SemIndex+: a Semantic Indexing Scheme for Structured, Unstructured, and Partly Structured Data

Joe Tekli[*]
E.C.E. Department
Lebanese American Univ.
36, Byblos, Lebanon
joe.tekli@lau.edu.lb

Richard Chbeir
Univ. of Pau & Pays Adour
ES2/UPPA
64600 Anglet, France
richard.chbeir@univ-pau.fr

Agma J.M. Traina
ICMC Institute
University of Sao Paulo,
13566 Sao Carlos, Brazil
agma@icmc.usp.br

Caetano Traina Jr.
ICMC Institute
University of Sao Paulo,
13566 Sao Carlos, Brazil
caetano@icmc.usp.br

Abstract. While Information Retrieval (IR) systems have gained success in Web-style search engines in the past two decades, nonetheless, the DataBase (DB) paradigm remains prevalent in handling data in enterprise environments and digital libraries, and is gaining even more importance in the Semantic Web with the increasing need to handle partly structured (NoSQL) data. This paper describes *SemIndex+*, a semantic-aware indexing and querying framework that allows semantic search, result selection, and result ranking of structured (relational DB-style), unstructured (IR-style), and partly structured (NoSQL) data. Various weighting functions and a parallelized search algorithm have been developed for that purpose and are presented here. We provide a general keyword query model allowing the user to choose the results' semantic coverage and expressiveness based on her needs. Different from alternative solutions involving query relaxation, query refinement, or query disambiguation, our approach incorporates semantics at the most basic data indexing level: providing more opportunities toward speedups and semantic coverage. An extensive experimental evaluation, comparing *SemIndex+* with alternative methods, highlights our approach's flexibility and effectiveness, which in turn impact efficiency (requiring less or more time following the user specified index and query semantic coverages).

Keywords: Semantic Queries, Inverted index, NoSQL indexing, Semantic Network, Semantic-aware data processing, Textual databases, Query Relaxation, Semantic Disambiguation.

1. Introduction

Full-text search techniques originally developed in Information Retrieval (IR) systems, and more recently imbedded in DataBase (DB) systems, aim at providing the most relevant textual *data objects* (e.g., *documents* in IR, or *tuples* in a DB) to a user query consisting of a set of keywords [9, 10, 30]. The brute force approach is to sequentially scan the data objects in the collection, in order to search for terms matching those of the user query, which could be extremely inefficient for big data collections. Most IR/DB systems use some sort of indexing to speed up the search process. Currently, the foremost indexing structure used with full-text search is the *inverted index* model [56, 61, 79]. Inverted indexes associate each term in the text with a list of pointers to the data objects that contain the term, in the form of a list of (*term*, *objectIDs*[]).Then, when an enquiry is performed, the index is queried with every term within the user query, identifying as candidate results all data objects that contain the query terms in just one scan of the index [9]. The inverted index is widely adopted for full-text indexing of large textual collections [9], and is supported by many DBMSs[1] for storing and handling structured data (cf. Table 1) [4, 9], unstructured data (cf. Table 2) [30, 60], as well as partly structured[2] data (cf. Table 3) [40, 86]. Yet, as full-text search systems became available to non-expert users, keyword queries become noisier, where non-experts have poor or no knowledge about the data being searched. As a result, they tend to formulate query keywords which are often syntactically different from those used in indexing relevant documents in the DB [39], thus returning non-relevant results or completely missing relevant ones.

1.1. Motivation Examples

To illustrate this, consider a textual data collection Δ from a movie database, where each movie in Δ is identified with an *ID* and is described with some text, including the movie *title*, *year*, *plot*, *genre*, and *info*. The data collection can be presented in different forms: i) structured, following the traditional relational (SQL) DB model, noted Δ_{Struct} as shown

[*] Corresponding author. This study was partly conducted during the author's Fulbright Visiting Scholar research mission conducted in the Computer and Information Science Department, University of Michigan, Dearborn, USA

[1] Database Management Systems.

[2] We use the expression *partly structured*, to distinguish basic NoSQL data consisting of attribute-value items, from hierarchically structured data such as XML and RDF-based serializations, which are commonly referred to as *semi-structured* data [81]. The latter are out of the scope of this paper, and will be addressed in a dedicated study.

in Table 1, ii) unstructured, made of free text, noted Δ_{Free} as shown in Table 2, or iii) partly structured, following the basic attribute-value (NoSQL) DB model, noted Δ_{Part} as shown in Table 3.

Table 1. Sample *movies* data collection extracted from IMBD[1], provided in the form of structured text, noted Δ_{Struct}.

ID	title	year	plot	genre	info
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...*	*Horror, Thriller*	*When the Mandrakises were about to leave…*
O_2	*Days of Thunder*	1990	*Cole Trickle is a young racer from California with years of experience in open-wheel racing winning championships in Sprint car racing…*	*Action, Drama*	*In the clip from the race at Rockingham…*
O_3	*The Sound of Music*	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...*	*Drama, Family*	*The 1996 video fits the movie onto one VHS…*

Table 2. Sample data collection from IMBD (cf. Table 1), provided in the form of unstructured free text, noted Δ_{Free}.

ID	description
O_1	*When a Stranger Calls* (2006), Horror, Thriller: *A young high school student babysits for a very rich family. Driving by the house, a strange car...*
O_2	*Days of Thunder* (1990), Action, Drama: *Cole Trickle is a young racer from California with years of experience in open-wheel racing winning championships in Sprint car racing…*
O_3	*Sound of Music, The* (1965), Drama, Family: *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...*

Table 3. Sample data collection from IMBD (cf. Table 1), provided in the form of partly structured text, noted Δ_{Part}.

ID: O_1	**title:** *When a Stranger Calls*	**year:** *2006*	**plot:** *A young high school student babysits for a very rich family. Driving by the house, a strange car…*	**genre:** *Horror, Thriller*	**info:** *When the Mandrakises were about to leave…*

ID: O_2	**description:** *Days of Thunder* (1990), Action, Drama: *Cole Trickle is a young racer from California with years of experience in open-wheel racing winning championships in Sprint car racing…*

ID: O_3	**title:** *The Sound of Music*	**plot:** *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...*	**genre:** *Drama, Family*

Consider keyword-based queries q_1, q_2, and q_3 applied respectively on each the above data collections, formulated as standard DB *selection* (σ) *containment* ($\in$) queries (formally described in Section 5): $q_1 = \sigma_{title \in (\text{"sound"}, \text{"of"}, \text{"music"})}\Delta_{\text{Struct}}$, $q_2 = \sigma_{description \in (\text{"open-wheel"}, \text{"racer"})}\Delta_{\text{Free}}$, and $q_3 = \sigma_{genre \in (\text{"thriller"})}\Delta_{\text{Part}}$. The search result for query q_1 is movie O_3 (from Δ_{Struct}) which *title* attribute contains occurrences of each of the query's terms. Similarly, search results for q_2 and q_3 are: O_2 (from Δ_{Free}) and O_1 (from Δ_{Part}) respectively. Yet, if the user wants to search for a particular movie but cannot recall its exact title, plot, or genre descriptions, she will likely use her own terminology in choosing query terms which (we naturally assume) are lexically and/or semantically similar to the movie's description terms, e.g., *"voice of melody"*, *"auto rallying"*, or "*suspense*". Such terms might not exactly match those used to describe (and index) the movie objects (which is the case in our examples), and thus will miss movies O_1, O_2, and O_3 as relevant results. As a matter of fact, there are typically many ways to specify a given concept: as textual descriptions may involve terms with multiple meanings (*homonymy*, e.g., term "*sound*" could mean a *particular auditory effect* or a *narrow channel in the sea paper sheet* according to a general purpose knowledge base such as WordNet [66]), terms implied by other terms (*metonymy*, e.g., term "*thunder*" implies "*lightning*", and *"nun"* implies *"religion"*), several terms having the same meaning (*synonymy*, e.g., terms "*thunder*", "*roar*", and "*boom*" all refer to the same meaning: *a deep prolonged loud noise*), or terms related by some semantic relationships (e.g., *hypernymy* (*isA*), *holonymy* (*partOf*), such as *thunder-isA-noise*, or *engine-partOf-car*). Therefore, the terms used as keywords in a user's query might literally match terms in irrelevant movie objects (decreasing search *precision*[2]), or might completely miss relevant movies where no exact query term matches are identified (decreasing search *recall*[1]). In addition, the movie

[1] Internet Movie DataBase (http://www.imdb.com/).

[2] *Precision* highlights the percentage of correctly selected results, whereas *recall* designates the percentage of wrongly missed results (i.e., results that should have been selected, cf. Section 7.6).

objects might not be extensively described or well-tagged in the DB, or might not be described using the same attributes: namely in a partly structured DB (cf. Table 3).

This shows that the standard inverted index cannot deal with these cases. Solving this issue has been the main motivation for developing so-called *semantic-aware* or *knowledge-aware* (keyword) query systems, which have emerged since the past decade as a natural extension of traditional containment queries, encouraged by (non-expert) user demands. Most existing works in this area (cf. Background in Section 2) have incorporated semantic knowledge at the *query processing* level, to: i) pre-process queries using query relaxation and rewriting [17, 27, 64], ii) disambiguate queries using semantic disambiguation and entity recognition techniques [17, 58, 71], and/or iii) post-process query results using query refinement and results re-ranking [71, 80, 88]. Yet, various challenges remain unsolved, namely: i) time latencies when involving query pre-processing and post-processing [27, 64], ii) reduced quality of query relaxation/rewriting and query disambiguation results: requiring context information (e.g., expanded queries, user profiles, or query logs) which are not always available [19, 32], and iii) limited user involvement, where the user is usually constrained to providing feedback and/or performing query refinement only after the first round of results has been provided by the system [21, 67].

1.2. Proposal Overview

In this work, we adopt another alternative: building a semantic-aware inverted index called *SemIndex+*, integrating textual information with domain knowledge (not only at the querying level, but rather) at the most basic *data indexing* level, in order to support semantic-aware querying and answer most challenges identified above. Fig. 1 depicts the overall framework of our approach and its main components. Briefly, *SemIndex+* maps two data resources, namely a *textual data collection* (represented as an inverted index), and *a semantic knowledge base* (represented as a semantic network) into a single and tightly coupled semantic-aware data structure. This is performed offline – prior to online query execution, providing more opportunities toward both speed-ups and semantic-based filtering, thus minimizing the need for query pre- and post-processing.

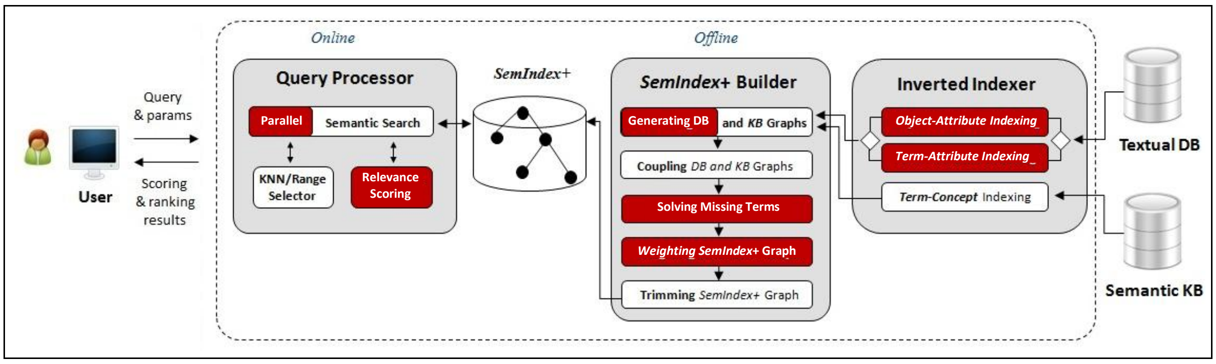


Fig. 1. Overall architecture of the *SemIndex+* framework (added components are highlighted in red).

