SemIndex: Semantic-Aware Inverted Index

Richard Chbeir[1], Yi Luo[2], Joe Tekli *[3], Kokou Yetongnon[2], Carlos Raymundo Ibañez[4], Agma J. M. Traina[5], Caetano Traina Jr.[5], and Marc Al Assad[3]

[1] University of Pau and Adour Countries, Anglet, France
[2] University of Bourgogne, Dijon, France
[3] Lebanese American University, Byblos, Lebanon
[4] Universidad Peruana de Ciencias Aplicadas, Lima, Peru
[5] University of São Paulo, São Carlos-SP, Brazil

Abstract. This paper focuses on the important problem of semantic-aware search in textual (structured, semi-structured, NoSQL) databases. This problem has emerged as a required extension of the standard containment keyword based query to meet user needs in textual databases and IR applications. We provide here a new approach, called *SemIndex*, that extends the standard inverted index by constructing a tight coupling inverted index graph that combines two main resources: a general purpose semantic network, and a standard inverted index on a collection of textual data. We also provide an extended query model and related processing algorithms with the help of *SemIndex*. To investigate its effectiveness, we set up experiments to test the performance of *SemIndex*. Preliminary results have demonstrated the effectiveness, scalability and optimality of our approach.

Keywords: Semantic Queries, Inverted lndex, NoSQL indexing, Semantic Network, Ontologies.

1 Introduction

Processing keyword-based queries is a fundamental problem in the domain of Information Retrieval (IR). Several studies have been done in the literature to provide effective IR techniques [10, 9, 6]. A standard containment keyword-based query, which retrieves textual identities that contain a set of keywords, is generally supported by a full-text index. Inverted index is considered as one of the most useful full-text indexing techniques for very large textual collections [10], supported by many relational DBMSs, and recently extended toward semi-structured [9] and unstructured data [6] to support keyword-based queries.

Besides the standard containment keyword-based query, semantic-aware or knowledge-aware (keyword) query has emerged as a natural extension encouraged by real user demand. In semantic-aware queries, some knowledge[6] needs to be taken into consideration while processing. Let's assume having a movie database, as shown in Table 1. Each movie, identified with an *id*, is described with some (semi-structured) text, including movie *title*, *year* and `plot`. For queries "sound of music", "Maria nun" and "sound Maria", the query result is movie O_3. However, if the user wants to search for a movie but cannot recall the exact movie title, it is natural to assume that (s)he may modify the query terms to some semantically similar terms, for example, "voice of music". Also, it is common that the terms provided by users are not exactly the same, but are semantically relevant to terms that the plot providers use. Clearly, the standard inverted index which only supports exact matching cannot deal with these cases. Various approaches combining different types of data and se-

* Corresponding author: joe.tekli@lau.edu.lb/jtekli@gmail.com
[6] Also called domain, collaborative, collective knowledge or semantic network

Table 1. A Sample Movie Data Collection

ID	Textual Contents
O_1	*When a Stranger Calls (2006)*: A young high school student babysits for a very rich family. She begins to receive strange phone calls threatening the children...
O_2	*Code R (1977)*: This CBS adventure series managed to combine elements of "Adam-12", "Emergency" and "Baywatch" at the same time...
O_3	*Sound of Music, The (1965)*: Maria had longed to be a nun since she was a young girl, yet when she became old enough discovered that it wasn't at all what she thought...
...	...

mantic knowledge have been propose to enhance query processing (cf. Related Works). In this paper, we present a new approach integrating knowledge into a semantic-aware inverted index called *SemIndex* to support semantic-aware querying. Major differences between our work and existing methods include:

- **Pre-processing the index**: Existing works use semantic knowledge to pre-process queries, such as query rewriting/relaxation and query suggestion [2, 5], or to post-process the query results, such as semantic result clustering [16, 17, 25]. Our work can be seen as another alternative to consider the semantic gap by enclosing semantic knowledge directly into an inverted index, so that main tasks can be done before query processing,
- **User involvement**: Most existing works introduce some predefined parameters (heuristics) to rewrite queries such that users are only involved in the query refinement (expansion, filtering, etc.) process after providing the first round of results [3, 4, 14, 20]. In our work, we aim at allowing end-users to write, using the same framework, classical queries but also semantically enriched queries according to their needs. They are involved in the whole process (during initial query writing and then query rewriting).
- **Providing more results**: Most existing works focus on understanding the meaning of dat/queries through semantic disambiguation [2, 13, 16], which: i) is usually a complex process requiring substantial processing time [15], and ii) depends on the query/data context which is not always sufficiently available, and thus does not guarantee correct results [7, 23]. The goal of our work is, with the help of semantic knowledge, to find *more* semantically relevant results than what a traditional inverted index could provide, while doing it more efficiently than existing disambiguation techniques.

In order to build *SemIndex*, we create connections between two data resources, *a textual data collection* and *a semantic knowledge base*, and map them into a single data structure. An extended query model with different levels of semantic awareness is defined, so that both semantic-aware queries and standard containment queries are processed within the same framework. Figure 1 depicts the overall framework of our approach and its main components. The *Indexer* manages *SemIndex*, while the *Query Processor* accepts semantic-aware queries from the user and processes the queries with *SemIndex*.

The rest of this paper is organized as follows. Designing and building *SemIndex* will be presented in Section 2. Section 3 will introduce our query model for semantic-aware queries. We will also present the algorithms for processing semantic-aware queries using *SemIndex*. We provide preliminary experimental results of executing different queries on a set of textual data collections in Section 4. Related works are discussed in Section 5 before concluding the paper and providing some future works.

