# Upgraded *SemIndex* Prototype supporting Intelligent Database Keyword Queries through Disambiguation, Query As You Type, and Parallel Search Algorithms

Joe Tekli[1], Richard Chbeir[2], Agma J.M. Traina[3], Caetano Traina Jr.[3],
Kokou Yetongnon[4], Carlos Raymundo Ibanez[5], and Christian Kallas[1]
[1] Lebanese American University (LAU), ECE Dept., Byblos, Lebanon
[2] University of Pau and Adour Countries (UPPA), LIUPPA Lab., Anglet, France
[3] University of Sao Paulo (USP), ICMC, Sao Carlos, Brazil
[4] University of Bourgogne (UB), LE2I Lab. UMR-CNRS, Dijon, France
[5] Peruvian University of Applied Sciences, Lima, Peru

***Abstract*—This paper describes an upgraded version of the *SemIndex* prototype system for semantic-aware search in textual SQL databases. Semantic-aware querying has emerged as a required extension of the standard containment keyword-based query to meet user needs in textual databases and IR applications. Here, we build on top of *SemIndex*, a semantic-aware inverted index previously developed by our team, to allow semantic-aware search, result selection, and result ranking functionality. Various weighting functions and intelligent search algorithms have been developed for that purpose and will be presented here. A graphical interface was also added to help end-users write and execute queries. Preliminary experiments highlight *SemIndex* querying effectiveness and efficiency, considering different querying algorithms, different semantic coverages, and a varying number of query keywords.**

***Keywords*—Semantic Queries, Inverted Index, Semantic Network, Textual Database, Semantic Search, Disambiguation.**

## I. Introduction

### A. Context

Processing keyword-based queries is a fundamental problem in the domain of Information Retrieval (IR) [3, 5, 16]. 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 [3], supported by many relational DBMSs, and then extended toward semi-structured and unstructured data to support keyword-based queries.

In a previous study [12], we proposed *SemIndex*: a semantic-aware inverted index model designed to process semantic-aware queries. An extended query model with different levels of semantic awareness was defined, so that both semantic-aware queries and standard containment queries can be processed within the same framework. Fig. 1 illustrates the overall framework of the *SemIndex* approach and its main components. Briefly, the Indexer manages the index generation and maintenance, while the Query Processor processes and answers semantic-aware (or standard) queries issued by the user using *SemIndex* component.

### B. Goal and Contributions

While the study in [12] introduced the core logical design of *SemIndex,* the goal of our current paper is to shed the light on upgrades to the *SemIndex* framework and components. At the indexer level, we add: i) dedicated weight functions, associated with the different elements of the *SemIndex* graph, allowing more sophisticated semantic query result selection and ranking, coupled with ii) a dedicated relevance scoring measure, required in the query evaluation process in order to retrieve and rank relevant query answers. At the query processing level, we develop iii) different alternative query processing algorithms (in addition to the main algorithm), and iv) a dedicated GUI interface allowing user to easily manipulate the prototype system.

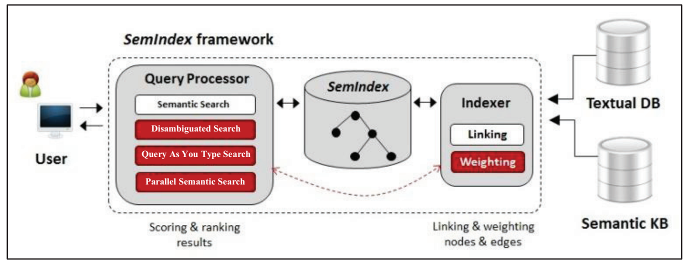



**Fig. 1.** *SemIndex* Framework

The remainder of the paper is organized as follows: Section II provides a literature review. Section III briefly reminds *SemIndex*'s logical design model. Section IV describes *SemIndex*'s weight functions and relevance scoring measure. Section V develops the newly introduced querying algorithms. Section VI highlights added prototype functionality and describes the experimental evaluation, before concluding in Section VII with ongoing works.

## II. Literature Review

Early approaches on keyword search queries for RDBs uses traditional IR scores (e.g., TF-IDF) to find ways to join tuples from different tables in order to answer a given keyword query [2, 7, 17]. The proposed search algorithms focus on enumeration of join networks called *candidate networks*, to connect relevant tuples by joining different relational tables. The result for a given query comes down to a sequence of candidate networks, each made of a set of tuples containing the query keywords in their text attributes, and connected through their primary-foreign key references, ranked based on candidate network size and coverage. More recent methods on RDB full-text search in [22, 23] focus on more meaningful scoring functions and generation of top-*k* candidate networks of tuples, allowing to group and/or expand candidate networks based on certain weighting functions in order to produce more relevant results. The authors in [24] tackle the issue of keyword search on streams of relational data, whereas the approach in [35] introduces keyword search for RDBs with star-schemas found in OLAP applications. Other approaches introduced natural language interfaces providing alternate access to a RDB using text-to-SQL transformations [20, 30], or extracting structured information (e.g., identifying entities) from text (e.g., Web documents) and storing it in a DBMS to simplify querying [14, 15]. Keyword-based search for other data models, such as XML [1, 5] and RDF [6, 8] have also been studied.

More recent approaches, e.g., [25, 29, 31, 34], have developing so-called *semantic-aware* or *knowledge-aware* (keyword) query systems, which have emerged since the past decade as a natural extension to traditional containment queries, encouraged by (non-expert) user demands. Most existing works in this area have incorporated semantic knowledge at the *query processing* level, to: i) pre-process queries using query rewriting/relaxation and query expansion [9, 25, 33], ii) disambiguate queries using semantic disambiguation and entity recognition techniques [9, 21, 29], and/or

iii) post-process query results using semantic result organization and re-ranking [29, 31, 34]. Yet, various challenges remain unsolved, namely: i) time latencies when involving query pre-processing and post-processing [25, 33], ii) complexity of query rewriting/relaxation and query disambiguation requiring context information (e.g., user profiles or query logs) which is not always available [10, 28], and iii) limited user involvement, where the user is usually constrained to providing feedback and/or performing query refinement after the first round of results has been provided by the system [11, 28].

Our work is complementary to most existing DB search algorithms in that our approach extends *syntactic* keyword-term matching: where only tuples containing exact occurrences of the query keywords are identified as results, toward *semantic* based keyword matching: where tuples containing terms which are lexically and semantically related to query terms are also identified as potential results, a functionality which - to our knowledge - remains unaddressed in most existing DB search algorithms. We build on an adapted index structure able to integrate and extend textual information with domain knowledge (not only at the querying level, but rather) at the most basic *data indexing* level, providing a semantic-aware inverted index capable of supporting semantic-based querying, and allowing to answer most challenges identified above.

## III. *SemIndex* Design Model

*SemIndex*'s logical design consists of a semantic knowledge graph structure, combining two graph representations for each of the input resources: i) textual data collection (e.g., IMDB) and ii) reference knowledge base (e.g., WordNet) into a single graph structure. Consider for instance the sample data collection in Table 1. Fig. 2 shows an extract from an inverted index built on the sample movie database in Table 1, where data objects $O_1$, $O_2$, and $O_3$ have been indexed using index terms extracted from the database, sorted in alphabetic order. It is important to note that this simple index is typically used to answer containment queries [3], aiming at finding data objects that contain one or more terms.