An initial solution design titled *SemIndex* was given in [23] aimed at indexing unstructured (free-text) data only. This paper introduces *SemIndex+*, a new framework allowing to index and search unstructured, structured (relational), and partly structured (NoSQL) textual data. *SemIndex+*'s main additions to the previous framework are highlighted in Fig. 1. At the indexer level, we add: i) an extension of *SemIndex*'s logical design to handle varying multi-attribute datasets (using attribute sensitive indexers), ii) a dedicated algorithm to handle terms with missing semantic connections (which we designate as *missing terms*), and iii) a mathematical model for weighting *SemIndex+* entries (i.e., the graph's nodes and edges). At the query processing level, we develop: iii) a parallelized (multithreaded) query processing algorithm, coupled with iv) a dedicated relevance scoring measure, required in the query evaluation process to retrieve and rank relevant query answers. In addition, we add: iv) a detailed complexity analysis covering the new index construction and querying algorithms, and v) an extensive experimental study comparing *SemIndex+*'s effectiveness and efficiency with various generic approaches (including *inverted index search*, *query relaxation*, *query disambiguation*, and *query refinement*). Results highlight *SemIndex+*'s flexibility (involving the user in the whole process: during initial index building, query writing, and then query refinement) and effectiveness (producing significantly more *semantically relevant* results compared with existing solutions) which in turn affect its efficiency (requiring less or more processing time following user-specified index and query semantic coverages).

The rest of this paper is organized as follows. Section 2 briefly reviews related works. Section 3 describes *SemIndex+*'s input resources. Section 4 develops *SemIndex+*'s design, including the index construction process and weighting functions. Section 5 develops *SemIndex+*'s query model and query processing algorithm. The computational complexity of *SemIndex+* construction and querying algorithms is provided in Section 6. Experimental results are presented in Section 7. Section 8 concludes the paper with ongoing and future directions.

2. Related Works

2.1. Keyword Search in Textual Databases

Early approaches on keyword search queries for RDBs use traditional IR scores (e.g., TF-IDF) to find ways to join tuples from different tables in order to answer a given keyword query [4, 35, 47]. 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. Recent methods on RDB full-text search in [14, 60] 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 [63] tackle the issue of keyword search on streams of relational data, whereas the approach in [89] 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 [57, 73], or extracting structured information (e.g., identifying entities) from text (e.g., Web documents) and storing it in a DBMS to simplify querying [25, 26]. Few recent approaches have addressed keyword-search in NoSQL DBs such as HBase [40, 49, 86], using dedicated (row or column) table schemas coupled with (horizontal or vertical) index partitioning, to support parallel index storage and search [40], as well as multi-term indexing and incremental query result fetching [86]. Keyword-based search for other data models, such as XML [2, 24] and RDF [13, 15] have also been studied.

Discussion: 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.

2.2. Extending *Syntactic* Search toward *Semantic* Search

While DB approaches focused on integrating traditional (syntactic) keyword-based search functionality, many efforts have been deployed by the IR community to extend syntactic processing toward semantic full-text search using dedicated semantic indexing techniques, leading to the so-called *concept-based* (or *knowledge-based*) IR [8, 11, 55]. The latter is an alternative IR approach that aims to tackle the semantic relatedness problem by transforming both documents and queries into semantic representations, using semantic concepts in a reference knowledge base (instead of syntactic keywords/terms) such that the retrieval process is undertaken in the concept space [12, 42, 55]. Existing concept-based methods, e.g., [8, 11, 12, 42, 54, 55], can be characterized by three parameters: i) *Semantic indexing*: consists of the representation model the concepts are based on, as well as the underlying indexing technique used to access the concepts. It attempts to solve the problems of lexical matching by using conceptual indices instead of individual word indices for retrieval [54]; ii) *Mapping method*: the mechanism that maps the lexical terms with these semantic concepts. The mapping can be performed using manual mapping w.r.t. a handcrafted ontology such as WordNet [66] or Yago [45], or using machine learning [41] or graph matching techniques [12], though this would usually imply less accurate mappings, iii) *Usage in the retrieval process*: the stages in which the concepts are used in information retrieval. Concepts would be best used throughout the entire process, in both the indexing and retrieval stages [12]. Yet, most existing solutions apply concept analysis in one stage only, performing so-called *query relaxation/refinement* or *query disambiguation* over the *bag of words* retrieval model [44], to reduce processing time.

2.2.1. Query Relaxation and Refinement

Traditional query relaxation and refinement methods (cf. surveys in [19, 77]), rely on *corpus-based* evidence: expanding user queries by adding words that often co-occur with the query terms in a given corpora (e.g., *France* and *Paris*; *car* and *driver*). Query expansion terms can also be identified from user feedback (frequent terms occurring in previous results) [20, 68, 76], as well as query logs (terms related to past queries and accessed documents) [29, 43, 51]. We distinguish between query relaxation and query refinement on the grounds that query relaxation occurs before query execution and is considered as a query pre-processing phase [6, 17, 68], whereas query refinement occurs after the first round of query results have been acquired by the user and is thus considered as a query post-processing phase [20, 21, 67]. Such approaches usually require manual tuning to improve performance: too few expansion terms may have no impact, and too many can cause a *query drift* [68]. In addition, corpus-based approaches require extensive training and huge corpora, which make such methods less practical especially in the context of Web applications. This has led to a growing interest *knowledge-based* solutions, e.g., [17, 58, 72], investigating the use of ontological information (rather than corpus statistics) to assist the user in formulating and/or expanding keyword queries by: i) allowing some user interaction to accurately identify the intended senses of query-terms, and then ii) expanding/rewriting query keywords via their most related semantic concepts in the reference semantic source [7] (such as WordNet [66] or Yago [45]). Note that query relaxation and query refinement techniques introduce additional query pre-processing and post-processing overhead respectively compared with the traditional *bag-of-words* approach, adding statistical and/or knowledge-based information to expand/rewrite each query before/after processing, such that users are generally involved in the query refinement process after the system provides the first round of results.

2.2.2. Query Disambiguation

An alternative approach to handle semantic meaning is to apply automatic *word sense disambiguation* (WSD) to queries, during query execution time. Disambiguation methods usually use knowledge resources such as WordNet [59], and/or co-occurrence statistical data in a corpus [78] to find the possible senses of a word and map word occurrences to the correct sense (cf. WSD surveys in [69, 81]). Semantic query analysis in IR usually involves two steps: i) WSD to identify the user's intended meaning for query terms, and ii) semantic query representation/enhancement in order to alter the query so that it achieves better (precision and recall) results [7, 58]. The disambiguated query terms are then used in query processing, so that only documents that match the correct sense are retrieved [59]. Yet, the performance of WSD-based approaches depends on the performance of the automated WSD process [37] which generally: i) is computationally complex requiring substantial execution time [69], ii) depends on the context of the query/data processed (e.g., surrounding terms) [22, 90] which is not always sufficiently available (e.g., keyword queries on the Web are typically 2-to-3 words long [53, 86]), and thus iii) do not guarantee correct results [37, 52] as incorrect disambiguation is likely to harm performance [37].

Discussion: Most existing methods focus on query pre-processing using query relaxation [17, 64, 68], query disambiguation [7, 58, 59], or query post-processing using query refinement (exploiting user feedback after the system provides the first round of results) [20, 21, 67]. In contrast, our study encloses the semantic knowledge directly into an inverted index so that semantic-aware processing can be done at the most basic indexing level, where more opportunities can be explored toward semantic-aware filtering. This allows users to be involved in the whole process: during data indexing, initial query writing, processing, and then performing query refinement and rewriting.

2.3. Inverted Indexes handling Data Semantics

Various efforts have been recently deployed to extend the inverted index toward handling data semantics. These can be organized in three main categories: 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 [54] suggest extending the traditional (*term*, *docIDs*[]) inverted index toward a (*term*, *context*, *docIDs*[]) structure where contexts designates senses (synsets) extracted from WordNet, associated to each term in the index taking into account the statistical occurrences of concepts in Web document [11]. The authors however 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 is introduced in [92], extending the inverted index structure by adding extra pointers linking each entry of the index to semantically related terms, (*term*, *docIDs*[], *relatedTerms*[]). Term links are identified by analyzing term occurrences in Web documents, based on Web document Page-Rank linkage analysis. Yet, the authors do not describe how they consider semantic relatedness between terms (what kinds of semantic relations and processing are used), nor how the index is actually built based on linked Web documents. A second approach to semantic indexing is to add words as entities in the ontology [11, 85]. 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 right hand sides of this *occurs-in-context* index, which would be more time consuming [11] than reading an indexed entry such as with the inverted index. A possible optimization would be to split the relation into *word occurs-in context* and *concept occurs-in context*, yet the relations would remain huge and *concept occurs-in-context* always has to be processed entirely [11]. A related approach has been used to disambiguate WordNet glosses [70, 85], and has been proven useful in enhancing WSD-based query expansion. 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 [11] 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). They produce a 4-tuples index structure (*prefix*, *terms*[]) ↔ (*term*, *context*, *concepts[]*) where a prefix contains one index item per occurrence of a term starting with that prefix, adding an entry item for each occurrence of an ontological concept in the same context as one of these words. The authors tailor their method toward incremental query construction, performing context-sensitive prefix suggestions of terms in building queries.

Discussion: The method in [11] seems arguably the most related to our study, with major differences in objectives and theoretical/technical contributions: the authors in [11] target semantic full-text search with special emphasis on incremental query construction and suggestion based on query term prefixes and result excerpts, whereas we target semantic search in textual DBs extending DB-style (SQL and NoSQL based) querying capability toward semantic full-text search. Hence, while the authors in [11] focus on the IR aspects of indexing, keyword query construction, and query evaluation, we rather present a full-fledged textual DB solution, with structures and algorithms designed for seamless storage and manipulation within a typical DB system, allowing to process unstructured (IR-style), structured (SQL-style), and partly structured (NoSQL) data.

3. Input Resources

As indicated previously, our work consists in combining two resources, a *textual data collection* and a *semantic knowledge base*, in order to build *SemIndex+*. We first describe the textual resource in Section 3.1, and then describe the semantic resource in Section 3.2.

3.1. Textual Data Collection

In our study, a textual data collection can be a set of: i) documents or unstructured text fields in a textual DB, ii) structured tuples in a relational DB, or iii) partly structured data items in a NoSQL DB, as shown in Tables 1 to 3 respectively. Here, we provide a unified formal definition:

Definition 1 - Textual Data Collection: A textual data collection Δ (i.e., *textual collection* for short) is defined as a collection of data objects where every object $O_i \in \Delta$ has a unique identifier $id(O_i)$ and is made of a set of attribute-value pairs $O_i\{A_m{:}a_m,\dots, A_j{:}a_j,\dots, A_n{:}a_n\}$. Each attribute A_j has domain $dom(A_j)$ designating the set of values (strings, numbers, etc.) allowed in A_j, where $a_j \in dom(A_j)$. Each data value a_j from O_i associated to attribute A_j is denoted as $O_i.a_j$. We designate by $\Delta.A = \{A_1, A_m,\dots, A_n,\dots, A_p\}$ the set of all attributes associated to all objects in Δ ●

Definition 1 ‑ can be used to describe: structured, unstructured, and partly structured (NoSQL) data. It can describe a structured (relational) data collection, such as Δ_{Struct} in Table 1, which consists of 3 data objects with textual contents organized following a set of 6 attributes: $\Delta_{\text{Struct}}.A$ = {*ID*, *title*, *year*, *plot*, *genre*, *info*}. Similarly, an unstructured (free text) data collection, such as Δ_{Free} in Table 2, would be represented as a set of data objects having each an object *ID* coupled with its textual description, e.g., $\Delta_{\text{Free}}.A$ ={*ID*, *description*}. Definition 1 ‑ also allows to represent partly structured data collections, such as Δ_{Part} in Table 3, where every data object in the data collection is made of the object *ID* combined with a different subset of attributes defined in the data collection, e.g., a subset of $\Delta_{\text{Part}}.A$={*ID*, *title*, *year*, *plot*, *genre*, *info*, *description*}. Here, we adhere to the most basic form of partly structured data collections known as NoSQL attribute-value or key-value stores [33], where every data object (identified by its key) is made of a set of *attribute-value* items (e.g., *title*-"*When a Stranger Calls*" is the first item in data object O_1 of Δ_{NoSQL}). More sophisticated NoSQL models such as document DBs [13] or graph stores [74], where attribute-value items can be linked/nested with hierarchical or cross relations, are disregarded here and will be covered in a dedicated study.