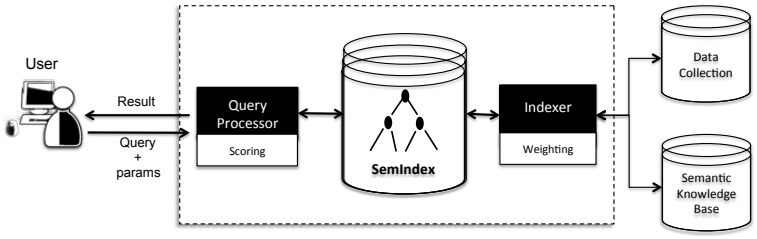

Fig. 1. SemIndex Framework

2 Index Design

In the following, we analyze the (textual and semantic) input resources required to build our index structure, titled *SemIndex*. We also show how to create connections between the input resources and how to design the logical structure of *SemIndex*. The physical structure will be detailed in a dedicated paper.

2.1 Representation and Definitions

Textual Data Collection: In our study, a textual data collection could be a set of documents, XML fragments, or tuples in a relational or NoSQL database.

Definition 1 *A* Textual Data Collection Δ *is represented as a table defined over a set of attributes* $\mathcal{A} = \{A_1, \ldots, A_p\}$ *where each* A_i *is associated with a set of values (such as strings, numbers, BLOB, etc.) called the* domain of A_i *and denoted by* $dom(A_i)$*. A semantic knowledge base (KB) can be associated to one or several attribute domains* $dom(A_i)$*. Thus, given a table* Δ *defined over* $\mathcal{A}$*, objects* t *in* Δ *are denoted as* $\langle a_1, \ldots, a_p \rangle$*, where* $a_i \in dom(A_i)$*. Each* a_i *from* t *is denoted as* $t.a_i$*.*

Semantic Knowledge-Base: We adopt graph structures for modeling semantic knowledge. Thus, entities are represented as vertices, and semantic relationship between entities are modeled as directed edges [7]. In this work, we will illustrate the design process of *SemIndex* using WordNet version 3.0 [8] as the semantic knowledge resource. Part of the WordNet ontology is shown in Figure 2. Each synset represents a distinct concept, and is linked to other synsets with semantic relations (including *hypernymy*, *hyponymy*, *holonymy*, etc.). Note that multiple edges may exist between each ordered pair of vertices, and thus the knowledge graph is a multi-graph.

Definition 2 *A* Semantic Knowledge Base KB*, such as WordNet, is a graph* $G_{kb}(V_{kb}, S_{kb}, E_{kb}, L_{kb})$ *such that:*

- V_{kb} *is a set of vertices/nodes, denoting entities in the given knowledge base. For WordNet,* V_{kb} *includes synsets and words*
- S_{kb} *is a function defined on* V_{kb}*, representing the string value of each entity*
- E_{kb} *is a set of directed edges; each has a label in* L_{kb} *and is between a pair of vertices in* V_{kb}
- L_{kb} *is a set of edge labels. For WordNet,* L_{kb} *includes hyponymy, meronymy, hypernymy, holonymy, has-sense and has-sense-inverse, etc.*

[7] We use the terms "edge" and "directed edge" interchangeably in this paper

In Figure 2, W_4, W_6, W_7, W_8 and W_9 represent words, and their string values (lemma of the words) are shown aside of the nodes. S_1, S_3 and S_4 are synsets, and their string values are their definitions. If one sense of a word belongs to a synset, it is represented with two edges between the synset node and word node with opposite directions, labeled *has-sense* and *has-word*, that we represent here with only one left-right arrow.

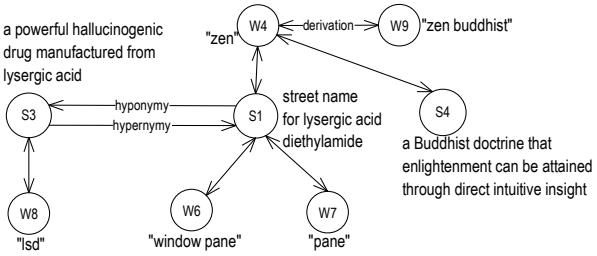

Fig. 2. Part of the Semantic Knowledge Graph of WordNet

***SemIndex* graph:** To combine our resources, we define a *SemIndex* graph.

Definition 3 *A SemIndex* graph $\tilde{G}$ *is a directed graph* $(V_i, V_d, L, E, S_v, S_e, W)$*:*

- V_i *is a set of index nodes (denoting entities in a knowledge-base, index or searchable terms in a textual collection) represented visually as circles* ○
- V_d *is the set of data nodes (belonging to the textual collection) represented visually as squares* □
- L *is a set of labels*
- E *is a set of ordered pairs of vertices in* $V_i \bigcup V_d$ *called edges. Edges between index nodes are called index edges (represented visually as* $\rightarrow$*), while edges between index nodes and data nodes are called data edges (represented visually as* $\dashrightarrow$*)*
- S_v *is a function defined on* $V_i \bigcup V_d$*, representing the value of each node*
- S_e *is a function defined on* E*, assigning a label* $\in L$ *to an edge*
- W *is a weighting function defined on nodes in* $V_i \bigcup V_d$ *and edges in* E*.*

2.2 Logical Design

In this part, we introduce the logical design techniques in building *SemIndex*.