**Table 1.** Sample Movie data collection extracted from IMDB.

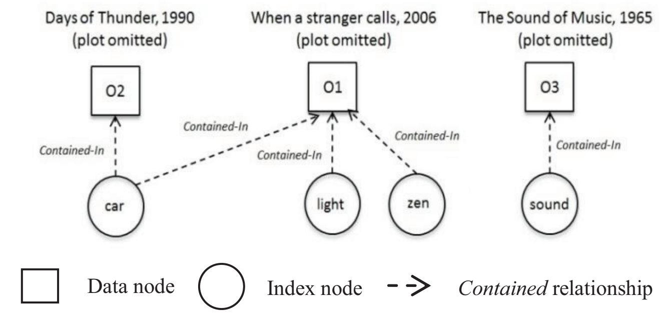
| ID | Textual content |
|---|---|
| $O_1$ | <u>When a Stranger Calls</u> (**2006**): *A young high school student babysits for a very rich family. She begins to receive strange phone calls threatening the children...* |
| $O_2$ | <u>Days of Thunder</u> (**1990**): *Cole Trickle is a young racer from California with years of experience in open-wheel racing winning championships in Sprint car racing...* |
| $O_3$ | <u>Sound of Music, The</u> (**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...* |

When a keyword query mapping two or more index terms must be processed, the corresponding posting lists are read and merged. The index terms and their mappings with the data objects can be generated using classical Natural Language Processing (NLP) techniques (including stemming, lemmatization, and stop-words removal) [27], which could be either embedded in the DBMS or supplied by a third-party provider.

An extract from the WordNet knowledge graph is shown in Fig. 3, where $S_1$, $S_2$ and $S_3$ represent senses/concepts (i.e., synsets in WordNet), and their string values (i.e., the synsets' glosses/definitions), and $T_1$, $T_2$, …, $T_{11}$ represent terms, and their string values (i.e., literal words/expressions) shown alongside the corresponding nodes.

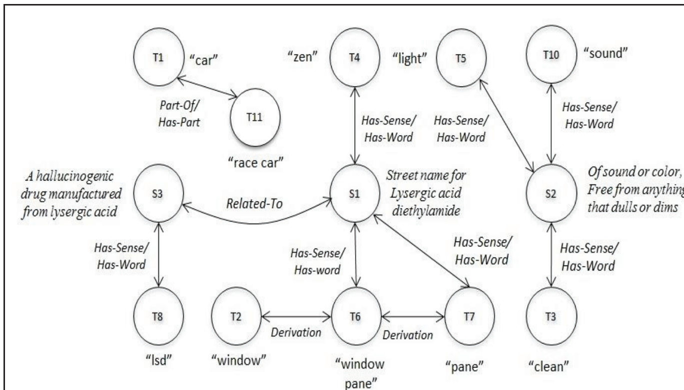
| Term | Object IDs[ ] |
|---|---|
| "car" | ⟨O1, O2⟩ |
| "light" | ⟨O1⟩ |
| "sound" | ⟨O3⟩ |
| "zen" | ⟨O1⟩ |
| … | … |

**a.** Inverted index *InvIndex*(Δ).




**b.** *SemIndex* graph representing *InvIndex*(Δ).

**Fig. 2.** Sample inverted index (a) and corresponding *SemIndex* graph (b), based on the textual collection in Table 1.



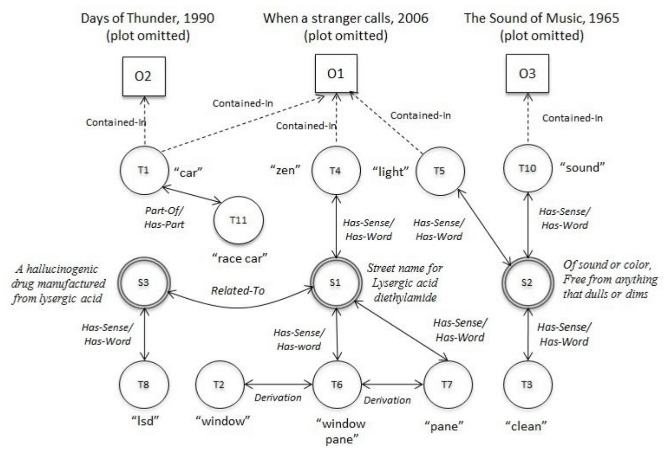

**a.** Sample $G_{KB}$ graph representing a *KB* extract from WordNet.

| Term | Sense IDs[] |
|---|---|
| "acid" | ⟨S1, S3⟩ |
| "clean" | ⟨S2⟩ |
| "light" | ⟨S2⟩ |
| "lsd" | ⟨S3⟩ |
| "lysergic" | ⟨S1, S3⟩ |
| … | … |

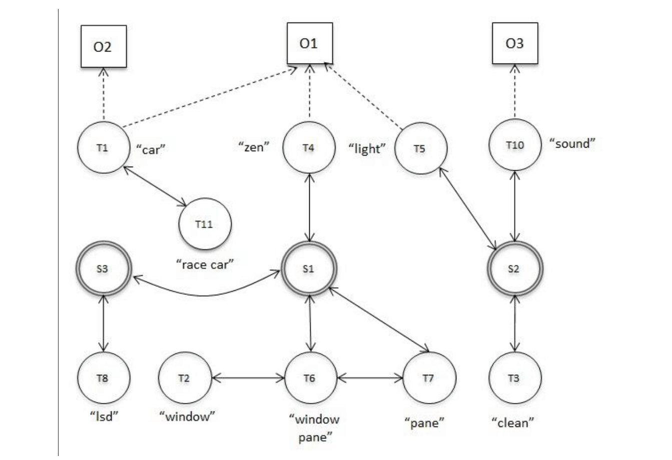
**b.** Extract of inverted index *InvIndex*($G_{KB}$) connecting terms in $G_{KB}$ with corresponding senses (to speed up term/synset lookup)

**Fig. 3.** Extract from the knowledge graph of WordNet, with corresponding inverted index.

Following [12], a *SemIndex* graph consists of an extended knowledge graph made of data nodes to represent data objects (e.g., movies), index nodes to represent senses/concepts (e.g., synsets) and terms (e.g., words/expressions), as well as corresponding data and index relations represented as graph edges. For instance, a sample *SemIndex* graph is shown in Fig. 4 built based on the textual collection from Table 1 and the *KB* extract in Fig. 3. It comprises 3 data nodes ($O_1$ – $O_3$), 3 index *sense* nodes ($S_1$ – $S_3$), and 11 index *term* nodes ($T_1$ – $T_{11}$), along with corresponding data edges and index edges.




**a.** *SemIndex* graph before removing edge labels and string values.




**b**. Final *SemIndex* graph representation, trimmed by removing edge labels and string values which will not be used in query evaluation.