Given a textual data collection Δ, an inverted index (also referred to as a posting file, or inverted list) built upon Δ, is (in its most basic form) a sorted list of index terms associated each with a set of object identifiers from Δ [61], disregarding attribute information as shown in Fig. 2.a. In this study, we extend the basic inverted index to handle multi-attribute data objects, introducing an *object-attribute* (OA) index:

Definition 2 - Object-Attribute (OA) Inverted Index: Given a textual data collection Δ, an OA inverted index built on Δ, denoted as $InvIndex_{OA}(\Delta)$, is a structure of the form ($dom(A)$, OAs, f) where:

- $dom(A)$ designates the set of values within the domains of all attributes $\forall A_j \in \Delta.A$. Considering text-only domains, values come down to textual tokens, i.e., *terms* (words/expressions),
- OAs designates the set of object (identifier)-attribute doublets, i.e., $OAs = \{(id(O_i), A_j)\}$ $\forall O_i \in \Delta$ and $\forall A_j \in \Delta.A \,/\, \exists\, O_i.a_j \neq \varnothing$, where A_j is an attribute for which object O_i has a non-null value,
- f is a function mapping each $term \in dom(A)$ with a list of object-attribute doublets $OAs[]$ designating the term's occurrence locations in Δ, i.e., $OAs[] = \langle (id(O_i), A_j)\rangle$[1] $/\ term \in O_i.a_j$

A *term* used as textual token in the inverted index is referred to as *index term*, whereas the list of object-attribute doublets, i.e., *OAs*[], mapping to each index term is referred to as the term's *posting list* (cf. example in Fig. 2.b) ●

Fig. 2 shows extracts of a basic inverted index (without attribute information, which was adopted in the initial *SemIndex* study [23]) as well as the corresponding *OA* multi-attribute inverted index, built on the sample movie data collection Δ_{Part} in Table 3, where data objects O_1, O_2, and O_3 have been indexed using index terms extracted from the data collection, and sorted in alphabetic order. Another form of a multi-attribute inverted index can also be defined as a *term-attribute* (TA) index, where term-attribute doublets are mapped with object identifies (as shown in Fig. 2.c). Both (logically equivalent yet technically different) variants can be straightforwardly integrated in *SemIndex+*. Note

[1] We use symbols ⟨ and ⟩ to designate an *ordered* list of elements, and symbols { and } to designate an *unordered* set.

that we will use Δ_{Part} as the running example data collection in the remainder of the paper since it integrates and generalizes all interesting properties from its Δ_{Struct} and Δ_{Free} counterparts.

SemIndex+ allows both *attribute-sensitive* and *attribute-free* indexing and querying (as opposed to only *attribute-free* processing in [23]). We can choose to disregard attributes: i) at the index level (using the traditional *InvIndex*, adopted in [23]), or ii) at the querying level, following the user query at hand (since $InvIndex_{OA}$ and $InvIndex_{TA}$ can be straightforwardly processed as traditional attribute-free indexes by simply disregarding attribute information and merging corresponding posting list entries). For instance, running query $q_4 = \sigma_{\perp \in (\text{"}light\text{", "}horror\text{"})} \Delta_{\text{Part}}$ on either $InvIndex_{OA}$ (Δ_{Part}) or $InvIndex_{TA}$ (Δ_{Part}), where $\perp$ designates the combined textual content from all attributes, would return as answer O_1, since O_1's textual content in Δ_{Part} contains both terms "*light*" and "*horror*" (despite the fact that the two terms occur in two separate attributes: *plot* and *genre* respectively).

Term	Object IDs[]
"Car"	⟨O1, O2⟩
"Horror"	⟨O1⟩
"Light"	⟨O1⟩
"Sound"	⟨O3⟩
"Zen"	⟨O1⟩
…	…

a. Simple (attribute free) inverted index *InvIndex*(Δ_{Part}), adopted in our original study in [23]

Term	(Object id, Att.)[]
"Car"	⟨(O1, *plot*), (O2, *description*⟩
"Horror"	(O1, *genre*),
"Light"	⟨(O1, *plot*)⟩
"Sound"	⟨(O1, *plot*), (O3, *title*), (O3, *plot*)⟩
"Zen"	⟨(O1, *plot*)⟩
…	…

b. Object-Attribute inverted index $InvIndex_{OA}$(Δ_{Part})

(Term, Attribute)	Object IDs[]
("Car, *plot*)	⟨O1⟩
("Car, *description*)	⟨O2⟩
("Horror, *genre*)	⟨O1⟩
("Light", *plot*)	⟨O1⟩
("Sound", *title*)	⟨O3⟩
("Sound", *plot*)	⟨O1, O3⟩
("Zen", *plot*)	⟨O1⟩
…	…

c. Term-Attribute inverted index $InvIndex_{TA}$(Δ_{Part})

Fig. 2. Sample inverted indexes based on text collection Δ_{Part} in Table 3.

3.2. Semantic Knowledge Base

In the Natural Language Processing (NLP) and Information Retrieval (IR) fields, semantic knowledge bases (i.e., thesauri, taxonomies, and/or Ontologies such as WordNet [66], Roget's thesaurus [90], and Yago [45]) provide a framework for organizing words/expressions into a semantic space [16]. A knowledge base[1] is usually modeled as a semantic network made of a set of entities representing semantic concepts (or groups of words/expressions), and a set of links between the entities, representing semantic relationships (*synonymy*, *hyponymy*, etc.). In this study, we adopt a graph-based structure to model semantic knowledge bases. In such a structure, entities are represented as vertices, and the semantic relationships between entities are represented as directed edges. Formally:

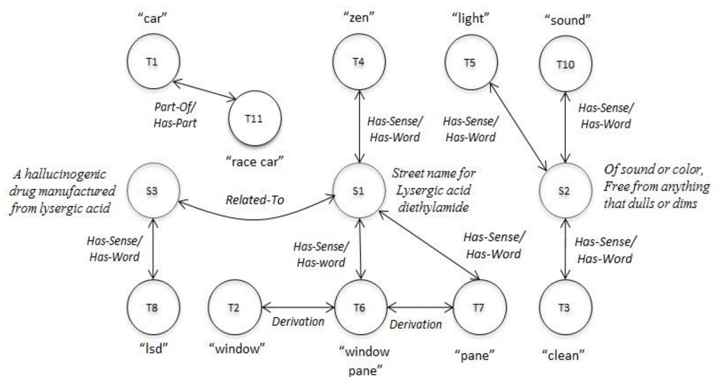
Definition 3 - Semantic knowledge base: A semantic knowledge base *KB* (i.e., *knowledge base* for short) is represented as a semantic network graph, also known as knowledge graph, $G_{KB}(V, E, L, f_V, f_E)$ where:

- *V* is a set of vertices (nodes), designating entities in the knowledge base. To illustrate this with WordNet for example, *V* includes both: i) *sense* nodes, representing semantic senses (*synsets*) with glosses, and ii) *term* nodes, representing literal words/expressions.
- *E* is a set of directed edges, an edge consisting of an ordered pair of vertices in *V*.
- *L* is a set of edge labels denoting semantic/lexical relationships. For WordNet, *L* includes:
  - Semantic relationships between concepts, e.g., *hyponymy*, *hypernymy*, *meronymy*, etc.
  - Semantic relationships between concepts and terms, namely *has-sense* and *has-term* (e.g., in Fig. 3, word *"Zen" has-sense* S_1, and S_1 *has-term "Zen"*)
  - Lexical relationships between terms, namely *derivation* (e.g., term *"Zen" derives* term "*Buddhist Zen*", and "*Buddhist Zen*" *is-derived-from* "*Zen*")
- f_V is a function defined on *V*, designating the string value of each node in *V*. For WordNet, string values include: i) glosses/definitions, when dealing with *sense* nodes, and ii) and literal words/expressions,
- f_E is a function defined on *E*, assigning a label from *L* to each edge in *E*. Multiple edges may exist between the same pair of vertices when dealing with *term* nodes, which makes G_{KB} a multi-graph •

An extract from the WordNet lexical ontology is shown in Fig. 3, where S_1, S_2 and S_3 represent senses (i.e., synsets), 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 along aside the nodes. Given that most semantic/lexical relationships are symmetrical (*hyponymy*/*hypernymy*, *meronymy*/*holonymy*, *has-sense/has-term*, etc.), and given that a relationship cannot exist without its symmetrical counterpart, we simplify our graph model by representing each couple of symmetrical relationships between senses and/or terms with one edge having opposite directions (instead of two edges), labeled with the names of the symmetrical relationships.

[1] In the remainder of the paper, we will use WordNet [66] as the illustrative semantic knowledge base (cf. Fig. 3).

A simple inverted index *InvIndex*(G_{KB}) can be subsequently built for the textual tokens of each G_{KB} entity (i.e., string values of *term* nodes and *sense* nodes, cf. Fig. 3.b) to speed up term/sense lookup when creating and then querying the integrated *SemIndex+* structure.



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⟩
"window pane"	⟨S1⟩
…	…

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 the corresponding inverted index.

4. *SemIndex+* Design

In this section, we present the logical design techniques of *SemIndex+*, highlighting extensions to our initial model from [23] necessary to handle structured and partly structured data. In the following, we first present *SemIndex+*'s graph model, and then describe its construction process.

4.1. *SemIndex+* Graph Model

We define *SemIndex+* as an extension of the *SemIndex* knowledge graph developed in [23], where additions to the initial model necessary to handle multi-attribute indexes are highlighted in ***bold***:

Definition 4 - *SemIndex+* graph: Given an input textual collection Δ and an input knowledge base *KB*, we define *SemIndex+*(Δ, *KB*) as an extended knowledge graph $\tilde{G}_{SI}$ (V_i, V_d, $\boldsymbol{V_a}$, L, E_i, $\boldsymbol{E_d}$, $\boldsymbol{f_V}$, f_E, $\boldsymbol{f_W}$) where:

- V_i is a set of *index nodes*, denoting i) entities (*senses* and *terms*) from *KB*, and ii) *index terms* from Δ:
  - $V_i^+ \subseteq V_i$ the subset of *term* nodes designating *searchable terms*[1] in $\tilde{G}_{SI}$, i.e., nodes referring to *terms* from *KB* and *index terms* from Δ (represented visually as *circle* nodes ○)
  - $V_i^\# \subseteq V_i$ the subset of *sense* nodes in $\tilde{G}_{SI}$ referring to *senses* from *KB* (represented as *double circles* ◎)

Naturally, $V_i = V_i^+ \cup V_i^\#$

- V_d is a set of *data nodes*, denoting data objects from Δ (represented visually as *square* shaped nodes □[2])
- **V_a is a set of *attribute nodes*, denoting attributes from Δ (represented as *polygon* shaped nodes ⬡)**
- E_i is the set of edges between index nodes, called *index edges*, defined as ordered pairs of index nodes in V_i (represented visually as straight arrows →)
- **E_d is the set of 3-uniform hyper-edges linking index nodes with data nodes through attribute nodes, called *data edges* (represented visually as dashed arrows ⇢)**
- L is a set of edge labels including:
  - *Index edge* (E_i) labels which represent semantic/lexical relationships between index nodes (e.g., *hyponymy*, *meronymy*, *has-sense*, etc.)
  - A single *data edge* (E_d) label: *contained*, designating the containment relationship between term nodes in V_i^+ and data nodes in V_d
- **f_V is a function defined on $V_i \cup V_d \cup V_a$, representing the string value of each node in $V_i \cup V_d \cup V_a$**
- f_E is a function defined on $E_i \cup E_d$, assigning a label from L to each edge in $E_i \cup E_d$
- **f_W is a weighting scheme defined on the nodes in $V_i \cup V_d \cup V_a$ and the edges in $E_i \cup E_d$. The weights will be used in selecting and ranking semantic-aware query results** (cf. Section 4.5) •

[1] Searchable terms will be mapped against query terms when performing query processing (cf. Section 5).
[2] Data nodes will designate (potential) query search results (Section 5.2).

Building the *SemIndex+* graph comes down to: i) generating two separate graph representations for each of the input resources: the textual collection (noted $\widetilde{G}_{\Delta}$) and the knowledge base (noted $\widetilde{G}_{KB}$), following our *SemIndex+* graph model, and then ii) tightly coupling the resulting graphs into a single *SemIndex+* graph structure (noted $\widetilde{G}_{SI}$), which we describe in the following subsections. A sample *SemIndex+* graph is shown in Fig. 7 (Section 4.4), built based on textual collection Δ_{Part} from Table 3 (where corresponding $\widetilde{G}_{\Delta}$ in provided in Fig. 5.a) and the *KB* extract in Fig (where corresponding $\widetilde{G}_{KB}$ is reported in Fig. 5.b). It comprises 3 data nodes ($O_1 – O_3$), 4 attribute nodes ($A_1 – A_4$), 3 index *sense* nodes ($S_1 – S_3$), and 12 index *term* nodes ($T_1 – T_{12}$) along with corresponding data and index edges.