Building *SemIndex*: *SemIndex* adapts *tight coupling* techniques to index a textual data collection and a semantic knowledge base in the same framework, and directly create a posting list for all searchable contents. In the following, we describe how to construct *SemIndex*.

1- Indexing Input Resources: Given a textual data collection Δ, a multi-attributed inverted index (associated to one or several attributes A) is a mapping $ii : V_A^s \rightarrow V^t$, where V_A^s is a set of values for attributes A, which we also call a *searchable terms*, and V^t is the set of textual data objects. A multi-attributed inverted index of a set of textual data objects Δ is represented as a *SemIndex* graph $\tilde{G}_A$ such that:

- V_i is a set of *index nodes*, representing all searchable terms which appear in the attribute set A of the collection
- V_d is the set of textual data objects in Δ
- L includes *contained* label indicating the containment relationship from a searchable term in V_i to a data object in V_d
- E is a set of ordered pairs of vertices in $V_i \bigcup V_d$
- S_v assigns a term to an index node, and its text contents to a data node
- S_e assigns the *contained* label to each edge
- W assigns a weight (according to the importance/frequency of the term within the text content) to an edge E.

An example of a *SemIndex* graph inverted index $\tilde{G}_A$ based on the textual collection provided in Table 1 is shown in Figure 3 (upper part).

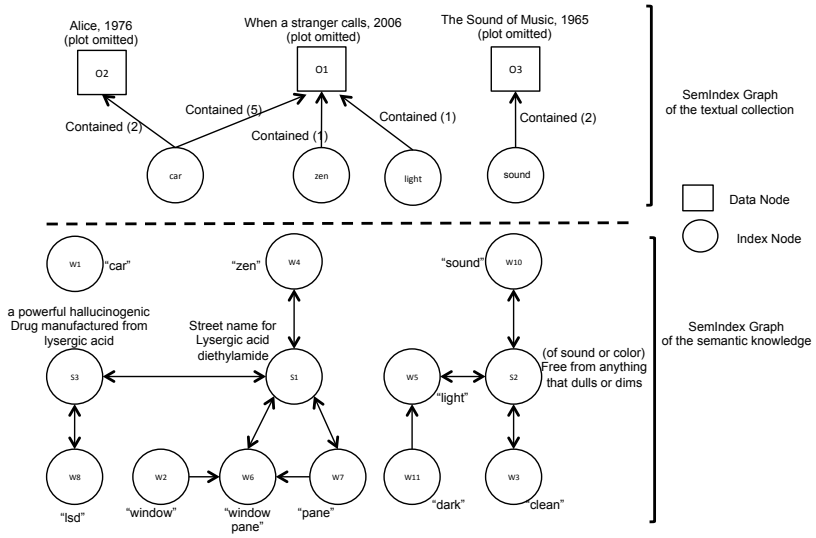

Fig. 3. Part of the *SemIndex* Graph of the textual collection

Similarly, indexing the semantic knowledge base G_{kb} is also represented as a *SemIndex* graph $\tilde{G}_{kb}$ that inherits the properties of G_{kb} where:

- V_d is an empty set
- V_i is a set of vertices/nodes, denoting entities in the given knowledge base and all other searchable terms. For WordNet, V_i includes synsets and words as well as other terms that appear in the string value in G_{kb}. Thus, V_i is a superset of vertices in the knowledge graph G_{kb}
- L is a set of edge labels, including those inherited from G_{kb} (e.g., *hyponymy*, *meronymy*, etc.), and a special label *meronymy** indicating the *containment* relation from a searchable term to an entity in V_i
- S_e assigns, in addition to previous edges of G_{kb}, the *meronymy** label from a searchable term to an entity in V_i
- W is the weighting function assigning a weight (default weight is 1) to all nodes and edges.

We assume that connections between searchable terms and entities in V_i can be found by the same Natural Language Processing techniques used when indexing the textual collection. For example, in Figure 3 (lower part), we find that word "window" (W_2) is contained in the word "window pane" (W_6). Thus,

an extra edge labeled *meronymy** from W_2 to W_6 is inserted into the graph as shown in Figure 3. Note that the NLP algorithms run only on synsets and multi-term words, in order to prevent duplicated nodes to be produced.

2- Coupling Resources: When coupling both indexes, we get one *SemIndex* graph called $\tilde{G}_{SemIndex}$. In $\tilde{G}_{SemIndex}$, only *searchable terms* have (mainly string) values (which will be defined in Step 3 in the construction procedure). Weights are assigned to all edges and data nodes in the graph. To summarize:

- V_i is a set of index nodes of $\tilde{G}_A \cup \tilde{G}_{kb}$
- V_d is the set of data nodes of $\tilde{G}_A$
- L is a set of labels in the $\tilde{G}_{kb}$ and a special label *contained*, indicating the containment relationship from a searchable term in V_i to an entity in V_d
- E is a set of ordered pairs of nodes in V_i (*index edges*) and V_d (*data edges*)
- S_v is a function of string values defined on *searchable terms*: a subset of V_i
- S_e assigns $\tilde{G}_{kb}$ relationship labels to *index* edges and *contained* label to *index/data* edges
- W is the weighting function defined on all nodes in $V_i \bigcup V_d$ and edges in E.

The pseudo-code of constructing $\tilde{G}_{SemIndex}$ is composed of 7 steps as shown in Algorithm 1. Each step is detailed as follows.