**Fig. 4.** Extract of *SemIndex* graph obtained after coupling the data collection and the knowledge base graphs in Fig. 2 and 3.

## IV. INDEX WEIGHT FUNCTIONS

After indexing and coupling the textual resource and the semantic resource into a unified *SemIndex* graph, we introduce a set of weighting functions to assign weight scores to *SemIndex*' entries, including: *data nodes*, *index nodes*, *data edges* and *index edges*. Other weight functions could be later added to cater to the index designer's needs.

### A. Index Node Weight

Considering an index node $n_i$ in the *SemIndex* graph, the weight of $n_i$ is computed according to the below formula where we consider "Fan-in" to be the number of nodes connected with the target index node:

$$\mathrm{W}_{\mathrm{IndexNode}}(\mathrm{n_i}) = \frac{Fan-in(n_i)}{\underset{\forall v_j \in V_{index}}{Max}\ (Fan-in(n_j))} \quad \in [0,1] \tag{1}$$

The rational here is that an index node is more important if it receives more links from other indexing nodes.

### B. Index Edge Weight

Given an index edge $e_i^j$ incoming from index node $n_i$ and outgoing toward index node $n_j$ in the *SemIndex* graph, we define the weight of $e_i^j$ as follows:

$$\mathrm{W}_{\mathrm{IndexEdge}}(e_i^j) = \frac{1}{Fan-out_{Label}(n_i)} \quad \in\ ]0,1] \tag{2}$$

The weight of an index edge inversely proportional to the number of outgoing links from a certain index node to another, taking into account the semantic relation type of the index link at hand. The rationale here is that an index edge designates a stronger connection between two index nodes when it carries most of the descriptive power from the source node to the destination node, such that the source node has few other out-going connections.

### C. Data Node Weight

The weight of a data node $n_d$ in the *SemIndex* graph is defined as:

$$\mathrm{W}_{\mathrm{DataNode}}(n_d) = \frac{Fan\text{-}In(n_d)}{\mathrm{Max}(Fan\text{-}In(n_q \in G_{SI}))} \quad \in [0,1] \tag{3}$$

where $Fan\text{-}In(n_d)$ designates the number of foreign key/primary key data links (joins) outgoing from data nodes (tuples) $\forall\ n_q \in G_{SI}$ where the foreign keys reside, toward data node (tuple) $n_d$ where the primary key resides. Here, similarly to index node weight, the rational is that a data node is more important if it receives more links from other data nodes.

### D. Data Edge Weight

Given a data edge $e_i^d$ connecting an index node $n_i$ with a data node $n_d$ (e.g., data edge connecting index node $T_1$ with data node $O_2$ since the term "*car*" occurs in the textual description of $O_2$, likewise for $T_1$-$O_2$, $T_4$-$O_1$, $T_5$-$O_1$ and $T_{10}$-$O_3$ in Fig. 4), we compute the weight of $e_i^d$ as a typical TF-IDF (Term Frequency Inverse Document Frequency) score where TF underlines the frequency (number of occurrences) of the index node string literal within a given data node, connected via the data edge in question, and IDF underlines the number of data edges connecting the same index node with other data nodes (i.e., the fan-out of the index node in question). Hence, given a data edge $e_i^d$ incoming from index node $n_i$ toward data node $n_d$, we define:

$$\mathrm{W}_{\mathrm{DataEdge}}(e_i^d) = \mathrm{TF}(n_i.l,\ n_d) \times \mathrm{IDF}(n_i.l) \tag{4}$$

TF and IDF are calculated as follows:

$$\mathrm{TF}(n_i.l,\ n_d) = \frac{\mathrm{NbOcc}(n_i.l)}{\underset{\mathrm{n_k} \in \mathrm{e}_k^d}{Max}(\ \mathrm{NbOcc}(n_k.l))} \quad \in [0,\ 1] \tag{5}$$

TF is normalized w.r.t.[1] the maximum number occurrences of any index node string literal $n_k.l$ within the target data node $n_d$:

$$\mathrm{IDF}(n_i.l,\ G_{SI}) = 1 - \frac{DF(n_i.l, G_{SI})}{N} \quad \in [0,\ 1] \tag{6}$$

where $N$ is the total number of data nodes in the *SemIndex* graph, and $DF(n_i.l,\ G_{SI})$ is the number of data nodes in the graph containing at least one occurrence of $n_i.l$.

[1] with respect to

### E. Relevance Scoring Measure

The scores of index/data nodes/edges returned as query answers are computed using typical *Diskstra*-style shortest distance computations. Yet, instead of identifying the shortest (smallest) distance, we identify as answers data objects having the maximum similarity (similarity being the inverse function of distance) w.r.t. the starting index nodes (mapping to keyword queries). In other words, given the sample node linkage in Fig. 5, with data node $n_d$ and starting index node $n_i$, we define the relevance score of $n_d$ w.r.t. $n_i$ as follows:

$$\text{score}(n_d, n_i) = \frac{\left[\begin{array}{l} \mathrm{W}_{\text{DataNode}}(n_d) \times \mathrm{W}_{\text{DataEdge}}(e_p^d) \times \dfrac{1}{d(n_p, n_d)} + \\ \quad \mathrm{W}_{\text{IndexNode}}(n_p) \times \mathrm{W}_{\text{IndexEdge}}(e_j^p) \times \dfrac{1}{d(n_j, n_d)} + \\ \qquad \mathrm{W}_{\text{IndexNode}}(n_j) \times \mathrm{W}_{\text{IndexEdge}}(e_i^j) \times \dfrac{1}{d(n_i, n_d)} \end{array}\right]}{d(n_i, n_d)} \in [0,1] \quad (7)$$

where $d(n_i, n_j)$ is the distance in number of edges between two nodes. In other words, in the following example, $d(n_p, n_d) = 1$, $d(n_j, n_d) = 2$, and $d(n_i, n_d) = 3$.

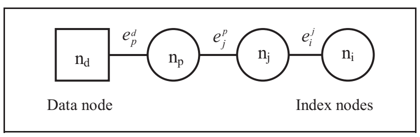



**Fig. 5.** Sample node linkage in the *SemIndex* graph

## V. QUERYING ALGORITHMS

Our upgraded prototype includes four query processing algorithms: i) the core algorithm and three other variants designed to improve: ii) query performance, iii) user involvement, and iv) query efficiency:

i. Core algorithm: titled *SemIndex Semantic Search* (*SI_SS*) originally developed in [12], performs semantic-aware search using shortest path navigation in the *SemIndex* graph,

ii. Query performance: *SemIndex Disambiguated Search* (*SI_DS*), integrates semantic disambiguation within the semantic search process, aiming to improve querying result quality,

iii. User involvement: *SemIndex Query-As-You-Type Search* (*SI_QAYTS*), allows users to manually choose the meanings of query keywords before performing semantic search, aiming to involve the user in improving search result quality,

iv. Query Efficiency: *SemIndex Parallel Semantic Search* (*SI_PSS*) is a parallel processing (multithreading) version of *SI_SS*, aiming to reduce query execution time.