4.2. Indexing the Textual Collection

Given an input textual collection Δ, we use a conversion function following Definition 4 -to produce a *SemIndex+* graph representation of Δ denoted as $\widetilde{G}_{\Delta}$ = *SemIndex+*(Δ, ∅)[1]. It comes down to first generating Δ's inverted index: either using $InvIndex_{OA}(\Delta)$ (cf. Fig. 2.b) or $InvIndex_{TA}(\Delta)$ (cf. Fig. 2.c), which will be represented as the same *SemIndex+* graph $\widetilde{G}_{\Delta}$ (cf. Fig. 5.a). $\widetilde{G}_{\Delta}$ consists of: i) a set of index nodes V_i representing *index terms* in Δ (*searchable term* nodes), i.e., $V_i = V_i^{+}$ (since Δ does not contain *senses*, i.e., $V_i^{\#} = \phi$), ii) a set of data nodes V_d representing data objects in Δ, iii) a set of attribute nodes V_a representing attributes from Δ.*A*, and iv) a set of edge labels *L* including one single label: *contained*, underlining containment relationships represented as 3-uniform hyper-edges linking index nodes in V_i with data nodes in V_d through attribute nodes V_a.

Definition 5 - 3-Uniform Data Edge in *SemIndex+*: Given a *SemIndex+* graph $\widetilde{G}_{SI}$, we define $e^{d}_{\overset{a}{i}} \in \widetilde{G}_{SI}.E_d$ as an ordered 3-uniform hyper-edge connecting data node $n_d \in \widetilde{G}_{SI}.V_d$ with searchable term node $n_i \in \widetilde{G}_{SI}.V_i^{+}$ through attribute node $n_a \in \widetilde{G}_{SI}.V_a$ (cf. Fig. 4) •

A 3-*uniform hyper-edge* is a generalized definition of an edge connecting 3 nodes, forming a so-called *hyper-graph*[2] [36]. We adopt the 3-uniform hyper-edge relationship model to handle attribute nodes in our index, compared with the (attribute-free) 2-uniform *data edge* relationship model used in the original *SemIndex* [23] (cf. Fig. 4).

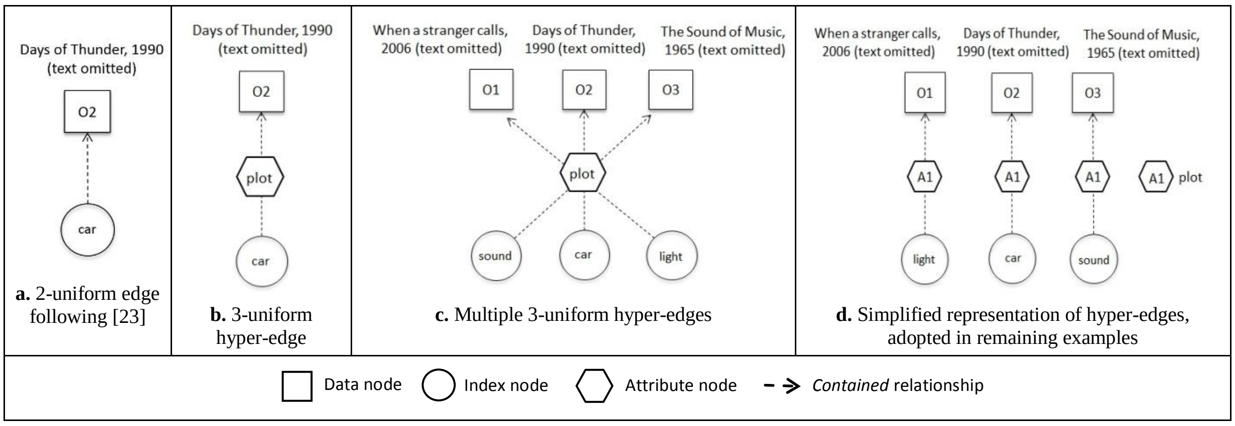

Days of Thunder, 1990 (text omitted)
O2
car
a. 2-uniform edge following [23]
Days of Thunder, 1990 (text omitted)
O2
plot
car
b. 3-uniform hyper-edge
When a stranger calls, 2006 (text omitted)
Days of Thunder, 1990 (text omitted)
The Sound of Music, 1965 (text omitted)
O1
O2
O3
plot
sound
car
light
c. Multiple 3-uniform hyper-edges
When a stranger calls, 2006 (text omitted)
Days of Thunder, 1990 (text omitted)
The Sound of Music, 1965 (text omitted)
O1
O2
O3
A1
A1
A1
A1 plot
light
car
sound
d. Simplified representation of hyper-edges, adopted in remaining examples
Data node
Index node
Attribute node
Contained relationship

Fig. 4. Sample 3-uniform data edges following *SemIndex+* extended model.

A sample $\widetilde{G}_{\Delta}$ 3-uniform hyper-graph representing our running example inverted indexes (cf. Fig. 2.b and c), based on textual collection Δ_{Part} in Table 3, is shown in Fig. 5.a.

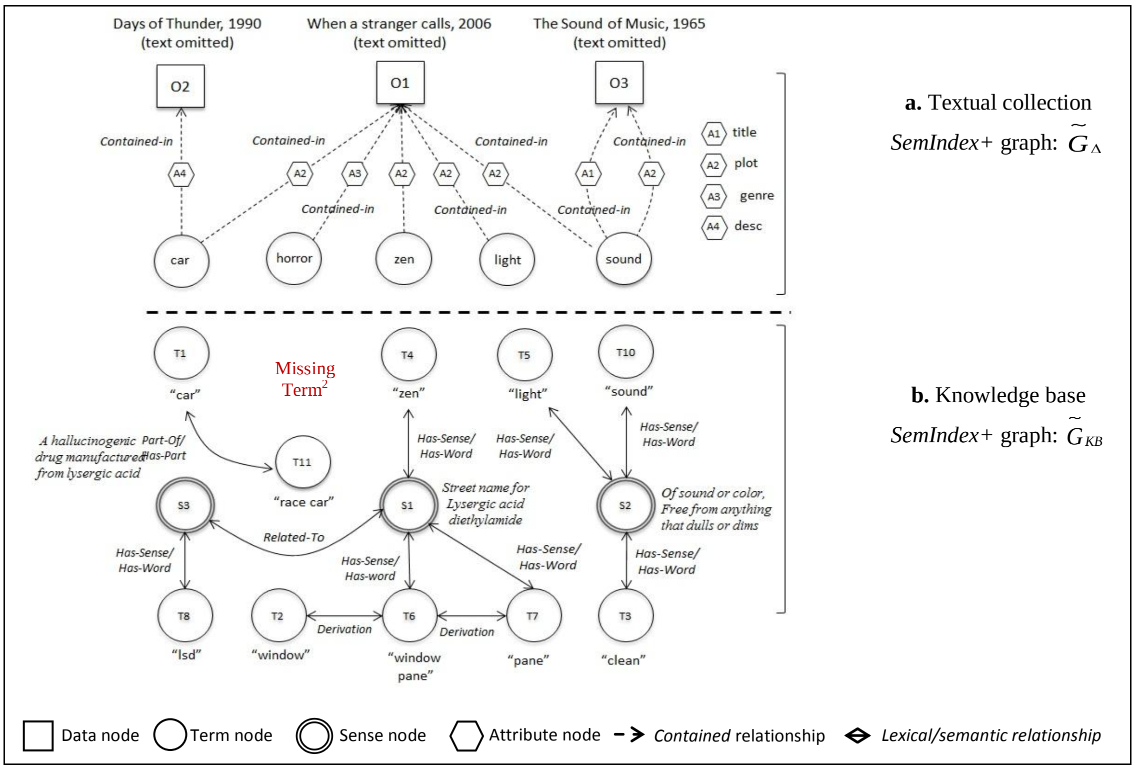
4.3. Indexing the Knowledge Base

Similarly, given a semantic knowledge base *KB*, represented as a knowledge graph G_{KB}, we use a conversion function following Definition 4 - to produce a *SemIndex+* graph representation of *KB* denoted as $\widetilde{G}_{KB}$ = *SemIndex+*(∅, *KB*)[1].

[1] Creating the textual collection index graph does not involve an input knowledge base, thus *KB* = ∅ (following Definition 5).

[2] A hyper-graph is a generalization of a graph *G*(*V*, *E*) where *V* is a set of nodes and *E* is set of edges underlining non-empty subsets of nodes from *V*. A hyper-graph is said to be *r*-uniform if all edges have cardinality *r*, i.e., if each edge connects *r* nodes together (e.g., a traditional graph comes down to a 2-uniform hyper-graph). An *r*-uniform hyper-graph is ordered if the occurrence of nodes in every edge is ordered from 1 to *r* [36].

G_{KB}'s inverted index $InvIndex(G_{KB})$ is generated and then represented as a *SemIndex+* graph $\tilde{G}_{KB}$ which inherits the properties of G_{KB}, such that: i) the set of index nodes V_i represents all nodes in G_{KB}, including *term* nodes (V_i^+) and *sense* nodes ($V_i^{\#}$), ii) the sets of data nodes V_d and attribute nodes N_a are empty (since *KB* does not contain data objects), and iii) the set of edge labels L includes all *index edge* labels designating semantic/lexical relationships in G_{KB} (e.g., *hyponymy*, *meronymy*, *has-sense*, *derivation*, etc.). A sample $\tilde{G}_{KB}$ graph representing our running example knowledge base is shown in Fig. 5.b.



Days of Thunder, 1990 (text omitted)
When a stranger calls, 2006 (text omitted)
The Sound of Music, 1965 (text omitted)
O2
O1
O3
Contained-in
A1 title
A2 plot
A3 genre
A4 desc
car
horror
zen
light
sound
a. Textual collection
SemIndex+ graph: $\tilde{G}_{\Delta}$
T1 "car"
Missing Term[2]
T4 "zen"
T5 "light"
T10 "sound"
T11 "race car"
A hallucinogenic drug manufactured from lysergic acid
Part-Of/ Has-Part
Has-Sense/ Has-Word
S3
S1
S2
Street name for Lysergic acid diethylamide
Of sound or color, Free from anything that dulls or dims
Related-To
Derivation
T8 "lsd"
T2 "window"
T6 "window pane"
T7 "pane"
T3 "clean"
b. Knowledge base
SemIndex+ graph: $\tilde{G}_{KB}$
Data node
Term node
Sense node
Attribute node
Contained relationship
Lexical/semantic relationship

Fig. 5. *SemIndex+* graph representations of input resources.

4.4. Coupling Resources to Build *SemIndex+*

Producing the combined *SemIndex+* graph structure $\tilde{G}_{SI}$ comes down to coupling both $\tilde{G}_{\Delta}$ and $\tilde{G}_{KB}$, noted as: $\tilde{G}_{SI} = \tilde{G}_{\Delta} \oplus \tilde{G}_{KB}$, where: i) the set of index nodes $\tilde{G}_{SI}.V_i = \tilde{G}_{\Delta}.V_i \cup \tilde{G}_{KB}.V_i$, including corresponding index edges from $\tilde{G}_{KB}$ such that $\tilde{G}_{SI}.E_i \cong \tilde{G}_{KB}.E_i$[3], ii) the set of data nodes $\tilde{G}_{SI}.V_d = \tilde{G}_{\Delta}.V_d$ and the set of attribute nodes $\tilde{G}_{SI}.V_a = \tilde{G}_{\Delta}.V_a$, including corresponding data edges from $\tilde{G}_{\Delta}$ such that $\tilde{G}_{SI}.E_d = \tilde{G}_{\Delta}.E_d$, and iii) the set of edge labels $\tilde{G}_{SI}.L = \tilde{G}_{\Delta}.L \cup \tilde{G}_{KB}.L$, including all index node semantic/lexical relationships as well as the *contained* data edge label. The pseudo-code of algorithm *SI_Construct* to build $\tilde{G}_{SI}$ consists of 6 main steps as shown in Fig. 6. Each step is described as follows:

- **Step 1:** Given an input textual collection Δ, build the corresponding inverted index, i.e., using either $InvIndex_{OA}(\Delta)$ or $InvIndex_{TA}(\Delta)$, and generate the corresponding $\tilde{G}_{\Delta}$ graph as previously defined.
- **Step 2:** Receiving a semantic knowledge graph G_{KB} representing the semantic knowledge base *KB* provided as input, build an inverted index $InvIndex(G_{KB})$ for the string values of each *KB* entity (i.e., *sense* nodes and *term* nodes, to access them more efficiently during resource coupling, and later during query execution), and then construct $\tilde{G}_{KB}$ graph as illustrated previously.