Algorithm 1 $\tilde{G}_{SemIndex}$ Construction

Input:
KB: a semantic knowledge base;
Δ: a textual data collection;
ω: a weighting schema;
c_1: a constant (used in Step 4) to delimit the co-occurrence window
c_2: a constant (used in Step 4) to select top terms
Output:
$\tilde{G}_{SemIndex}$: a *SemIndex* graph instance

1: Build inverted index for Δ to construct $\tilde{G}_A$
2: Build inverted index for G_{kb} to construct $\tilde{G}_{kb}$
3: Merge $\tilde{G}_A$ and $\tilde{G}_{kb}$ into $\tilde{G}_{SemIndex}$ and find searchable terms
4: For each missing term, find the most relevant terms in $\tilde{G}_{kb}$
5: Assign weights to all edges in $\tilde{G}_{SemIndex}$ and all data nodes according to ω
6: Aggregate edges between each ordered pair of nodes
7: Remove from $\tilde{G}_{SemIndex}$ all edge labels, and string values of all nodes except searchable terms

- **Step 1:** It builds the multi-attributed inverted index on the contents of the textual data collection as a graph $\tilde{G}_A$ as defined and illustrated previously.
- **Step 2:** Given a semantic knowledge graph G_{kb} representing the semantic knowledge base KB given as input, it builds an inverted index for string values of each knowledge base entity, and construct the graph $\tilde{G}_{kb}$.
- **Step 3:** it combines the two *SemIndex* graphs. Data nodes in the result graph $\tilde{G}_{SemIndex}$ are the set V_d in $\tilde{G}_A$ (denoted as V_d^A), while all other nodes are index nodes. This step denotes the searchable terms of $\tilde{G}_{kb}$ as V_i^{kb} (which are vertices with one or more outgoing edges labeled *contained*) and then merges the two sets of searchable terms V_i^{kb} and V_i^A (representing the index nodes of $\tilde{G}_A$) as follows: if string values of two vertices are equal, remove one of them and merge all the connected edges. We use V_i^+ to denote the conjunctive set $V_i^{kb} \bigcup V_i^A$, which is the set of all *searchable terms* in

SemIndex. Figure 4 shows the result of combining the two *SemIndex* graphs of the sample textual collection and the WordNet extract provided here.

- **Step 4:** For the missing term problem, we create links from each missing term to one or more closely related terms, with a new edge label *refers-to*, using a distributional thesaurus[8] based on the textual collection to mine relativeness between missing terms and used index. We cover the *missing term problem* in more detail in a dedicated paper.
- **Step 5:** It assigns weights to edges and textual objects, according to ω. The weight will be used to rank query results. Different weighting schemes can be adopted in our approach. We propose below the principles of a simple weighting schema for computing edge and node weights:
  - *Containment edges*: For a (data) edge from a term to a textual object, its weight is an IR score, such as term frequency. If the textual collection is formatted, this IR score could also be assigned to reflect the importance of a term, e.g., in large font size, in capitalized form, etc. When the textual collection is structured, higher weights are given to terms which appear in important places, e.g., title, author's name, etc.
  - *Structural edges*: The weight of a structural (index) edge is determined by edge label and by the number of edges with the same label from its starting node [21]. Please note that if the knowledge base is hierarchical (which is not the case for WordNet), the level of the edge in the hierarchy can also be taken into consideration [19].
  - *Nodes*: Assign "object rank" to all object nodes, based on metadata of objects, including text length, importance or reliability of data source, its publishing date, query logs, and so on. A PageRank-style weighting schema could also be adapted for Web documents.
- **Step 6:** If an ordered pair of vertices is connected with two or more edges, it merges the edges and aggregates the weights. This means that $\tilde{G}_{SemIndex}$ becomes a graph rather than a multi-graph, which simplifies processing.
- **Step 7:** It removes edge labels and string values of all nodes except V_i^+ (searchable terms), since they are not required for processing semantic queries, which helps improve *SemIdex*'s scalability.

Figure 4 illustrates an instance of $\tilde{G}_{SemIndex}$ (without edge and node weights) which is based on the knowledge graph depicted in Figure 2.

3 Executing queries with *SemIndex*

In this section, we define our query model and present a processing algorithm to perform semantic-aware search with the help of *SemIndex*.

3.1 Queries

The *semantic-aware queries* considered in our approach are *conjunctive projection-selection* queries over Δ of the form $\pi_X \sigma_P \ell(\Delta)$ where X is a non empty subset of $\mathcal{A}$, $\ell \in \mathbb{N}$ is a query-type threshold, and P is a selection projection predicate defined as follows.

Definition 4 *A selection projection predicate P is an expression, defined on a string-based attribute A in $\mathcal{A}$, of the following forms: $(A\,\theta\,a)$, where a is a user-given value (e.g., keyword), and $\theta \in \{=, \texttt{like}\}$ whose evaluation against values in $dom(A)$ is defined. A conjunctive selection projection query is made of a conjunction of selection projection predicates.*

[8] A distributional thesaurus is a thesaurus generated automatically from a corpus by finding words that occur in similar contexts to each other [11, 24].