### A. SemIndex Semantic Search (SI_SS)

This is the original data search algorithm developed for *SemIndex*. The algorithm's pseudo-code is provided in Fig. 6, and consists of the following main steps:

1- Identify the index (*searchable term*) nodes mapping to each query term (lines 1-4),
2- Identify, for each of the selected index nodes, the minimum distance paths at distance $\ell$, i.e., using *Dijkstra*'s shortest path algorithm (lines 5-7)
3- Identify the shortest paths which contain data edges linking to data nodes, and then add the resulting data nodes to the list of output data nodes (line 8)
4- Merge the resulting data nodes with the list of existing answer data nodes. Each answer node is then assigned a score by adding its distance from every query term index node. The algorithm finally returns the list of answer data nodes ranked by order of path scores in ascending order (lines 9-12).

```
Algorithm SI_SS      // SemIndex Semantic Search
Input:   G_SI        // SemIndex graph
         T           // Set of query selection terms
         ℓ           // Link distance value, designating semantic coverage
         {r, k}      // Range and k-nearest neighbor selection operators
Ouput: N_d_Out       // List of query answers
Begin
1     N_d_Out = ϕ
2     For each query term T_i ∈ T
3     {
4        Step 1: N_i_In = getNodeIDs(T_i, G_SI)
5        For each searchable node n_i ∈ N_i_In
6        {
7            Step 2: SP = findShortestPaths(n_i, ℓ, G_SI)
8            Step 3: N_d_ni = getDataNodeIDs(SP, G_SI)
9            Step 4: N_d_Out = mergeAndRank(N_d_ni , N_d_Out)
10       }
11    }
12    Step 5: N_d_Out = select(N_d_Out, {range, kNN})
      Return N_d_Out
End
```

**Fig. 6.** Pseudo-code of algorithm *SemIndex Semantic Search* (*SI_SS*)

### B. SemIndex Disambiguated Search (SI_DS)

This algorithm consists in disambiguating query keywords, to pinpoint the corresponding indexing nodes (synsets) in the *SemIndex* graph $G_{SI}$, before applying semantic search, in an attempt to reduce *SemIndex* graph navigation time.

On one hand, the core *SI_SS* algorithm considers all occurrences of each query keyword as starting nodes for query processing. For instance, query $q$ = {"*pane*", "*clean*"} consisting of two keywords: "*pane*" and "*clean*", would result in 4 starting index nodes: 3 index term nodes corresponding to the three possible meanings of "*pane*" (following the WordNet semantic reference [26], used as reference *KB* in creating our current $G_{SI}$) and one index term node corresponding to the single meaning of term "*clean*" (following WordNet). Hence, applying *SI_SS* in this case requires navigating $G_{SI}$ from 4 starting points, until reaching potential answers (i.e., data nodes reachable from both terms within the user specified link distance $\ell$).

On the other hand, *SI_DS* aims to reduce the number of starting nodes, in order to navigate $G_{SI}$ and reach potential query answers faster. Its pseudo-code is shown in Fig. 7, and consists of the following main steps:

1- Perform WSD on query terms using the Adapted LESK algorithm [4] (lines 1-4),
2- Identify in the *SemIndex* graph the index nodes of disambiguated senses (line 5),
3- Run the resulting query, starting from the identified index nodes, as a typical keyword containment query on *SemIndex* (similarly to *SI_SS,* lines 6-11).

```
Algorithm SI_DS   // SemIndex Disambiguated Search
Input: G_SI     // SemIndex graph
        TA      // Set of query selection terms
        ℓ       // Link distance value, designating semantic cover
      {r, k}    // Range and k-nearest neighbor selection operators
Ouput: Nd_Out   // List of query answers
Begin
   Nd_Out = ϕ
   For each query term Ti ∈ T
   {
         Step 0: Si = SimpleLesk(Ti, T, G_SI)
         Step 1: ni = getNodeID(Si, G_SI)
         Step 2: SP = findShortestPaths(ni, ℓ, G_SI)
         Step 3: Nd_ni = getDataNodeIDs(SP, G_SI, Ai)
         Step 4: Nd_Out = mergeAndRank(Nd_ni , Nd_Out)
   }
   Step 5: Nd_Out = select(Nd_Out, {range, kNN})
   Return Nd_Out
End
```

**Fig. 7.** Pseudo-code of *SemIndex Disambiguated Search* (*SI_DS*)

### C. *SemIndex Query As You TypeSearch* (*SI_QAYTS*)

This algorihtm allows the user to choose the proper meaning for every query keyword, by allowing her to choose the intended sense from the set of all possible senses provided by WordNet. Once the senses have been chosen, the algorithm pinpoints in the *SemIndex* graph the indexing nodes corresponding to the chosen senses, and then runs typical shortest path search starting from the chosen index nodes. The pseudo-code of *SI_QAYTS* is basically the same as that of *SI_DS*, except for *step 0* which becomes: $S_i$ = Manual($T_i$, T, $G_{SI}$), i.e., allowing the user to manually choose the proper meaning of every query term, among the list of possible meanings presented to the user through the system's GUI (cf. Fig. 10). Then, *SI_DS* resumes by identifying and only processing the starting index nodes corresponding to the term senses (synsets) chosen by the user. The algorithm's main steps can be described as follows:

1- Allow the user to choose the sense of each term in the query according to WordNet,
2- Identify in the *SemIndex* graph the index nodes corresponding to the chosen senses,
3- Run the resulting query, starting from the identified index nodes, as a typical *SI_SS* keyword containment query on *SemIndex.*

### D. *SemIndex Parallel Semantic Search* (*SI_PSS*)

We have also introduced a parallelized version of algorithm *SI_SS* (cf. Fig. 8), which preserves (more or less) the same workflow of the original algorithm except that it processes query terms and starting index nodes using multiple threads running in parallel. The algorithm's main steps are described as follows:

1- Every query term is assigned a dedicated thread, and is thus processed independently from other threads (lines 1-2).
2- After identifying the starting nodes for a query term (line 4), every starting node is then assigned its own dedicated thread (line 5), allowing to: compute the shortest paths from the starting node to data nodes in the *SemIndex* graph (line 7), and then identify the reached data nodes designating potential query answers (line 8).
3- Results are gradually merged (line 9) as they are produced by each thread, to rank and select (lines 10-12) query answers.