[1] Creating the knowledge base index graph does not involve an input textual collection, thus $\Delta = \varnothing$ (following Definition 5).
[2] The *missing term* problem is discussed in Section 4.4.
[3] The set of index edges in $\tilde{G}_{SI}$ is not exactly equivalent to that in $\tilde{G}_{KB}$ since it might contain additional index edges connected with *index terms* in $\tilde{G}_{\Delta}$ which do not map to any *term* node in $\tilde{G}_{KB}$. This is discussed as the *missing terms problem* in Step 4 of algorithm *SI_Construction* (cf. Fig. 6).

– **Step 3:** Combine the two *SemIndex+* graphs into a single graph structure $\tilde{G}_{SI}$. To do so, we map and then merge all *searchable term* nodes in $\tilde{G}_{\Delta}$, i.e., $\tilde{G}_{\Delta}.V_i^+$, with *searchable term* nodes in $\tilde{G}_{KB}$, i.e., $\tilde{G}_{KB}.V_i^+$, as follows:

  1. For each pair of *searchable term* nodes in $\tilde{G}_{\Delta}.V_i^+$ and $\tilde{G}_{KB}.V_i^+$, if their string values are equal, then remove one of them and merge all the connected edges.

  2. Sense nodes in $\tilde{G}_{KB}$ are kept the same in $\tilde{G}_{SI}$, i.e., $\tilde{G}_{SI}.V_i^\# = \tilde{G}_{KB}.V_i^\#$, but are connected with the corresponding *searchable term* nodes $\tilde{G}_{SI}.V_i^+$

  3. Data nodes and attribute nodes in $\tilde{G}_{\Delta}$ are kept the same in $\tilde{G}_{SI}$, i.e., $\tilde{G}_{SI}.V_d = \tilde{G}_{\Delta}.V_d$, and $\tilde{G}_{SI}.V_a = \tilde{G}_{\Delta}.V_a$, but are connected with the corresponding *searchable term* nodes $\tilde{G}_{SI}.V_i^+$ using the *contained* data edge relationship.

Fig. 7.a shows the result of combining the two sample *SemIndex+* graphs used in our running example: $\tilde{G}_{\Delta}$ of the extract of textual collection Δ_{Part} and $\tilde{G}_{KB}$ of the extract knowledge base *KB*.

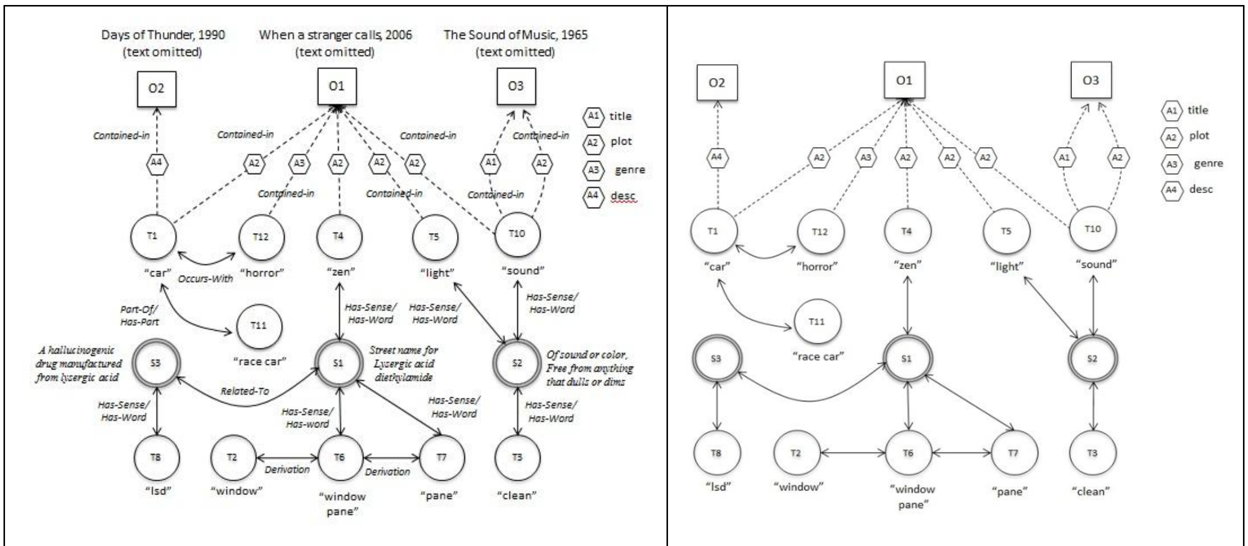
```
Algorithm SI_Construct    // SemIndex+ construction

Input: Δ      // Textual data collection
       KB     // Semantic knowledge base
       W      // Weighting function parameters

Ouput: $\tilde{G}_{SI}$  // SemIndex+ graph

Begin
Step 1: Build InvIndex(Δ) to construct $\tilde{G}_{\Delta}$
Step 2: Build InvIndex($G_{KB}$) to construct $\tilde{G}_{KB}$
Step 3: Coupling $\tilde{G}_{\Delta}$ and $\tilde{G}_{KB}$ into $\tilde{G}_{SI}$ by:
    1. Mapping & Merging searchable term nodes in $\tilde{G}_{\Delta}.V_i^+$ and $\tilde{G}_{KB}.V_i^+$
    2. Including sense nodes from $\tilde{G}_{KB}.V_i^\#$
    3. Including data nodes from $\tilde{G}_{\Delta}.V_d$
Step 4: Run MissingTerms_Linkage algorithm
            // Connect Missing terms in $\tilde{G}_{SI}$
Step 5: Assign weights to edges & data nodes in $\tilde{G}_{SI}$
        - According to parameters W and weighting function $f_W$
Step 7: Remove from $\tilde{G}_{SI}$ :
    1. Labels from all edges: $\tilde{G}_{SI}.E$
    2. String values from all nodes except searchable terms: $\tilde{G}_{SI}.V_i^+$
Return $\tilde{G}_{SI}$
End
```

Fig. 6. Pseudo-code of *SemIndex+* construction algorithm.

– **Step 4:** *Searchable terms* from $\tilde{G}_{\Delta}.V_i^+$ which do not map to any searchable term in $\tilde{G}_{KB}.V_i^+$ can exist, which we identify as *missing terms* (e.g., term "*Horror*" in Fig. 5). These come down to terms from the data collection with no semantic cues in the knowledge base (e.g., "*horror*" appears in object O_1 of Δ_{Part} but does not appear in the extract knowledge base *KB* in Fig. 3). To solve this, we introduce algorithm *MissingTerms_Linkage* (cf. Appendix II) inspired from distributional thesaurus construction methods, e.g. [18, 87], which creates links from each missing term to one or more closely related terms, i.e., terms that co-occur together in the text collection (e.g., term "*horror*" links with "*car*", considered as its most related – highest co-occurrence frequency term – in Δ_{Part}, cf. Fig. 7). The new co-occurrence links (index edges) are labeled *occurs-with.* Our *MissingTerms_Linkage* algorithm is provided in Appendix II since it's outside of the main scope of this paper and will be evaluated in a dedicated future study.

– **Step 5:** Assign weights to edges and textual objects, according to f_W. The weights will be used to select and rank query results. Different weighting functions can be used, which we describe in Section 4.5.

- **Step 6:** It removes edge labels and string values of all nodes in $\tilde{G}_{SI}$ except for V_i^+ (*searchable term* nodes) and V_a (attribute nodes), since all other nodes are not required for processing semantic queries. Removing node string values helps improve *SemIndex+*'s scalability in terms of size, construction time, and query processing time (cf. experiments in Section 7).


Days of Thunder, 1990 (text omitted)
When a stranger calls, 2006 (text omitted)
The Sound of Music, 1965 (text omitted)
O2
O1
O3
A1 title
A2 plot
A3 genre
A4 desc
Contained-in
A4
A2
A3
A1
T1
T12
T4
T5
T10
"car"
"horror"
"zen"
"light"
"sound"
Occurs-With
Part-Of/ Has-Part
T11
"race car"
Has-Sense/ Has-Word
A hallucinogenic drug manufactured from lysergic acid
S3
S1
S2
Street name for Lysergic acid diethylamide
Of sound or color, Free from anything that dulls or dims
Related-To
T8
T2
T6
T7
T3
"lsd"
"window"
"window pane"
"pane"
"clean"
Derivation

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

b. Final *SemIndex+* graph representation.

Fig. 7. *SemIndex+* graph $\tilde{G}_{SI}$ obtained after coupling the data collection and the knowledge base graphs in Fig. 5.

Fig. 7.b illustrates our running example $\tilde{G}_{SI}$ excluding edge and node labels except for *searchable term* nodes and attribute nodes which are required in the querying process. Edge and node weights are omitted from the figure for clearness of presentation.

4.5. *SemIndex+* Weighting Functions

After indexing and coupling the textual resource and the semantic resource into a unified *SemIndex+* graph (i.e., *Step 3* of algorithm *SI_Construct*), and handling the *missing terms* problem (*Step 4*), we introduce a set of weighting functions (*Step 5*) to assign weight scores to *SemIndex+*'s entries, including: *data nodes*, *index nodes*, *attribute nodes*, as well as *data edges* and *index edges*. The weighting functions will be used to effectively select and rank semantically relevant results w.r.t. the user's query (cf. *SemIndex+* query processing in Section 5). Other weight functions could be later added to cater to the index designer's needs.

4.5.1. Index Node Weight

Considering an index node $n_i \in \tilde{G}_{SI}.V_i$, the weight of n_i denoted as $W_{IndexNote}(n_i)$, is computed according to the below formula where we consider "Fan-in" to be the number of nodes connected with the target index node:

$$W_{IndexNode}(n_i) = \frac{Fan-in(n_i)}{\underset{\forall v_j \in \tilde{G}_{SI}.V_i}{Max}\ (Fan-in(n_j))} \quad \in [0,1] \qquad \textbf{(1)}$$

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

4.5.2. Index Edge Weight

Given an index edge $e_i^j \in \tilde{G}_{SI}.E_i$ outgoing from index node n_i and incoming toward index node n_j in the *SemIndex+* graph, we define the weight of e_i^j as:

$$W_{IndexEdge}(e_i^j) = \frac{1}{Fan-out_{Label}(n_i)} \quad \in \ ]0,1] \qquad \textbf{(2)}$$

The weight of an index edge increases with 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.

4.5.3. Data Node Weight

The weight of a data node $n_d \in \tilde{G}_{SI}.V_d$ in the *SemIndex+* graph is defined as:

$$W_{DataNode}(n_d) = \frac{Fan\text{-}In(n_d)}{\underset{\forall n_q \in \tilde{G}_{SI}.V_d}{\text{Max}}(Fan\text{-}In(n_q))} \in [0,1] \qquad \textbf{(3)}$$

where *Fan-In*(n_d) designates the number of foreign key/primary key data links (joins) outgoing from data nodes (tuple) where the foreign keys reside, toward data node (tuple) n_d where the primary key resides. Similarly to index node weight, the rational is that a data node is more important when it receives more links from other data nodes.

4.5.4. Attribute Node Weight

The weight of an attribute node $n_a \in \tilde{G}_{SI}.V_a$ in *SemIndex+* is manually (or semi-automatically[1]) acquired from the data creator/user, since it is a data design issue, such that:

$$W_{AttNode}(n_a) \in [0, 1] \quad \text{where for each } n_d \in \tilde{G}_{SI}.V_d\,,\ \sum_{n_a \in e^d_{\overset{a}{i}}} W_{AttNode}(n_a) = 1 \qquad \textbf{(4)}$$

In other words, the sum of the weights of all *attribute nodes* (excluding the identifier attribute) connected with a given *data node* n_d through any index *term node* $\forall\ n_i \in \tilde{G}_{SI}.V_i^+$ needs to be normalized in order to sum up the full (100%) descriptive power of n_d. For instance, considering our running example *SemIndex+* graph from Fig. 7, we (as users) consider $W_{AttNode}(title) = 0.4$, $W_{AttNode}(genre) = 0.3$, $W_{AttNode}(description) = 0.3$, $W_{AttNode}(year) = W_{AttNode}(plot) = W_{AttNode}(info) = 0.1$. Hence:

- For data object O_1, attribute weights are already normalized such that: $W_{AttNote}(title) + \ldots + W_{AttNode}(info) = 1$
- For O_2, $W_{AttNode}(description)$ needs to be normalized to become = 1, since *description* is the only attribute describing O_2 (and thus should sum the full descriptive power of the data node).
- Similarly for O_3, attribute weights need to be normalized to obtain $W_{AttNote}(title) + W_{AttNote}(plot) + W_{AttNote}(genre) = 1$. By applying linear normalization to the above user chosen weights for instance, we obtain $W_{AttNode}(title) = 0.5$, $W_{AttNote}(genre) = 0.375$, and $W_{AttNote}(plot) = 0.125$.