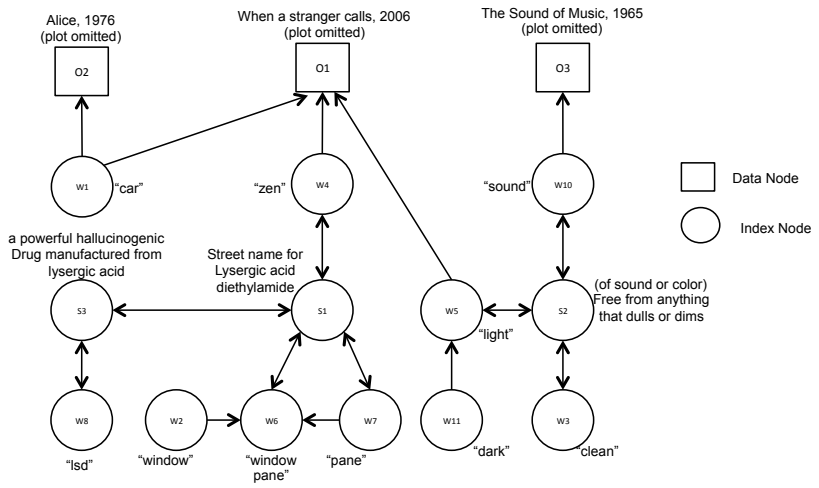

Fig. 4. *SemIndex* graph integrating the textual collection and semantic knowledge

---

Following the value of ℓ, we consider four semantic-aware query types:

- **Standard Query**: When $\ell = 1$, the query is a standard containment query and no semantic information is involved.
- **Lexical Query**: When $\ell = 2$, besides from the previous case, lexical connections, i.e., links between terms, may be involved in the result.
- **Synonym-based Query**: When $\ell = 3$, synsets are also involved. Note that there is no direct edge between textual object and synset node.
- **Extended Semantic Query**: When $\ell \geqslant 4$, the data graph of *SemIndex* can be explored in all possible ways. When ℓ grows larger, the data graph is explored further to reach even more results.

3.2 Query Answer

The answer to q in Δ, denoted as $q(\Delta)$, is defined as follows.

Definition 5 *Given a SemIndex graph $\tilde{G}$, the query answer $q(\Delta)$ is the set of distinct root nodes of all answer trees. We define an answer tree as a connected graph T satisfying the following conditions:*

- *(tree structure) T is a subgraph of $\tilde{G}$. For each node in T, there exists exactly one directed path from the node to the root object.*
- *(root object) The tree root is a data node, and it is the only data node in T, which corresponds to the textual object returned to the user.*
- *(conjunctive selection) For each query term in S, its corresponding index node is in the answer tree.*
- *(height boundary) Height of the tree, i.e., the maximal number of edges between root and each leaf, is no greater than the threshold ℓ.*
- *(minimal tree) No node can be removed from T without violating some of the above conditions.*

It can be proven that all leaves in the answer tree are query terms, and the number of leaves in T is smaller or equal to k, where k is the number of query terms. Also, the maximal in-degree of all nodes in T is at most k.

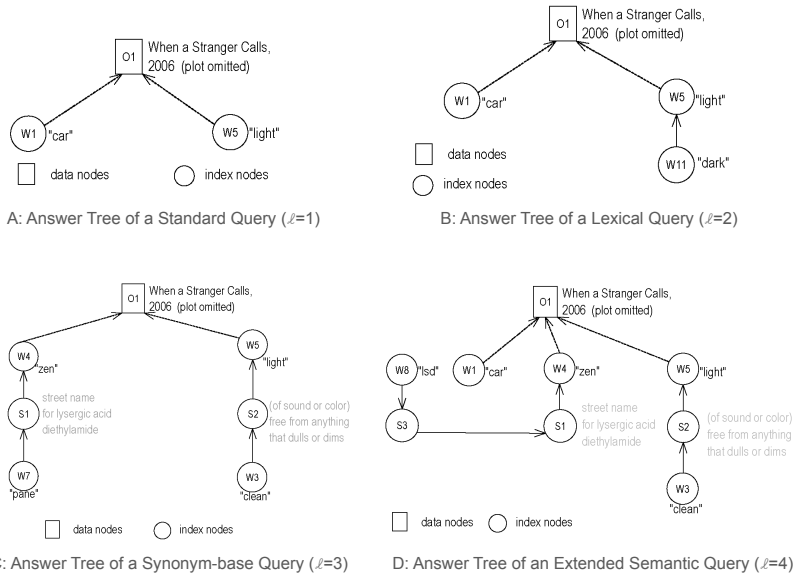

Fig. 5. Sample query answer trees

According to the value of ℓ which serves as an interval radius in the*SemIndex* graph, various answer trees can be generated for a number of query types:

- **Standard Query**: When $\ell = 1$, the root of the answer tree is linked directly to all leaves, representing the fact that the result data object contains all query selection terms directly. An example answer tree is shown in Figure 5-A for the query q: $\pi_{\mathcal{A}}\sigma_{A\in(\text{“}car\text{”},\text{“}light\text{”})}\ell_{=1}(\Delta)$.
- **Lexical Query**: When $\ell = 2$, the answer tree contains also lexical connections between selection terms. Figure 5-B is an example answer tree of q: $\pi_{\mathcal{A}}\sigma_{A\in(\text{“}car\text{”},\text{“}dark\text{”})}\ell_{=2}(\Delta)$.
- **Synonym-based Query**: When $\ell = 3$, the answer tree contains, in addition to the two previous cases, the synsets. Note that due to the “minimal tree” restriction, a synset cannot be a leaf node of an answer tree. Thus, if an answer tree contains a synset, the height of the tree is no less than 3. A sample answer tree is shown in Figure 5-C for q: $\pi_{\mathcal{A}}\sigma_{A\in(\text{“}pane\text{”},\text{“}clean\text{”})}\ell_{=3}(\Delta)$. Synonyms of the two query terms, “zen” and “light”, are also contained in the result object O_1.
- **Extended Semantic Query**: When $\ell \geqslant 4$, the answer tree contained additional nodes according to the provided value. An example answer tree is shown in Figure 5-D for q: $\pi_{\mathcal{A}}\sigma_{A\in(\text{“}lsd\text{”},\text{“}clean\text{”})}\ell_{=4}(\Delta)$.