The physical implementation of algorithm *SI_PSS* is configured to run as many threads as there are terms in the user query, where thread scheduling and parallel execution is left to the operating system.

```
Algorithm SI_PSS   // Parallel_Semantic_Search
Input:  G_SI     // SemIndex graph
         TA      // Set of query selection terms
         ℓ       // Link distance value, designating semantic coverage
       {r, k}    // Range and k-nearest neighbor selection operators
Ouput: Nd_Out    // List of query answers
Begin
1     Nd_Out = ϕ
2     Create Thread for each query term Ti ∈ T
3     {
4        Step 1: Ni_In = getNodeIDs(Ti, G_SI)
5        Create Thread for each ni ∈Ni_In
6        {
7              Step 2: SP = findShortestPaths(ni, ℓ, G_SI)
8              Step 3: Nd_ni = getDataNodeIDs(SP, G^SI, Ai)
9              Step 4: Nd_Out = mergeAndRank(Nd_ni , Nd_Out)
10       }
11       Step 5: Nd_Out = select(Nd_Out, {range, kNN})
12       Return Nd_Out
End
```

**Fig. 8.** Pseudo-code of *SemIndex Parallel Semantic Search* (*SI_DS*)

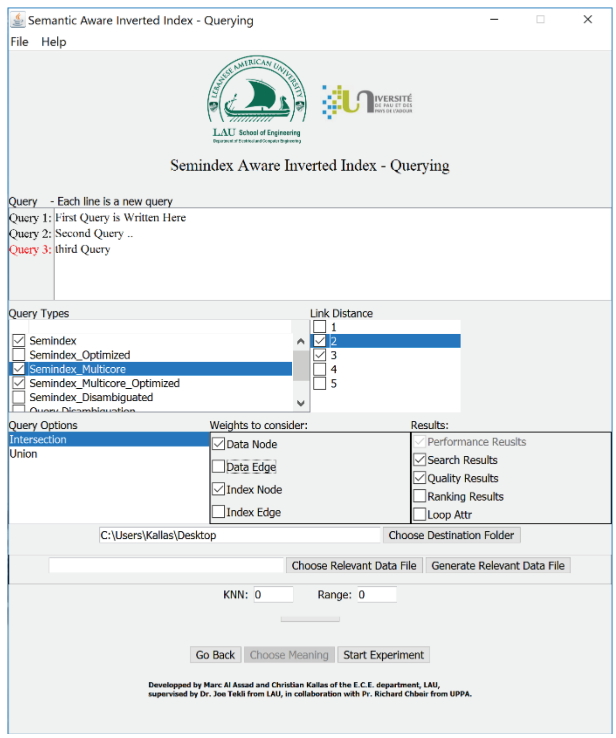



**Fig. 9.** Main querying interface in upgraded *SemIndex* prototype

## VI. Experimental Evaluation

### A. *Prototype System*

We have implemented our new *SemIndex* algorithms and functionality using Java, and have used MySQL 5.6 as an RDBMS to store the data collection. The prototype's GUI has also been upgraded to handle multiple options (launching multiple queries simultaneously) for customized search. Data sheets (in the form of Excel files) are automatically generated after every run of the system, storing all statistical and experimental data pertaining to the queries executed, including: CPU and SQL execution times, memory consumption, number of nodes visited in the *SemIndex* graph, number of retrieved results, as well as a various experimental metrics including *precision*, *recall*, *f-value*, and *mean average precision* (MAP). The main querying interface is shown in Fig. 9, where one can see multiple querying options such as: *Query Types*, *Link Distance*, among others. New interfaces have also been developed to allow additional functionality such as: a dedicated interface allowing the user to choose proper meanings (synsets) for query keywords when running the *Query-As-You-Type* algorithm (Fig. 10), as well as an interface to select relevant (versus non-relevant) data objects (Fig. 11) to be used as reference for computing experimental metrics (e.g., precision, recall, cf. Section VI.B).

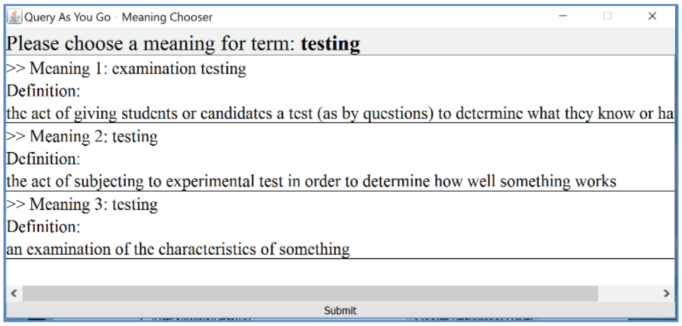



**Fig. 10.***Query-As-You-Type* sub-interface

## B. Experimental Scenario

We evaluated the performance of our *SemIndex* querying algorithms by assessing their: i) query processing time, and ii) the quality of returned results. We used IMBD *movies* dataset[2] as an average-scale[3] input textual collection, including attribute *movie_id* and the combined textual contents of attributes *title*, *year*, *plot*, and *info*, with a total size of around 75 MBytes including more than 143k data (movie) objects, and more than 7 million index terms. We used WordNet 3.0 as our reference knowledge base, with a total size of around 26 Mbytes, including more than 117k synsets (senses). Detailed descriptions of both the textual collection and the knowledge base are provided in a technical report [32].

We formulated different queries organized in two categories: i) *unrelated queries* and ii) *expanded queries*.

**Table 2.** Sample test queries used in our experiments.

| **Query group *Q1* – *Unrelated queries*** | | **Query group *Q2* – *Expanded queries*** | |
|---|---|---|---|
| **ID** | **Terms** | **ID** | **Terms** |
| Q1_1 | "time" | Q2_1 | "car" |
| Q1_2 | "love", "date" | Q2_2 | "car", "muscle" |
| Q1_3 | "fly", "power", "man" | Q2_3 | "car", "muscle", "classic" |
| Q1_4 | "robot", "human", "war", "world" | Q2_4 | "car", "muscle", "classic", "speed" |
| Q1_5 | "west","cowboy","peacekeeper","sheriff", "law" | Q2_5 | "car","muscle","classic","speed","thrills" |

The first category consists of queries with varying numbers of selection terms (keywords), e.g., from 1 (single term query) to 5, where all terms are different and all queries are unrelated (i.e., queries with no common selection terms, cf. sample query group *Q1* in Table 2). The second category consists of queries with varying numbers of selection terms, where terms are different yet queries are related: such that each query expands its predecessor by adding an additional selection term to the latter (cf. sample query group *Q2*).

Tests were carried out on a PC with an *Intel I7* system with 2.9 GHz CPU, 8GB RAM memory, and a 500 GB built-in NTFS disk drive. The database (IMDB), knowledge graph (WordNet), and index files were stored on the disk drive's main partition.

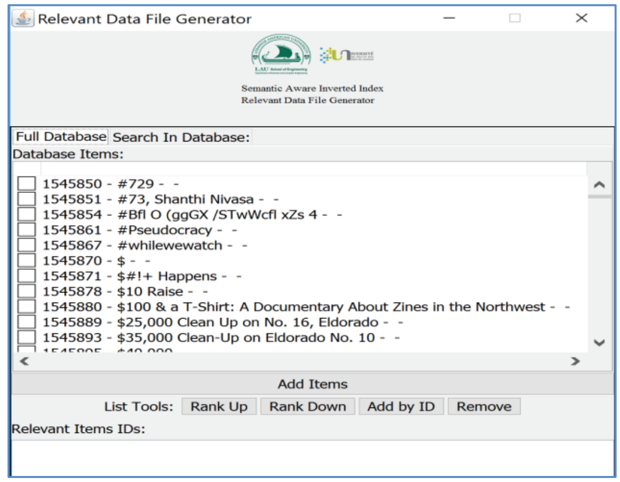



**Fig. 11.** *Relevant data objects generator* interface

## C. Query Processing Time

We ran the same queries through the four *SemIndex* querying algorithms: *SI_SS*, *SI_DS*, *SI_QAYTS*, and *SI_PSS*. Fig. 12 provides average processing time results for all queries, plotted by varying the number of query terms *k* and *SemIndex* link distance threshold $\ell$. Note that when considering $\ell=1$, algorithm *SI_SS* comes down to performing traditional keyword containment search using a traditional inverted index (cf. Fig. 2).

