not added if a string field containing $k$ exists. This avoids penalizing results in which $k$ is contained in the name (so $r$ is very relevant) but description field has few occurrences. More formally, let $\mathcal{G} \subseteq \mathcal{F}$ be the set of string or description fields.

$$\text{Relevance}(r) := sr + \sum_{i:f_i \in \mathcal{G}} \delta(w_i) \tag{10}$$

where $\delta(w_i)$ is a function that returns 0 if $w_i < 0$ and a string field containing k exists, otherwise it returns $w_i$.

If a record is relevant to two or more keywords, the overall relevance is evaluated as the sum of the single relevances.

## VIII. PERFORMANCE EVALUATION

Our algorithm was evaluated considering a set of query, where different combination of keywords have been considered, as shown in Table II. Standard precision, recall and balanced f-measure were computed on the returned results. We recall that

$$\text{Precision} = \frac{\text{Correct results}}{\text{Retrieved results}} \quad \text{Recall} = \frac{\text{Correct results}}{\text{Relevant results}} \tag{11}$$

 ISBN: 978-1-61208-408-4

$$\text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (12)$$

In our case, results whose rank is greater than 1.000 are considered as *relevant*. We tested our algorithm on a relatively small size database. Although testing on a larger amount of data is still missing, results are promising, as Table II clearly shows.

TABLE II. PERFORMANCE EVALUATION.

| *Keywords* | *Precision* | *Recall* | *F-measure* |
|---|---|---|---|
| Plan | 53.28% | 100.00% | 69.52% |
| Cost | 80.00% | 100.00% | 88.89% |
| Make | 31.30% | 100.00% | 47.68% |
| Plan, Cost | 88.24% | 100.00% | 93.75% |
| Plan, Make | 60.00% | 100.00% | 75.00% |
| Cost, Make | 75.76% | 100.00% | 86.21% |
| Plan, Cost, Make | 93.94% | 100.00% | 96.88% |
| Deliver | 52.94% | 100.00% | 69.23% |
| Source | 77.05% | 100.00% | 87.04% |
| Time | 85.33% | 100.00% | 92.09% |
| Deliver, Source | 83.48% | 100.00% | 91.00% |
| Deliver, Time | 90.38% | 100.00% | 94.95% |
| Source, Time | 96.43% | 100.00% | 98.18% |
| Deliver, Source, Time | 96.08% | 100.00% | 98.00% |
| AVERAGE | 76.01% | 100.00% | 84.89% |

## IX. CONCLUSION AND FUTURE WORK

In this work, we have devised a novel ontology-based database search algorithm. So, we achieve the objective of retrieving information from a database using a semantic approach. Specifically, given a set of keywords, our goal was to retrieve a list of results relevant to all the keywords and have this list sorted by relevance.

A case study has been examined to apply and test such algorithm. We focused on the context of KPIs. In order to design a database to manage KPIs described by several standards, we have individuated a common structure for KPIs. We also designed an ontology to describe the KPIs domain. Obviously both database and ontology could be extended and designed in many other ways.

We modeled both database and ontology as a graph to solve flexibility and navigation issues. Semantic has been added understanding which tables are relevant and searching not only the principal keyword, but also those linked. Synonyms have been taken into account while searching. In order to measure the relevance, we introduced a heuristic ranking system, that evaluates how much and in which type of attribute the keyword appears in each record found.

Our algorithm requires just an existing database and an ontology describing the underlying domain. From a practical point of view, OBDBSearch is an external tool that could be easily and immediately used with real-world systems to perform semantic searches.

We observed promising experimental results, but testing on a larger amount of data would be fundamental to try out our algorithm performances.

## REFERENCES


[1] G. Chowdhury, Introduction to Modern Information Retrieval, 3rd ed., 2010.

[2] C. D. Manning, P. Raghavan, and H. Schütze, An Introduction to Information Retrieval, online ed. Cambridge University Press, Cambridge, England, 2009.