4.5.5. Data Edge Weight

Given a data edge $e^d_{\overset{a}{i}} \in \tilde{G}_{SI}.E_d$ connecting an index node n_i with a data node n_d through an attribute node n_a (e.g., data edge connecting index node T_1 with data node O_2 through attribute A_4 since the term "*car*" occurs in the textual description of O_2 through its *desc* attribute, likewise for T_1-A_4-O_2, T_4-A_2-O_1,…, T_{12}-A_3-O_1, in Fig. 7), we compute the weight of $e^d_{\overset{a}{i}}$ as an adapted 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 $e^d_{\overset{a}{i}}$ incoming from index node n_i toward data node n_d through attribute node n_a, where $n_i.l$ denotes the string value of n_i, we define:

$$\text{W}_{\text{DataEdge}}(e^d_{\overset{a}{i}}) = \text{TF}(n_i.l, n_d, n_a) \times \text{IDF}(n_i.l, \tilde{G}_{SI}.V_d) \qquad \textbf{(5)}$$

where TF and IDF are calculated as follows:

$$\text{TF}(n_i.l,\ n_d,\ n_a) = \frac{NbOcc(n_i.l) \times W_{AttNode}(n_a)}{\underset{e^d_{\overset{q}{j}} \in \tilde{G}_{SI}.E_d}{Max}(NbOcc(n_j.l))} \in [0,\ 1] \qquad \textbf{(6)}$$

where the number of occurrences of a term $n_i.l$, denoted as $NbOcc(n_j.l)$, is weighted by the corresponding attribute (such that terms occurring through higher weight/more relevant attributes will have a higher impact on the data edge

[1] Learning algorithms can be devised to evaluate the relevance of attributes based on existing data collections [46]. Yet, such approaches also require manual expert input to train the learning algorithms.

weight), and where TF is normalized w.r.t. the maximum number of occurrences of any index node string literal $n_j.l$ within the target data node n_d.

$$\text{IDF}(n_i.l,\ \widetilde{G}_{SI}.V_d) = 1 - \frac{DF(n_i.l, \widetilde{G}_{SI}.V_d)}{N} \quad \in [0, 1[\tag{7}$$

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

5. *SemIndex+* Query Processing

Given the above weighting functions (others could be added later), we define our query model and present our algorithm to perform semantic-aware search with *SemIndex+*.

5.1. Query Model

The *semantic-aware queries* considered in our approach are *conjunctive projection selection* queries of the form $q = \pi_X\, \sigma_P\, \ell(\Delta)$, defined over a data collection Δ (structured, unstructured, or partly structured, cf. Definition 1), where X is a subset of attributes $X \subseteq A \cup \perp$ (where $\perp$ designates the combined textual content from all attributes, allowing both *attribute-sensitive* and *attribute-free* querying), $\ell \in \mathbb{N}$ represents a link distance threshold designating different levels of semantic awareness in query execution, and P is a conjunctive selection predicate defined as follows:

Definition 6 - Conjunctive Selection Predicate: It is defined as an expression P on a string-based attribute[1] or on the combined textual content of all attributes $A_i \in A \cup \perp$[2], of the form: $(A_i\ \theta\ s)$, where s is a user-given string value (e.g., a selection term/keyword), and $\theta \in \{=, like\}$ whose evaluation against values in $dom(A_i)$ is previously defined •

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

- **i. Standard Query**: When $\ell = 1$, the query is a standard containment query, involving only *data edges* (connecting *data nodes* with *searchable term* nodes through *attribute nodes*, using the *contained* relationship), such that no semantic information is involved.
- **ii. Lexical Query**: When $\ell = 2$, the link distance is increased by 1 to include (in addition to *data edges*), first level *index edges*. They designate lexical relationships between *searchable term* nodes (namely the *derivation* relationship, where one term *derives* another term), such that basic lexical information is involved.
- **iii. Synonym-based Query**: When $\ell = 3$, the senses (synsets) are also involved. Here, link distance includes the second level *index edges*: connecting *searchable term* nodes with corresponding *sense nodes* (via the *has-sense* and *has-term* semantic relationships), such that synonymous terms corresponding to the *sense nodes* are involved. Note that there is no direct edge between *data nodes* and *sense nodes.*
- **iv. Extended Semantic Query**: When $\ell \geq 4$, the data graph of *SemIndex+* can be explored in all possible ways, covering *index edges* designating all kinds of semantic relationships (*hyponymy*, *meronymy*, etc.) between *index nodes*, to reach even more semantically relevant results.

While we currently focus on relaxing "strict" conjunctive querying by increasing link distances between query and data nodes, yet our query model and approach can also incorporate different kinds of "weak AND" operators such as *fuzzy* predicates [50, 91] (which we will investigate in an upcoming study).

5.2. Query Answer

The answer to a query $q = \pi_X\, \sigma_P\, \ell(\Delta)$ in *SemIndex+*(Δ, *KB*), noted $q(\Delta)$, is defined as follows:

Definition 7 - Query answer: Given *SemIndex+*(Δ, *KB*) and its graph representation $\widetilde{G}_{SI}$, a query answer $q(\Delta)$ is the set of distinct root nodes of all answer trees in $\widetilde{G}_{SI}$, where every root node represents a data object in Δ. We define an answer tree as a connected sub-graph $T \subseteq \widetilde{G}_{SI}$ satisfying the following conditions:

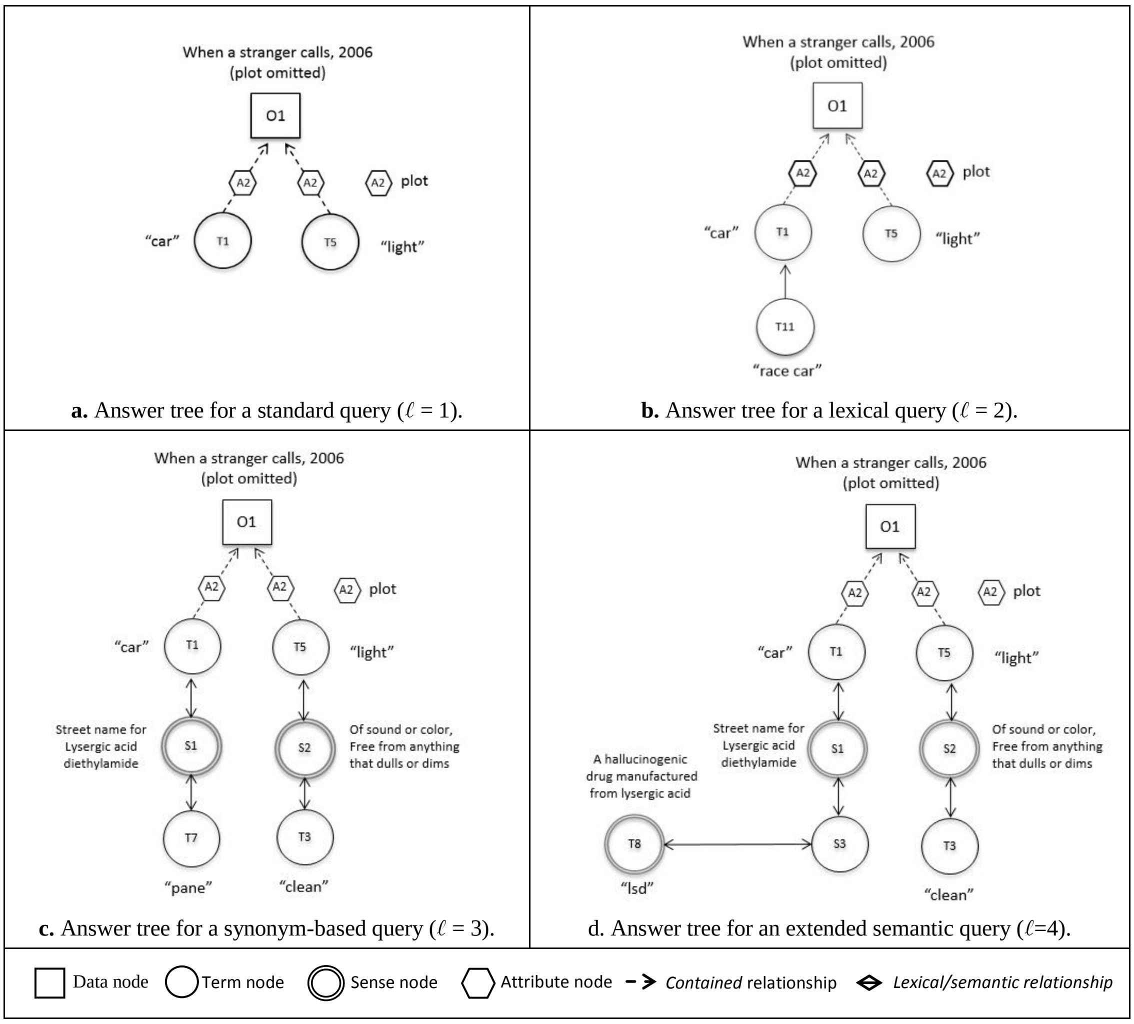
- *Root node*: T's root is a *data node*, i.e., $R(T) \in \widetilde{G}_{SI}.N_d$, and it is the only data node in T, designating the corresponding textual object in Δ to be returned to the user,

[1] Although our approach is generic and can be defined on other types of attributes.

[2] $\perp$ designates the combined textual content from all attributes

– *Leaf nodes*: All leaf nodes in the answer tree T are *searchable term* nodes mapping to query terms (keywords),
– *Tree structure*: For each node $n \in T$, there exists exactly one directed path from n to T's root node R(T),
– *Depth boundary*: The depth of T, i.e., the maximal number of edges between the root and a leaf node, is not greater than the link distance threshold ℓ,
– *Minimal tree*: No node can be removed from T without violating some of the above conditions.

The answer tree comes down to a conjunction of paths starting at leaf nodes designating each a query term, and ending at a common root designating the textual data object to be returned as result •


When a stranger calls, 2006
(plot omitted)
O1
A2
plot
"car"
T1
T5
"light"
T11
"race car"
S1
S2
Street name for Lysergic acid diethylamide
Of sound or color, Free from anything that dulls or dims
T7
T3
"pane"
"clean"
A hallucinogenic drug manufactured from lysergic acid
T8
S3
"lsd"
Data node
Term node
Sense node
Attribute node
Contained relationship
Lexical/semantic relationship

a. Answer tree for a standard query (ℓ = 1). **b.** Answer tree for a lexical query (ℓ = 2).

c. Answer tree for a synonym-based query (ℓ = 3). d. Answer tree for an extended semantic query (ℓ=4).

Fig. 8. Sample answer query trees[1] with different link distance threshold values ℓ, extracted from our running example *SemIndex+* graph (Fig. 7).