First, results of all four algorithms show that query execution time increases almost linearly with the number of query terms *k* (when fixing link distance $\ell$), and increases linearly with $\ell$ (when fixing *k*), highlighting the algorithms quadratic complexity levels. Second, results show that all four algorithms have very close query time levels when both *k* and $\ell$ are small (=1 and 2), such that time difference increases as both *k* and $\ell$ increase. This is due to the fact increasing either *k* or $\ell$ means increasing the number of nodes to be navigated in the *SemIndex* graph: increasing *k* means navigating the *SemIndex* graph starting from a larger number of initial nodes, and increasing $\ell$ means reaching deeper into the *SemIndex* graph structure to identify more semantically relevant results. Third, one can clearly realize that the most time consuming algorithm is *SI_DS* due to the overhead it adds to process the different possible meanings of every query term (for disambiguation) before navigating the *SemIndex* graph. Algorithms *SI_SS* and *SI_QAYTS* produced almost identical time levels (disregarding the manual effort required in *SI_QAYTS*[4]), whereas our parallel processing *SI_PSS* algorithm is clearly the most efficient of its counterparts, requiring almost 50% less time than *SI_DS* and almost 33.34% less time than *SI_SS*/*SI_QAYTS* with maximum $k$=5 and $\ell$=5.

## D. Query Result Quality

Table 3 shows the *precision*, *recall*, *f-value* and *MAP* results obtained with the four *SemIndex* querying algorithms, averaged over all test

[2] Internet Movie DataBase raw files are available from online http://www.imdb.com/. We used a dedicated data extraction tool (at http://imdbpy.sourceforge.net/) to transform IMDB files into a RDB.

[3] Tests using large-scale TREC data collections and the Yago ontology as a reference KB are underway within a dedicated study.

[4] *SI_QAYTS*'s time shown in Fig. 12 does not encompass the time it took the testers to manually choose the meanings of query terms (which we did not consider to be part of the algorithm itself), but only considers actual algorithm (CPU and SQL) execution time.

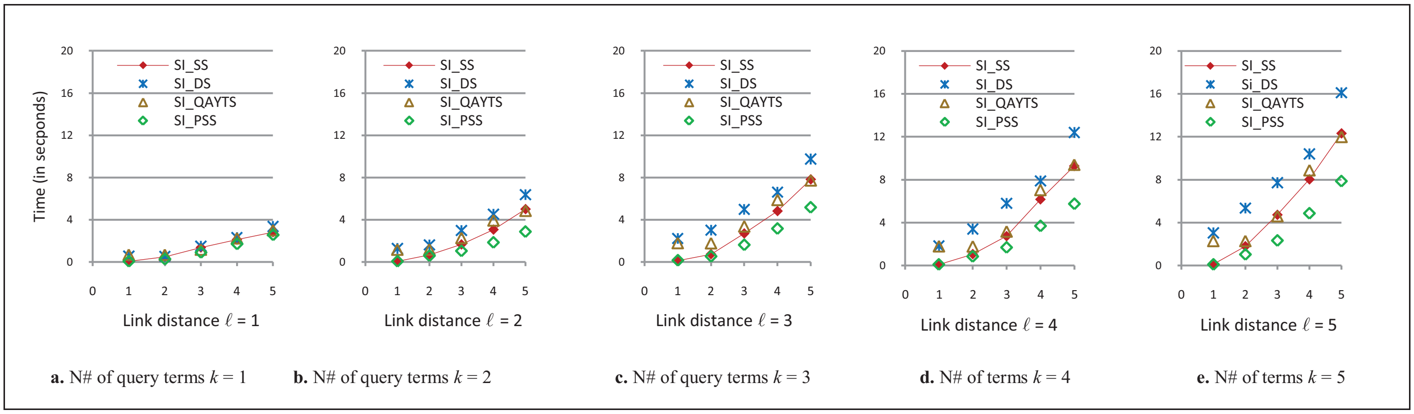



**Fig. 12.** Comparing average query execution time of *SI_SS* with its three variants: *SI_DS*, *SI_QAYTS*, and *SI_PSS*, while varying link distance threshold ℓ, and fixing the number of query terms *k*.

queries. Note that *SI_SS* and *SI_PSS* produce exactly the same query answers (recall that *SI_PSS* is a parallelized version of *SI_SS*), and hence their results are fused together in the below graphs. Results averaged per link distance ℓ and number of query terms *k* are provided in Table 1. These highlight several observations.

1) ***Precision*** and ***recall***: One can realize that *precision* levels with all *SemIndex* algorithms, while fluctuating, generally increase with link distance (ℓ) until reaching ℓ=3 or ℓ=4 where *precision* starts to decrease toward ℓ=5. However, one can realize that *recall* levels steadily increase with ℓ (with reduced fluctuation compared with *precision*). On one hand, this shows that the number of correct (i.e., user expected) results increases as more semantically related terms are covered in the querying process (with ℓ > 1). On the other hand, this also shows that over-navigating the *SemIndex* graph to link terms with semantically related ones located as far as ℓ ≥ 3 hops away might include results which: i) are somehow semantically related to the original query terms, but which ii) are not necessarily interesting for the users. For instance, term *"congo"* (meaning: *black tea grown in China*) is linked to term *"time"* through ℓ=5 hops in *SemIndex* ("*time*" >> "*snap*" >> "*reception*" >> "*tea*" >> "*congo*"). Yet, results (movie objects) containing term *"congo"* (e.g., movies about the country *Congo*, or its continent *Africa*) were not judged to be relevant by human testers when applying query *"time"* (testers were probably expecting movies about the *passage of time* or *time travel* instead, etc.)[5]. Many such examples occurred when running multiple term queries such as *Q1_4* (consisting of terms "*robot*", "*human*", "*war*", "*world*")[6], where movies like *The Taking of Pelham One Two Three*[7] and *Showtime*[8] (among others) where returned as results by *SemIndex*'s *SI_SS* when reaching ℓ=5. Such results were deemed not relevant by the testers since they do not correspond to the semantics of the query. Note that returning noisy (incorrect) results along with correct ones does not affect *recall*, but rather affects *precision*.