3.3 Query Processing

Algorithm 2 is the procedure to process semantic-aware queries, given a set of query terms, *terms*, and a query-type threshold ℓ. Function expandNode(n,ℓ) performs the expansion of a node n. Basically, it explores the *SemIndex* graph with Dijkstra's algorithm from multiple starting points (multiple query terms). For each visited node n, we store its shortest distances from all starting points (query terms). The path score of a node n to a query term t is the sum of all weights on index edges along the path between t and n, thus the shortest distances of n are also the minimal path scores of n to all query terms.

Algorithm 2 SemSearch(terms[], ℓ)

Input:
terms: a set of selection terms
ℓ : a query-type threshold
Output:
out: a set of data nodes

```
1: for each i ∈ terms do
2:    rs = Selecting nodeid from SemIndex with value = i); //selecting index nodes
      from the knowledge base as well as missing terms from the textual collection
3:    for each nodeid ∈ rs do
4:       n = nodes.cget(nodeid); //retrieve or create a node with given id
5:       n.initPathScores(); //initialize the path scores of n
6:       todo.insert(n); //insert n into todo list
7:    end for
8: end for
9: while todo.isNotEmpty() do
10:   n = todo.pop(); //retrieve node with minimal structural score
11:   expandNode(n,ℓ);
12: end while
13: return out;
```

For example in Figure 5-C, the query terms are "pane" and "clean", and the algorithm starts to expand from two nodes W_7 and W_3. Path scores of W_7 are initialized to be a vector $< 0, \infty >$, since the shortest distance from W_7 to "pane" is 0, but the node is not reachable from "clean". Similarly, the path scores of W_3 are initially $< \infty, 0 >$. The minimal path scores must be updated when an edge is explored in the graph. For example, before finding the tree in Figure 5-B, the path scores of node O_1 is $< 2, \infty >$ (assume all edge weights are equal to 1), and the path scores of W_5 are $< \infty, 2 >$. After exploring the edge from W_5 to O_1, the path scores value of O_1 becomes $< 1, 2 >$, and O_1 is reachable from all query terms. The algorithm also keeps a *todo* list, which contains all nodes to be further expanded. The *todo* list is ordered on structural scores of the nodes. We define the structural score of a node n to be the maximal path score in the tree rooted on n, as shown in the following formula.

$$sscore_n = \max_{t \in rterms} pathscore_n(t) \tag{1}$$

where $rterms$ is the set of reachable query terms of n. It can be proven that if n is reachable from every term and all path scores of n are minimal, the structural score of n must be minimal among all trees rooted on n. For each result textual object, the algorithm always returns the answer tree with minimal structural score, thus it is not necessary to prune duplicated query results.

4 Experiments

We conducted a set of preliminary experiments to observe the behavior of *SemIndex* in weighting, scoring, and retrieving results. In this paper, we only present results related to processing time. We are currently working on experimentally comparing our approach with existing methods.

4.1 Experimental Setup

We ran our experiments on a PC with Intel 2GHz Dual CPU and 2GB memory. *SemIndex* was physically implemented in a MySQL 5.1 database with the query

processor written in Java. We downloaded 90,091 movie plots from the IMDB database[9], and used WordNet 3.0 as our semantic knowledge base. We build *SemIndex* on plot contents and movie titles, which means that each textual object is a movie title combined with its plot (cf. Table 2).

Table 2. *SemIndex* Database Size

Database Name	Table Name	Table Size	Table Cardinality (#Row)
IMDB	Data IMDB	56 M	90K
WordNet	Data Adjective	3,2M	18K
	Data Verb	2,8M	13K
	Data Noun	15,3M	82K
	Data Adverb	0,5K	3K
	Index Verb	0,5M	11k
	Index Adverb	0,2M	3,6K
	Index Noun	4,8M	117K
	Index Adjective	0,8M	21k
	Index Sense	7,3M	207K
SemIndex	Lexicon	5.8M	146K
	Neighbors	116M	230K
	PostingList	340M	740K

4.2 Query Processing

In order to test the performance of *SemIndex*, we manually picked two groups of queries, shown in Table 3. In the first query group $Q1$, from $Q1-1$ to $Q1-8$, the height of the answer trees is bounded, the number of returned results is limited to 10, and each query contains from 2 to 5 selection terms. In the second group $Q2$ group, from $Q2-1$ to $Q2-4$, queries share the same query terms with different levels of tree height boundary and with an unlimited number of query results. All queries were processed 5 times, retaining average processing time. Detailed statistics are shown in Table 4.