[3] P. Szeredi, G. Lukácsy, and T. Benkő, The Semantic Web Explained: The Technology and Mathematics behind Web 3.0. Cambridge University Press, Cambridge, England, Oct 2014.

[4] ISO-22400-2, "Manufacturing operations management — key performance indicators — part 2: Definitions and descriptions of kpis", Standard ISO 22400-2, December 2012.

[5] T. Tran and P. Mika, "A survey of semantic search approaches", 2012.

[6] T. Tran, H. Wang, and P. Haase, "Hermes: Data web search on a pay-as-you-go integration infrastructure", Web Semantics: Science, Services and Agents on the World Wide Web, vol. 7, no. 3, 2009, pp. 189 – 203.

[7] T. Tran, D. M. Herzig, and G. Ladwig, "Semsearchpro – using semantics throughout the search process", Web Semantics: Science, Services and Agents on the World Wide Web, vol. 9, no. 4, 2011, pp. 349 – 364.

[8] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu, "Spark: Adapting keyword query to semantic search", in The Semantic Web. Springer Berlin Heidelberg, 2007, vol. 4825, pp. 694–707.

[9] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu, "Avatar semantic search: A database approach to information retrieval", in Proc. of the 2006 ACM SIGMOD International Conference on Management of Data. ACM, 2006, pp. 790–792.

[10] E. Airio, K. Järvelin, P. Saatsi, J. Kekäläinen, and S. Suomela, "Ciri - an ontology based query interface for text retrieval", in Web Intelligence: Proc. of the 11th Finnish Artificial Intelligence Conference, 2004.

[11] T. Tran, P. Cimiano, S. Rudolph, and R. Studer, "Ontology-based interpretation of keywords for semantic search", in Proc. of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference. Springer-Verlag, 2007, pp. 523–536.

[12] D. Bonino, F. Corno, L. Farinetti, and A. Bosca, "Ontology driven semantic search", dec 2004.

[13] K.U. Sattler, I. Geist, and E. Schallehn, "Concept-based querying in mediator systems", The VLDB Journal, vol. 14, 2005, pp. 97–111.

[14] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Ontology-based database access", in Proc. of the Fifteenth Italian Symposium on Database Systems, 2007, pp. 324–331.

[15] A. Poggi, M. Rodriguez, and M. Ruzzi, "Ontology-based database access with dig-mastro and the obda plugin for protégé", in Proc. of the 4th Int. Workshop on OWL: Experiences and Directions, 2008.

[16] H. Dehainsala, G. Pierra, and L. Bellatreche, "Ontodb: An ontology-based database for data intensive applications", in Advances in Databases: Concepts, Systems and Applications. Springer Berlin Heidelberg, 2007, vol. 4443, pp. 497–508.

[17] S. Agrawal, S. Chaudhuri, and G. Das, "Dbxplorer: a system for keyword-based search over relational databases", in Proc. of the 18th International Conference on Data Engineering, 2002, pp. 5–16.

[18] N. Guarino, D. Oberle, and S. Staab, International Handbooks on Information Systems. Springer Berlin Heidelberg, 2009, ch. What Is an Ontology?, pp. 1–17.

[19] R. Studer, R. Benjamins, and D. Fensel, "Knowledge engineering: Principles and methods", Data and Knowledge Engineering, vol. 25 (1-2), 1998, pp. 161–198.

[20] R. T. Gruber, "A translation approach to portable ontologies", Knowledge Acquisition, vol. 5 (2), 1993, pp. 199–220.

[21] P. Patel-Schneider, B. Parsia, and B. Motik, "{OWL} 2 web ontology language structural specification and functional-style syntax (second edition)", W3C, {W3C} Recommendation, dec 2012.

[22] SCOR11, "Scor supply chain operations reference model", Standard SCOR rev. 11, December 2012.

[23] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 5th ed., 2005.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. The MIT Press Cambridge, Massachusetts, 2009.


 ISBN: 978-1-61208-408-4 