2) ***F-value*** and ***MAP***: levels clearly increase with the increase of link distance ℓ, whereas they show the same fluctuating behavior with the increase of the number of keywords *k*. First, *f-value* levels confirm the *precision* and *recall* levels obtained above, where the determining factor affecting retrieval quality remains link distance ℓ, whereas an increase in the number of keywords *k* tends to reduce system *recall* with lower values of ℓ and increase *recall* with higher values of ℓ. Second, *MAP* levels seem to concur with those of *f-value*, such that the ranking of relevant results compared with non-relevant ones in the queries' result lists seems to increase with the increase of ℓ and fluctuate (based on the values of ℓ) with the increase of *k*. In other words, increasing ℓ not only allowed retrieving more relevant results, but also allowed dropping non-relevant ones (from the selected top 100 results of the query result list), and consequently improved the ranking of relevant results w.r.t. non-relevant ones in the query result list.

**Table 3.** *Precision*, *recall*, *f-value*, and *MAP* results obtained with *SemIndex* query processing algorithms, averaged per link distance (ℓ) and per number of terms (*k*) (graphs are provided in the technical report [32]).

**a.** Average *precision* (*PR*) results

| | ℓ=1 | ℓ=2 | ℓ=3 | ℓ=4 | ℓ=5 | k=2 | k=3 | k=4 | k=5 | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| **SI_(P)SS** | 0.2758 | 0.3234 | 0.5193 | 0.3805 | 0.3189 | 0.4226 | 0.3632 | 0.2502 | 0.4184 | **0.3636** |
| **SI_DS** | 0.0833 | 0.4259 | 0.4487 | 0.3563 | 0.2866 | 0.3595 | 0.1941 | 0.2218 | 0.5053 | **0.3202** |
| **SI_QAYST** | 0.2758 | 0.2758 | 0.3762 | 0.1687 | 0.1678 | 0.7882 | 0.5268 | 0.1760 | 0.1946 | **0.2528** |

**b.** Average *recall* (*R*) results

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **SI_(P)SS** | 0.0327 | 0.0570 | 0.1487 | 0.3358 | 0.4684 | 0.2570 | 0.2219 | 0.1656 | 0.1895 | **0.2085** |
| **SI_DS** | 0.0063 | 0.0288 | 0.1191 | 0.3551 | 0.5026 | 0.1632 | 0.1651 | 0.2586 | 0.2227 | **0.2024** |
| **SI_QAYST** | 0.0327 | 0.0327 | 0.0508 | 0.0801 | 0.1192 | 0.1769 | 0.1448 | 0.0629 | 0.0361 | **0.0631** |

**c.** Average *f-Value* results

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **SI_(P)SS** | 0.1543 | 0.1902 | 0.3340 | 0.3581 | 0.3711 | 0.3398 | 0.2888 | 0.1937 | 0.3038 | **0.2815** |
| **SI_DS** | 0.0448 | 0.2274 | 0.2839 | 0.3507 | 0.3403 | 0.2614 | 0.1725 | 0.2000 | 0.3637 | **0.2494** |
| **SI_QAYST** | 0.1543 | 0.1543 | 0.2135 | 0.1225 | 0.1327 | 0.2896 | 0.1958 | 0.0672 | 0.0692 | **0.1554** |

**c.** Average *mean average precision* (*MAP*) results

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **SI_(P)SS** | 0.0273 | 0.0443 | 0.0982 | 0.2264 | 0.3002 | 0.1708 | 0.1222 | 0.1377 | 0.1264 | **0.1393** |
| **SI_DS** | 0.0045 | 0.0226 | 0.0711 | 0.2488 | 0.3195 | 0.0829 | 0.0781 | 0.1864 | 0.1856 | **0.1333** |
| **SI_QAYST** | 0.0275 | 0.0275 | 0.0394 | 0.0482 | 0.0605 | 0.0768 | 0.0456 | 0.0289 | 0.0111 | **0.0406** |

3) **Comparing *SemIndex* algorithms**: results show that *SI_SS* (same for *SI_PSS*) surpasses its *SI_DS* and *SI_QAYTS* algorithms in all four *precision*, *recall*, *f-value*, and *MAP* metrics (cf. Table 3). Concerning *SI_SS* (*SI_PSS*) versus *SI_DS*, results indicate that i) navigating the *SemIndex* graph considering all possible meanings (senses) of every query keyword (as done by *SI_SS* and *SI_PSS*) produced more relevant and better ranked results, whereas ii) disambiguating query terms first, and then navigating the graph

[5] Even though testers had difficulty agreeing on the results of single keyword queries as mentioned previously, yet such cases occurring at link distance ℓ=4 or 5 were clearly deemed irrelevant by all testers.

[6] Query *Q1_4* = { "robot", "human", "war", "world"}

[7] 2009 movie starring Denzel Washington and John Travolta, about a train hijacking in New York city.

[8] 2002 comedy movie starring Eddy Murphy and Robert De Niro, about police officers starring in a reality TV show.

starting from the disambiguated senses' nodes (as done by *SI_DS*), produced less relevant and worse ranked results. However, the worst results (in terms of both relevance and ranking) were obtained with *SI_QAYTS*. In fact, in designing *SI_QAYTS*, we intuitively thought that allowing the user to choose the proper meanings (senses) for query terms before processing (before *SemIndex* navigation) would be the most promising approach, especially when the user considers that she/he knows the exact meanings of the terms utilized to formulate the query. Yet, it turns out that choosing the meanings of terms can be a very delicate task in most cases. First, the user might be confused when trying to choose among a large number of very close or semantically related meanings for a given term (e.g., choosing the right meaning for query *Q1_4*'s term "*world*" in WordNet: sense#1 - *everything that exists*, sense#2 – *reality as in how things appear*, sense#3 – *people in general*, sense#4 – *Planet earth*, sense#5 – *the human race*, and three more other senses). Second, the user chosen meaning could be very different from the one intended by the data creator (e.g., when processing query *Q1_5*[9] through *SI_QAYTS*, most users chose for term "*peacekeeper*" its sense#1 in WordNet: *someone who keeps peace*. Yet, we realized that the meaning of "*peacekeeper*" that was more closely related to *Q1_5*'s intended (*golden truth*) results (i.e., western movies) was sense#3: *the pistol of a law officer in the old West*.

To sum up, *SI_SS* (*SI_PSS*) seem(s) more effective than the other alternative algorithms.

## VII. Conclusion

The main goal of our study was to complete the design and development of *SemIndex*'s query evaluation engine. To this end, we upgraded *SemIndex* by designing and implementing new components and functionality, including: i) dedicated weight functions, associated with the different elements of *SemIndex*, allowing semantic query result selection and ranking, coupled with iii) a dedicated relevance scoring measure, required in the query evaluation process in order to retrieve and rank relevant query answers, iii) various alternative query processing algorithms (in addition to the main algorithm), as well as iv) a dedicated GUI interface allowing user to easily manipulate the prototype system. Preliminary experiments highlight *SemIndex*'s effectiveness and efficiency, considering different querying algorithms, different semantic coverages, and a varying number of query keywords.