Table 3. Sample Queries

Query Id	Height Boundary	Max. N# of Results	Selection Terms
Q1-1	4	10	car window
Q1-2	4	10	reason father
Q1-3	4	10	car window clean
Q1-4	4	10	apple pure creative
Q1-5	4	10	car window clean music
Q1-6	4	10	death piano radio war
Q1-7	4	10	car window clean music Tom
Q1-8	4	10	sound singer stop wait water
Q2-1	1	unlimited	car window clean
Q2-2	2	unlimited	car window clean
Q2-3	3	unlimited	car window clean
Q2-4	4	unlimited	car window clean

Figure 6 shows query processing time to retrieve the 10000 results of $Q2-4$. Initial response time is the processing time to output the first result. We break the latter into CPU and I/O time in order to better evaluate the processing costs of the algorithm. Minimal height is the height of the first returned answer tree while the maximal one is the height of the last answer tree allowing to reach the number of expected results. k, N_E and N_O are respectively the number of query terms, visited entity nodes, and object nodes during query processing.

From Table 4, we see that most queries are processed within 2 seconds, which is positively encouraging, except for Q2-4, since it is of an extended

[9] http://imdb.com

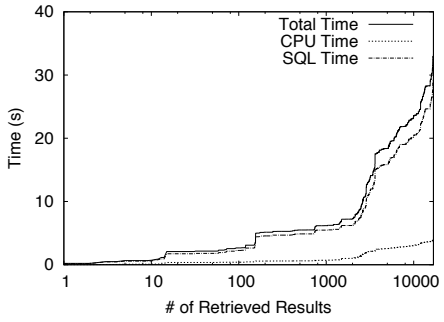

Fig. 6. Processing Time (Q2-4)

semantic type, which retrieves 16830 results in about a half minute. In fact, while the total number of textual objects in our plot dataset is around 90K, Q2-4 visits about 80K of object nodes which explains the significant increase in time. Also, we realize that the initial response times of Q1-5 and Q1-6 are quite different. Both of them contain 4 terms and all statistics of the two queries are similar except the minimal tree height. We also observe that for the second query group, the CPU time is dominated by the SQL time, while for the first query group, CPU cost is dominant. We analyzed the difference and found that, for the second query group, the algorithm cannot stop until all visited entity nodes are queried and the whole search space is examined. However for the first query group, since the number of query results is limited, the algorithm stops whenever it has found enough answer trees. Thus, for the first group of queries, although large N_E and N_O suggest large CPU time to create those nodes in memory, yet most of the nodes are not revisited for expansion before the algorithm stops, which significantly reduces overall query processing time.

Table 4. Processing Statistics

Queryid	Initial Response Time (ms)	Total Process Time (ms)	Min. Height	Max. Height	# of Results Returned	SQL Time (ms)	CPU Time (ms)	k	N_E	N_O
Q1-1	26	27	1	1	10	15	12	2	399	3346
Q1-2	39	40	1	1	10	21	19	2	252	9386
Q1-3	25	148	1	3	10	87	60	3	2528	7646
Q1-4	90	315	4	4	10	236	79	3	2812	11543
Q1-5	694	700	4	4	10	224	475	4	7849	32776
Q1-6	175	511	3	4	10	193	318	4	5071	38988
Q1-7	1223	1371	4	4	10	643	728	5	9505	50747
Q1-8	1469	1555	4	4	10	467	1076	5	12449	45715
Q2-1	18	18	1	1	2	11	7	3	3	3803
Q2-2	25	208	1	2	6	191	16	3	533	4394
Q2-3	21	842	1	3	515	678	164	3	1564	34564
Q2-4	200	31991	1	4	16830	27793	4198	3	19007	79997

5 Related Work

Including semantic processing in inverted indexes to enhance data search capabilities has been investigated in different approaches: i) including semantic knowledge into an inverted index, ii) including full-text information into the semantic knowledge base, and iii) building an integrated hybrid structure.

The first approach consists in adding additional entries in the index structure to designate semantic information. Here, the authors in [12] suggest extending the traditional (*term, docIDs*) inverted index toward a (*term, context, docIDs*) structure where contexts designate synsets extracted from WordNet,

associated to each term in the index taking into account the statistical occurrences of concepts in Web document [1]. An approach in [26] extends the inverted index structure by adding additional pointers linking each entry of the index to semantically related terms, (*term, docIDs, relatedTerms*). Yet, the authors in [12, 26] do not provide the details on how concepts are selected from WordNet and how they are associated to each term in the index.

Another approach to semantic indexing is to add words as entities in the ontology [1, 18, 22]. For instance, adding triples of the form *word occurs-in-context concept*, such that each word can be related to a certain ontological concept, when used in a certain context. Following such an approach: i) the number of triples would naturally explode, given that ii) query processing would require reaching over the entire left and the right hand sides of this *occurs-in-context* index, which would be more time consuming [1] than reading on indexed entry such as with the inverted index.

A third approach to semantic indexing consists in building an integrated hybrid structure: combining the powerful functionalities of inverted indexing with semantic processing capabilities. To our knowledge, one existing method in [1] has investigated this approach, introducing a joint index over ontologies and text. The authors consider two input lists: containing text postings (for words or occurrences), and lists containing data from ontological relations (for concept relations). The authors tailor their method toward incremental query construction with context-sensitive suggestions. They introduce the notion of *context lists* instead of usual inverted lists, where a prefix contains one index item per occurrence of a word starting with that prefix, adding an entry item for each occurrence of an ontological concept in the same context as one of these words, producing an integrated 4-tuples index structure (*prefix, terms*) $\leftrightarrow$ (*term, context, concepts*). The method in [1] is arguably the most related to our study, with one major difference: the authors in [1] target semantic full-text search and indexing with special emphasis on IR-style incremental query construction, whereas we target semantic search in textual databases: building a hybrid semantic inverted index to process DB-style queries in a textual DB.