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

i. Standard Query: When ℓ = 1, the root of the answer tree is linked directly to all leaves, representing the fact that the result data object contains all query terms directly. A sample answer tree is shown in Fig. 8.a for query $q = \sigma_{plot\ \in\ (\text{"car"},\ \text{"light"})}\ \ell_{=1}(\Delta_{\text{Part}})$ considering our running example data collection Δ_{Part} (Table 3) and the corresponding *SemIndex+*(Δ_{Part}, *KB*) (Fig. 7),

[1] While all edge and node labels are removed from the *SemIndex* graph except for *searchable term* nodes, we show *synset* node glosses here for the sake of presentation.

ii. Lexical Query: When ℓ = 2, the answer tree includes lexical connections between query term nodes and other index term nodes. Fig. 8.b is an example answer tree for query $q = \sigma_{plot \in (\text{“}race\ car\text{”},\text{“}light\text{”})}\ \ell_{=2}\ (\Delta_{\text{Part}})$,

iii. Synonym-based Query: When ℓ = 3, the answer tree includes *sense nodes*, in addition to the two previous cases. Note that due to the *minimal tree* restriction (Definition 7 -), a sense node cannot be a leaf node of an answer tree. Thus, if an answer tree contains a sense node, the height of the tree is not less than 3. A sample answer tree is shown in Fig. 8.c for query $q = \sigma_{plot \in (\text{“}pane\text{”},\text{“}clean\text{”})}\ \ell_{=3}\ (\Delta_{\text{Part}})$. The synonyms of the two query terms, *“zen”* and *“light”* are also contained in the answer tree rooted at the data node of object O_1,

iv. Extended Semantic Query: When ℓ = 4, the answer tree contains additional index nodes connected via index edges designating different semantic relationships, according to the provided input selection terms. An example answer tree is shown in Fig. 8.d for query $q = \sigma_{plot \in (\text{“}lsd\text{”},\text{“}clean\text{”})}\ \ell_{=4}(\Delta_{\text{Part}})$.

Note that it is possible to have more than one path from a query term node to a data node in the *SemIndex+* graph (through different semantic links), which will naturally result in more than one possible answer tree.

5.3. Relevance Ranking

While a huge number of query answers could be identified for a given query, the objective of any typical IR system would be to identify the most relevant of these candidate results and rank them based on their respective relevance w.r.t. the query [9]. In *SemIndex+*, we evaluate the relevance of data nodes returned as candidate query answers (i.e., answer tree root nodes) using typical *Dijskstra*-style shortest distance computations (described in the following section). Yet, instead of identifying the shortest distance between *searchable term* nodes and *data nodes* in the *SemIndex+* graph, we compute their maximum similarity (as the inverse of distance). Formally:

Definition 8 - Relevance Score measure: Given a *SemIndex+* graph $\tilde{G}_{SI}$, a data node $n_d \in \tilde{G}_{SI}.V_d$ and a searchable term node $n_i \in \tilde{G}_{SI}.V_i^+$ (cf. visual representation in Fig. 9), we define the relevance (similarity) score of n_d w.r.t. n_i, noted $score(n_d, n_i)$, as the sum of the inverse of the lexical/semantic distances (in number of edges) between n_d and every index node on the path leading from n_d to n_i, noted $path(n_d, n_i) = \langle n_d, n_p, n_q, \ldots, n_j, n_i \rangle$, where every node and edge on $path(n_d, n_i)$ is weighted following *SemIndex+*'s weighting functions:

$$score(n_d, n_i) = \left. \begin{bmatrix} W_{DataNode}(n_d) \times W_{AttNode}(n_a) \times w_{DataEdge}(e^{d}_{a_p}) \times \dfrac{1}{d(n_d, n_p)} + \\ W_{IndexNode}(n_p) \times W_{IndexEdge}(e^{p}_{q}) \times \dfrac{1}{d(n_d, n_q)} + \ldots + \\ W_{IndexNode}(n_j) \times W_{IndexEdge}(e^{j}_{i}) \times \dfrac{1}{d(n_d, n_i)} \end{bmatrix} \middle/ d(n_d, n_i) \right. \in [0,1] \quad (8)$$

where $d(n_d, n_i)$ designates the distance in number of edges between data node n_d and index node n_i •

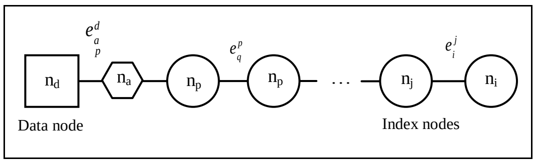


Fig. 9. Sample node linkage representation in the *SemIndex+* graph.

Our relevance score measure in Definition 8 - produces normalized relevance scores ∈[0, 1] where:

- A minimum relevance score =0 is reached when searchable term node n_i is not connected with n_d, i.e., there is no path $path(n_d, n_i)$ leading from n_i to n_d in $\tilde{G}_{SI}$.
- A maximum relevance score =1 is reached when searchable node n_i is directly connected with data node n_d through data edge $e^{d}_{a_i}$ (the searchable term occurs in the string value of the data node). For instance, this is the case of index node n_p in Fig. 9, which can produce $score(n_d, n_p) = W_{DataNode}(n_d) \times W_{AttNode}(n_a) \times w_{DataEdge}(e^{d}_{a_p}) \times \dfrac{1}{d(n_d, n_p)} = 1$ where $d(n_d, n_p)$ is minimum (=1) and given that all three: data node, attribute node, and data edge weights are maximum (=1).

- The relevance score increases with the semantic/lexical closeness between n_d and n_i in $\widetilde{G}_{SI}$, and decreases with their distance. In other words, the farther away a searchable term node n_i is from data node n_d in $\widetilde{G}_{SI}$ (i.e., the higher the distance $d(n_d, n_i)$), the lesser its semantic/lexical relatedness with n_d, and thus the lower its relevance score w.r.t. n_d.
- The relevance score also increases/decreases with *SemIndex+* node/edge weighting functions, allowing users to easily consider or disregard relevance weights following their needs (e.g., one user could prefer to consider *data node* weights only, while disregarding others).

Consider the example in Fig. 8.c. Here, we assume that users disregard all weighting functions for simplicity:

- The relevance of data node O_1 w.r.t. term node T_1, where T_1 is situated at link distance ℓ=1 from O_1 (direct linkage, where T_1 occurs in the string value of O_1): $score(O_1, T_4) = \left(1\times1\times1\times\frac{1}{1}\right)/1 = 1$ (i.e., maximum score).
- The relevance of O_1 w.r.t. S_1, at link distance ℓ =2: $score(O_1, S_1) = \left(\left(1\times1\times1\times\frac{1}{1}\right) + \left(1\times1\times\frac{1}{2}\right)\right)/2 = 0.75$
- The relevance of O_1 w.r.t. T_7, at link distance ℓ =3 $score(O_1, T_7) = \left(\left(1\times1\times1\times\frac{1}{1}\right) + \left(1\times1\times\frac{1}{2}\right) + \left(1\times1\times\frac{1}{3}\right)\right)/3 \simeq 0.6112$

One can clearly realize that index nodes T_1, S_1, and T_7 which are (semantically/lexically) decreasingly related to (they are increasingly more distant in the *SemIndex+* graph from) data node O_1, produce decreasing relevance scores (1, 0.75, and then 0.6112) respectively.

5.4. Querying Algorithm

The pseudo-code for our query processing algorithm, titled *SI_PSS*, is shown in Fig. 10. It takes as input a *SemIndex+* graph $\widetilde{G}_{SI}$, a conjunctive projection selection query q including link distance threshold ℓ, as well as *range* and *kNN* (k-nearest neighbor) query selection thresholds r and k, and produces as output the list of data nodes N_{d_Out} (i.e., the answer trees' root nodes) designating the data objects returned as query answers, selected and ordered following their relevance w.r.t. the query. It is parallelized (multithreaded), processing query terms and starting index nodes using multiple threads running in parallel. The overall process can be described as follows:

- **Step 1:** Every query term is assigned to a dedicated thread, and is thus processed independently from other query terms (line 2).
- **Step 2:** The algorithm then identifies in $\widetilde{G}_{SI}$ the index (*searchable term*) nodes mapping to each query term (using function *getNodeIDs()*, line 4) [1]. These will serve as starting nodes to navigate the *SemIndex+* graph.
- **Step 3:** Every starting (index) node is then assigned to its own dedicated thread, and processed independently from other starting nodes (line 5),
- **Step4:** For every starting (index) node, the minimum distance paths at ℓ from the starting node to data nodes are identified, i.e., using *Dijkstra*'s shortest path algorithm (performed by function *findShortestPaths()*, line 7).
- **Step 5:** Of these shortest paths, the algorithm then identifies the paths which contain data nodes (using function *getDataNodeIDs( )*, line 8), reachable through the designated query attribute for every term, and then adds the resulting data nodes to the list of output data nodes N_{d_Out}.
- **Step 4:** Consequently, the resulting data nodes are gradually merged with the list of existing answer data nodes as they are produced by each thread[2]. A score is then assigned to every answer node by computing its relevance score w.r.t. every query term's index node (using *mergeAndRank()*, line 9). The algorithm finally returns the list of answer data nodes, ranked in descending order following answer (data) node relevance scores (i.e., from the most to the least relevant answer node).
- **Step 5:** Data node result selection (line 12) is undertaken using a combined *range-kNN* query selection operator, following *range query* and *kNN* thresholds provided by the user. The user can choose to apply one, both, or none of the two selection operators, by specifying (disregarding) the value of the corresponding threshold(s).

---

[1] Initial index/query term mapping is performed regardless of query attributes. Term mapping identifies the leaf nodes of potential answer trees in the *SemIndex+* graph, which will be later pruned following the terms' container attributes (if any) in Step 3 (graph navigation phase) of the algorithm.

[2] The physical implementation the querying algorithm is configured to run as many threads as necessary to process the different query terms and starting nodes, where thread scheduling and parallel execution are left to the operating system.

Note that the scores of data nodes returned as query answers (i.e., answer tree root nodes) are computed/updated dynamically while executing function *findShortestPaths()* based on typical *Dijkstra* shortest distance computations [28]. Basically, *findShortestPaths()* explores the *SemIndex+* graph with *Dijkstra*'s algorithm from multiple starting index nodes n_{i_In} (multiple query terms T_i). For each visited node n_j, it stores its maximum relevance scores (minimum distances) from all starting nodes (query terms). The relevance score of an index node n_j (likewise for a data node n_d) w.r.t. a starting node (query term) n_{i_In} is evaluated using our relevance score measure (Definition 8 -) applied along the path between n_{i_In} and n_j (n_d). In other words, the shortest distance of n_i (n_d) from n_{i_In} is identified by computing the maximum relevance score of n_i (n_d) w.r.t. n_{i_In}.

```
Algorithm SI_ParallelSemanticSearch      // SemIndex+ Parallel Semantic Search
Input:  $\tilde{G}_{SI}$       // SemIndex+ graph
        q          // A conjunctive projection selection query, including link distance threshold $\ell$
        {r, k}     // range and k-nearest neighbor selection operators
Ouput: $N_{d\_Out}$   // A list of ranked data nodes from $\tilde{G}_{SI}$ designating query answers
Begin
  $N_{d\_Out} = \phi$
  Create Thread for each $(T_i, A_j) \in q$          // Processing each selection term simultaneously
  {
    Step 1: $N_{i\_In}$ = getNodeIDs($T_i$, $\tilde{G}_{SI}$)      // Identify index nodes for every selection term
    Create Thread for each $n_i \in N_{i\_In}$          // Processing every (starting) index node simultaneously
    {
        Step 2: SP = findShortestPaths($n_i$, $\ell$, $\tilde{G}_{SI}$)      // Identify shortest paths within distance $\ell$ from $n_i$
        Step 3: $N_{d\_ni}$ = getDataNodeIDs(SP, $\tilde{G}_{SI}$, $A_j$)   // Identify the set of data (root) nodes in each shortest path
                                                      //reachable from the term node $n_i$ through attribute $A_j$
        Step 4: $N_{d\_Out}$ = mergeAndRank($N_{d\_ni}$, $N_{d\_Out}$)   // Merging and ranking data nodes based on relevance
    }
  Step 5: $N_{d\_Out}$ = select($N_{d\_Out}$, {range, kNN})      // Result Selection using range and/or kNN threshold(s)
  Return $N_{d\_Out}$
End
```

Fig. 10. Pseudo-code of *SemIndex+* parallel semantic search algorithm.