We are currently conducting an extended experimental study to evaluate *SemIndex*'s properties in terms of i) genericity: to support different types of textual (structured, semi-structured, NoSQL) data collections, and different semantic knowledge sources (general purpose like Yago [18] and Google [19]), ii) effectiveness: evaluating the interestingness of semantic-aware query answers considering different query answer weighting and ranking (result ordering) schemes, in comparison with IR-based indexing, query expansion, and query refinement methods, and iii) efficiency: to reduce the index's building and query processing costs, using customozed multithreading, index fragmentation, and sub-graph mining techniques [13].

## Acknowledgments

This study is partly funded by the National Council for Scientific Research (CNRS-L) – Lebanon, the Lebanese American Unviersity (LAU), and the Research Support Foundation of the State of Sao Paulo (FAPESP), project: MIVisBD_2017.

[9] Query *Q1_5* = {"west", "cowboy", "peacekeeper", "sheriff", "law"}

## References

[1] Agarwal M.K. *et al.*., *Generic Keyword Search over XML Data.* International Conference on Extended DataBase Technology (EDBT'16), 2016. pp. 149-160.
[2] Agrawal S. *et al.*, *Exploiting Web Search Engines to Search Structured Databases.* World Wide Web Conference (WWW'09), 2009. pp. 501-510.
[3] Baeza-Yates R. and Ribeiro-Neto B., *Modern Information Retrieval: The Concepts & Technology behind Search.* Addison-Wesley Professional, 2nd Ed., 2011. p. 944.
[4] Banerjee S. and Pedersen T., *An adapted Lesk algorithm for word sense disambiguation using WordNet.* In Proc. of the International Conference on Intelligent Text Processing and Computational Linguistics, 2002.
[5] Bao Z. *et al.*, *A Query Refinement Framework for XML Keyword Search.* World Wide Web 2017. 20(6):1469-1505.
[6] Bednar Peter *et al.*, *RDF vs. NoSQL databases for the Semantic Web applications.* Inter. Symp. on Applied Machine Intell. and Informatics (SAMI), 2014, 361-364.
[7] Bergamaschi S. *et al.*, *Combining User and Database Perspective for Solving Keyword Queries over Relational Databases.* Info. Systems, 2016. 55: 1-19.
[8] Blanco R. *et al.*, *Effective and Efficient Entity Search in RDF data.* In International Semantic Web Conference (ISWC'11), 2011. pp. 83–97.
[9] Burton-Jones A. *et al.*, *A Heuristic-Based Methodology for Semantic Augmentation of User Queries on the Web.* In Proc. of the Inter. Conference on Conceptual Modeling (ER'03), 2003. pp. 476–489.
[10] Carpineto C. and Romano G., *A Survey of Automatic Query Expansion in Information Retrieval, .* ACM Computing Survey, ACM, NY, USA, 2012. 44(1):1.
[11] Chandramouli K. *et al.*, *Query Refinement and user Relevance Feedback for contextualized image retrieval.* Inter. Conf. on Visual Information Engineering (VIE), 2008. pp. 453 - 458.
[12] Chbeir R. *et al.*, *SemIndex: Semantic-Aware Inverted Index.* East-European Conf. on Advanced Databases and Information Systems (ADBIS'14), 2014. pp. 290-307.
[13] Cheng J. *et al.*, *Fast graph query processing with a low-cost index.* VLDB Journal, 2011. 20(4): 521-539.
[14] Cheng T. *et al.*, *EntityRank: searching entities directly and holistically.* 33rd inter. conf. on Very Large Data Bases (VLDB'07), 2007. pp. 387-398.
[15] Chu E. *et al.*, *A relational approach to incrementally extracting and querying structure in unstructured data.* inter. conf. on Very Large Data Bases (VLDB '07), 2007. pp. 1045-1056
[16] Das S. *et al.*, *Making unstructured data sparql using semantic indexing in oracle database.* In Proceedings of 29th IEEE ICDE Conf., 2012. pp. 1405–1416
[17] Ding B. *et al.*, *Finding top-k min-cost connected trees in databases.* Proceedings of the Inter. Conf. on Data Engineering (ICDE'07), 2007.
[18] Hoffart J. *et al.*, *YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia.* Artif. Intell., 2013. 194: 28-61.
[19] Klapaftis I. and Manandhar S., *Evaluating Word Sense Induction and Disamiguation Methods.* Language Resources & Evaluation, 2013. 47(3):579-605.
[20] Li F. and J. H.V., *Constructing an Interactive Natural Language Interface for Relational Databases.* Proceedings of the VLDB Endowment, 2014. pp. 73-84.
[21] Li Y. *et al.*, *Term Disambiguation in Natural Language Query for XML.* Inter. Conf. on Flexible Query Answering Systems (FQAS), 2006. LNAI 4027, 133–146.
[22] Liu F. *et al.*, *Effective keyword search in relational databases.* Proc. of the 2006 ACM SIGMOD inter. conf. on Management of data, 2006. pp. 563-574
[23] Luo Y. *et al.*, *Spark: top-k keyword query in relational databases.* Proc. of the 2007 ACM Inter. Conf. on Management of Data (SIGMOD-07), 2007, 115-126.
[24] Markowetz A. *et al.*, *Keyword search on relational data streams.* Proc. of the Inter. Conf. on Management of Data (SIGMOD'07), 2007. pp. 605–616.
[25] Martinenghi D. and Torlone R., *Taxonomy-based relaxation of query answering in relational databases.* VLDB Journal, 2014. 23(5):747-769.
[26] Miller G.A. and Fellbaum C., *WordNet Then and Now.* Language Resources and Evaluation, 2007. 41(2): 209-214.
[27] Miller S. *et al.*, *Hidden Understanding Models of Natural Language.* 32nd annual meeting on Ass. for Computational Linguistics, Stroudsburg, USA, 1994, 25–32.
[28] Mishra C. and Koudas N., *Interactive Query Refinement* International Conference on Extending Database Technology (EDBT'09), 2009. pp. 862-873.
[29] Navigli R. and Crisafulli G., *Inducing Word Senses to Improve Web Search Result Clustering.* Inter. Conf. on Empirical Methods in NLP, 2010. pp. 116–126.
[30] Nihalani N. *et al.*, *Natural language Interface for Database: A Brief review.* Inter. Journal of Computer Science Issues, 2011. 8(2):600-608.
[31] Nguyen S.H. *et al.*, *Semantic Evaluation of Search Result Clustering Methods.* Intelligent Tools for Building a Scientific Information Platform, Studies in Computational Intelligence Volume 467, 2013. 467(393-414).
[32] Tekli J. et al., *SemIndex Technical Report.* Available at http://sigappfr.acm.org/Projects/SemIndex/, 2018.
[33] Traina C. Jr. *et al.*, *Efficient Processing of Complex Similarity Queries in RDBMS through Query Rewriting.* Proc. of the 15th ACM International Conference on Information and Knowledge Management (CIKM), 2006. pp. 4-13.
[34] Wen H. *et al.*, *Clustering web search results using semantic information* International Conference on Machine Learning and Cybernetics, 2009. 3:1504-1509.
[35] Wu P. *et al.*, *Towards keyword-driven analytical processing.* Proceedings of the Inter. Conf. on Management of Data (SIGMOD'07), 2007. pp. 617–628.