6 Conclusions and Future Work

In this paper, we introduce a new semantic indexing approach called *SemIndex*, creating a hybrid structure using a tight coupling between two resources: a general purpose semantic network, and a standard inverted index defined on a collection of textual data, represented as (multi)graphs. We also provide an extended query model and related processing algorithms, using *SemIndex* to allow semantic-aware querying. Preliminary experimental results are promising, demonstrating the scalability of the approach in querying a large textual data collection (IMBD) coined with a full-fledge semantic knowledge base (WordNet). We are currently completing an extensive experimental study to evaluate *SemIndex*'s properties in terms of: i) genericity: to support different types of textual (structured, semi-structured, NoSQL) databases, ranking schema, and knowledge-bases, ii) effectiveness: evaluating the interestingness of semantic-aware answers from the user's perspective, and iii) efficiency: to reduce index's building and updating costs as well as query processing cost. The systems physical structure (in addition to the logical designs provided in this paper) will also be detailed in an upcoming study.

Acknowledgements

This study is partly funded by: Bourgogne Region program, CNRS, and STIC AmSud project Geo-Climate XMine, and LAU grant SOERC-1314T012.

References

1. H. Bast and B. Buchhold, *An index for efficient semantic full-text search*, 22nd ACM Int. Conf. on CIKM, 2013, pp. 369–378.
2. A. Burton-Jones and et al., *A heuristic-based methodology for semantic augmentation of user queries on the web*, Conceptual Modeling - ER 2003, LNCS, vol. 2813, 2003, pp. 476–489.
3. C. Carpineto and et al., *Improving retrieval feedback with multiple term-ranking function combination*, ACM Trans. Inf. Syst. **20** (2002), no. 3, 259–290.
4. K. Chandramouli and et al., *Query refinement and user relevance feedback for contextualized image retrieval*, 5th International Conference on Visual Information Engineering, 2008, pp. 453–458.
5. P. Cimiano and et al., *Towards the self-annotating web*, 13th Int. Conf. on WWW, 2004, pp. 462–471.
6. S. Das and et al., *Making unstructured data sparql using semantic indexing in oracle database*, IEEE 29th ICDE (2012), 1405–1416.
7. Erika F. de Lima and Jan O. Pedersen, *Phrase recognition and expansion for short, precision-biased queries based on a query log*, 22nd int. Conf. ACM SIGIR, 1999, pp. 145–152.
8. C. Fellbaum, *Wordnet an electronic lexical database*, MIT Press, May 1998.
9. D. Florescu and et al., *Integrating keyword search into xml query processing*, Comput. Netw. **33** (2000), no. 1-6, 119–135.
10. W. B. Frakes and R. A. Baeza-Yates (eds.), *Information retrieval: Data structures and algorithms*, Prentice-Hall, 1992.
11. G. Grefenstette, *Explorations in automatic thesaurus discovery*, Kluwer Pub., 1994.
12. S. Kumar and et al., *Ontology based semantic indexing approach for information retrieval system*, Int. J. of Comp. App. **49** (2012), no. 12, 14–18.
13. Y. Li and et al., *Term disambiguation in natural language query for xml*, 7th Int. Conf. on FQAS, 2006, pp. 133–146.
14. C. Mishra and N. Koudas, *Interactive query refinement*, 12th Int. Conf. on EDBT, 2009, pp. 862–873.
15. R. Navigli, *Word sense disambiguation: A survey*, ACM Comput. Surv. **41** (2009), no. 2, 10:1–10:69.
16. R. Navigli and G. Crisafulli, *Inducing word senses to improve web search result clustering*, Int. Conf. on Empirical Methods in Natural Language Processing, 2010, pp. 116–126.
17. S. Nguyen and et al., *Semantic evaluation of search result clustering methods*, Intelligent Tools for Building a Scientific Information Platform, Studies in Computational Intelligence, vol. 467, 2013, pp. 393–414.
18. R. Navigli Paola and et al., *Extending and enriching wordnet with ontolearn*, Int. Conf. on GWC 2004, 2004, pp. 279–284.
19. P. Resnik, *Using information content to evaluate semantic similarity in a taxonomy*, 14th Int. Conf. on Artificial intelligence, 1995, pp. 448–453.
20. G. Salton and C. Buckley, *Improving retrieval performance by relevance feedback*, Readings in Information Retrieval, 1997, pp. 355–364.
21. M. Sussna, *Word sense disambiguation for free-text indexing using a massive semantic network*, 2nd int. ACM Conf. on CIKM, 1993, pp. 67–74.
22. P. Velardi and et al., *Ontolearn reloaded: A graph-based algorithm for taxonomy induction*, Computational Linguistics **39** (2013), no. 3, 665–707.
23. E. M. Voorhees, *Query expansion using lexical-semantic relations*, 17th int. ACM conf. on SIGIR, 1994, pp. 61–69.
24. J. Weeds and et al., *Characterising measures of lexical distributional similarity*, 20th int. conf.on Computational Linguistics, 2004.
25. H. Wen and et al., *Clustering web search results using semantic information*, 2009 Int. Conf. on Machine Learning and Cybernetics, vol. 3, 2009, pp. 1504–1509.
26. S. Zhong and et al., *A design of the inverted index based on web document comprehending*, JCP **6** (2011), no. 4, 664–670.