For example in Fig. 8.c, given query $q = \sigma_{plot\ \in\ (\text{"}pane\text{"},\text{"}clean\text{"})}\ \ell{=}3\ (\Delta_{\text{Part}})$ made of terms "*pane*" and "*clean*", the algorithm starts to expand from index nodes T_7 and T_3. In this example, we disregard (i.e., assign unit scores to) all *SemIndex+* node/edge weighting functions (to simplify computations) in evaluating our relevance score measure. Hence, the relevance score of T_7 is initialized as a vector of two scores <1, 0>, the first representing the relevance score w.r.t. T_7 ("*pane*"), i.e., *score*(T_7, T_7)=1, and the second representing relevance score w.r.t. T_3 ("*clean*", i.e., *score*(T_7, T_3) = 0 since T_3 is not initially reachable from T_7). Similarly, the weight score vector of T_3 is initialized to <0, 1>. The weights of all other index nodes are initialized to <0, 0>. The relevance scores are then updated when each edge is explored in the graph. For example, starting from T_7, the weight of index node S_1, which was initialized to <0, 0> becomes <1, 0> when the node is reached, where *score*(S_1, T_7) = 1 (relevance score at link distance ℓ=1 from T_7) and *score*(S_1, T_3) = 0 (since S_1 is not yet reachable from T_3). Likewise, the weights of nodes T_1 and O_1 become <0.75, 0> and <0.6112, 0> respectively when the nodes are reached from T_7, and so forth. On the other hand, starting from T_3, the weights of nodes S_2, T_5, and O_1 become <0, 1>, <0, 0.75>, <0, 0.6112> respectively.

Consequently, given that a data node n_d can be reached from multiple starting nodes N_{i_In} (i.e., multiple leafs in the answer tree), function *mergeAndRank()* computes the combined relevance score of a data node (i.e., answer tree root node) as the aggregate relevance scores from each starting node (each answer tree leaf node). As for the aggregation function, various mathematical formulations for combining relevance scores can be used [5, 83], among which the *maximum*, *minimum*, *average* and *weighted sum* functions. Here, we utilize the *average* aggregation function to account for the average semantic relatedness between the query answer root node and all tree leaf nodes:

$$\text{score}(n_d) = \underset{\forall\, n_{i\_In} \text{ having } s_i \in S}{avg}\ score(n_{i\_In},\ n_d) \qquad \textbf{(9)}$$

For instance, considering our current example based on Fig. 8.c, the vector path score of data node O_1 would be <0.6112, 0.6112>, and thus its combined path score becomes 0.6112. Considering the example in Fig. 8.b, starting from query terms "*race car*" and "*light*", the vector path score of data node O_1 would be <0.75, 1> (assuming unit *SemIndex+* node/edge weights as in the previous example), and thus its combined path score becomes 0.875. A data node which is not reachable from all query term nodes will have at least one relevance score =0 (i.e., zero semantic relatedness), along one (or more) of its relevance vector dimensions.

6. Complexity Analysis

Our solution is of quadratic complexity, requiring $\boldsymbol{O(N^2)}$ time for building the *SemIndex+* graph where N represents the maximum size (in number of nodes) between the textual collection and the knowledge base, and $\boldsymbol{O(N_{i\_acc}{}^2)}$ time for executing semantic-aware queries where $\boldsymbol{N_{i\_acc}}$ is the number of index nodes accessed during query execution.

6.1. Building *SemIndex+*

6.1.1. Time Complexity

Building *SemIndex+* using algorithm *SI_Construct* (cf. Fig. 6) is done in quadratic time and simplifies to $\boldsymbol{O(N^2)}$ since:

- Step 1: Building the inverted index, and consequently the *SemIndex+* graph for the textual collection Δ, i.e., $\tilde{G}_\Delta$, is of typical $O(|\Delta| \times |A| \times N_\Delta)$ complexity, where $|A|$ and N_Δ designate the number of attributes and the number of *searching terms* in $\tilde{G}_\Delta$ respectively, which simplifies to $O(|\Delta| \times N_\Delta)$ since $|A|$ is usually limited,
- Step 2: Also, building the *SemIndex+* graph for the knowledge base *KB*, $\tilde{G}_{KB}$, is of $O(|KB| \times N_{KB})$, where N_{KB} is the number of *searchable term* nodes from $\tilde{G}_{KB}$,
- Step 3: Coupling both Δ and *KB*'s *SemIndex+* graphs by mapping and merging *searchable term* nodes in both $\tilde{G}_\Delta$ and $\tilde{G}_{KB}$ can be performed in $O(N_\Delta + N_{KB})$ time, given that both underlying structures are sorted,
- Step 4: Connecting missing terms with the merged index, using the algorithm *MissingTerms_Linkage* (cf. Appendix II) can be performed in worst case $O(N_{miss} \times N_{term})$, where N_{miss} and N_{term} respectively designate the number of *missing terms* and the number of *term* index nodes in the *SemIndex+* graph. Note that building the distributional thesaurus (to identify term relatedness vectors, based on their co-occurrences in the reference corpus) is conducted offline prior to *SemIndex+* building and thus does not affect its complexity.
- Step 5: The complexity of the weighting process varies according the weight functions used. It amounts to $O(1)$ when assigning equal weights, and varies following our weighting scheme as follows:
  - Data nodes: assigning an *object rank* score to compute data node weights simplifies to $O(|\Delta|+|\Delta_{\text{Joins}}|)$, where $|\Delta_{\text{Joins}}|$ designates the number of data links (i.e., foreign key/primary key data joins) in Δ,
  - Attribute nodes: normalizing user defined attribute weights for every attribute of every data node required $O(|\Delta|\times|A|)$ time, which simplifies to $O(|\Delta|)$ since $|A|$ is small w.r.t. $|\Delta|$,
  - Data edges: performing attribute-sensitive *term frequency - inverse document frequency* computations to assign data edge weights comes down to $O((N_{term})\times|\Delta|\times|A|)$ time, which simplifies to $O((N_{term}) \times |\Delta|)$ since $|A|$ is small w.r.t. N_{term} and $|\Delta|$,
  - Index nodes: assigning an *object rank* score to compute index node weights simplifies to $O(N_i + NE_i)$, where N_i designates the number of index nodes and NE_i the number of index edges in the *SemIndex+* graph,
  - Index edges: computing index edge weights comes down to $O(NE_i \times |L|)$, where $|L|$ designates the number of distinct lexical/semantic relationships, which simplifies to $O(NE_i)$ since $|L|$ is usually small.
- Step 6: Edge aggregation between each pair of index nodes in the *SemIndex+* graph can be performed in $O((N_{term} + N_{sense})^2 /2)$ time where N_{sense} designates the number of *sense* index nodes, which is the time needed to go through all pairs of index nodes in *SemIndex+*,
- Step 7: Removing edge labels and string values from non-searchable (i.e., sense) nodes in *SemIndex+* can be executed in $O\,(N_{E} + N_{sense})$, where N_E designates the number of index and data edges.

Hence, the overall complexity of our *SemIndex+* building process is bounded by $O(N^2) > \sum_{i=1\ldots7} Complexity(Step_{\,i})$ since $N \geq param,\ \forall\ param \in$ complexity parameters.

6.1.2. Space Complexity

Our approach requires space to store the final *SemIndex+* graph $\tilde{G}_{SI}$, which is also bounded by $\boldsymbol{O(N^2)}$ space, since storing data nodes requires $O(|\Delta|)$ space, storing attribute nodes requires $O(|A|)$ space, storing data edges (connecting data nodes with *searchable term* nodes through attribute nodes) requires in the worst case $O(|\Delta| + (N_\Delta \times |A|))$ space, storing index nodes requires $O(N_{term} + N_{sense})$ space, and storing index edges (connecting pairs of index nodes) requires $O((N_{term} + N_{sense})^2 /2)$ space (recall that only one edge exists between two nodes in *SemIndex+*). Note that these relations, whose total size is bounded by $O(N^2)$, can be stored on disk or in memory according to the size of the input textual collection and knowledge base used.

6.2. Query Processing

The complexity of our *SI_PSS* algorithm (cf. Fig. 10) which performs querying on *SemIndex+*, simplifies to $\boldsymbol{O(M^2)}$ in the worst case, and comes down to the sum of the complexities of its underlying functions, where for each query term:

- *getNodeID( )* identifies the IDs of *term* nodes in the *Lexicon* corresponding to the query term, and thus requires in the worst case $O(N_{term} + N_{sense})$ time,
- *findShortestPaths( )* runs *Dijkstra*'s algorithm to identify the minimum paths at distance ℓ from each of the starting *term* nodes N_{term_hom} (i.e., homonymous query terms), which comes down to $O(N_{i\_acc}^{\;2} \times \ell \times N_{term\_hom})$ where N_{i_acc} designates the number of accessed index nodes in *SemIndex+* when executing a query,
- *getDataNodeIDs()* identifies the IDs of data nodes in *PostingList* for each shortest path, requiring $O(|\Delta| + N_{\Delta})$,
- *mergeAndRank( )* merges and ranks data nodes with existing query answer nodes, by comparing the latter with node IDs in the *PostingList*, thus requiring at most $O(|\Delta| \times N_{d\_acc})$ where N_{d_acc} designates the number of accessed data nodes when executing a query,
- *select()* selects data nodes as results, using *range* and/or *kNN* selection operators, requiring at most $O(N_{d\_acc})$.

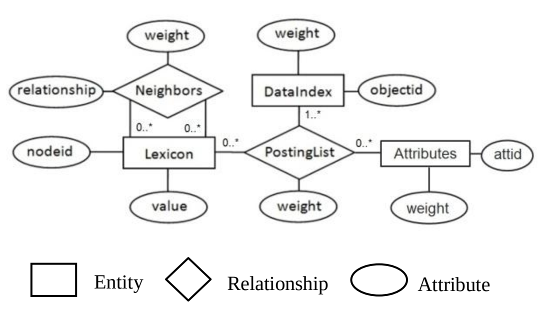
Hence, *SI_PSS*'s complexity comes down to that of function *findShortestPaths( )* applied on *k* query terms while multithreading, which requires $O((k \times (N_{i\_acc}^{\;2} \times \ell \times N_{term\_hom})) \;/\; |Threads|)$, where the algorithm allows as many simultaneous shortest path calls as there are threads[1]. It simplifies to $O((k \times N_{i\_acc}^{\;2}) \;/\; |Threads|)$, which can further simplify to $O(N_{i\_acc}^{\;2} \;/\; |Threads|)$ given that *k* is usually limited (e.g., keyword queries on the Web are usually 2-3 words long [53]), and is bounded by $O(N_{i\_acc}^{\;2})$ in the worst case (when applied on single thread/non-parallel systems).

7. Experimental Evaluation

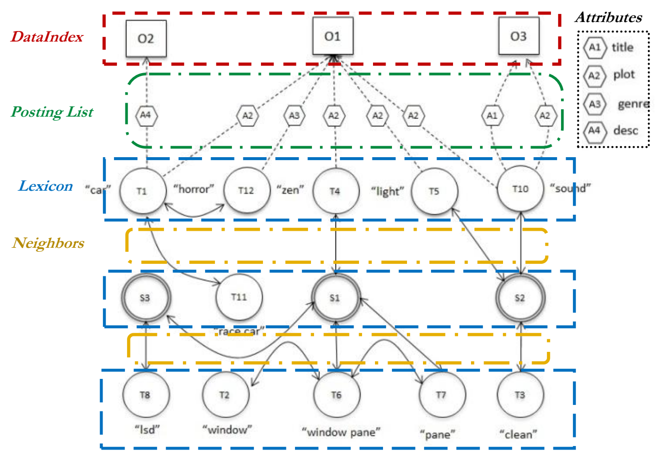
We first start by describing our prototype and experimental scenario, and then we present experimental results.

7.1. Prototype System

We have implemented the *SemIndex+* framework using open source technologies, namely: Java as the programming platform, MongoDB[2] to handle the textual collection, WordNet 3.0 as the reference knowledge base, and MySQL 5.6 to persist the graph structures of SemIndex+. The physical design of *SemIndex+* is shown in Fig. 11. We chose to handle it using a well known RDBMS (i.e., MySQL) to take advantage of its different useful features, including concurrency control and powerful index and memory management: allowing bulk index data loading and fast query execution[3]. Note that *SemIndex+*'s physical design is independent of the DB structure and system used, and can be built directly on top of the file system, or using any other DB system. The prototype system is available online[4].



a. Conceptual ER model describing *SemIndex+*'s physical design.



b. Data representation of each relation in the resulting DB schema.

Fig. 11. *SemIndex+* physical design.

[1] In the physical implementation of the algorithm, thread scheduling and parallel execution are left to the operating system.

[2] NoSQL DB using BSON binary format for storing documents in the JSON format, https://www.mongodb.com/

[3] Preliminary experiments showed that handling SemIndex+ using MySQL was more efficient in build time and query execution time, compared with MongoDB, which we will highlight in a dedicated study.

[4] Available at: http://sigappfr.acm.org/Projects/SemIndex/

7.2. Experimental Scenario and Test Data

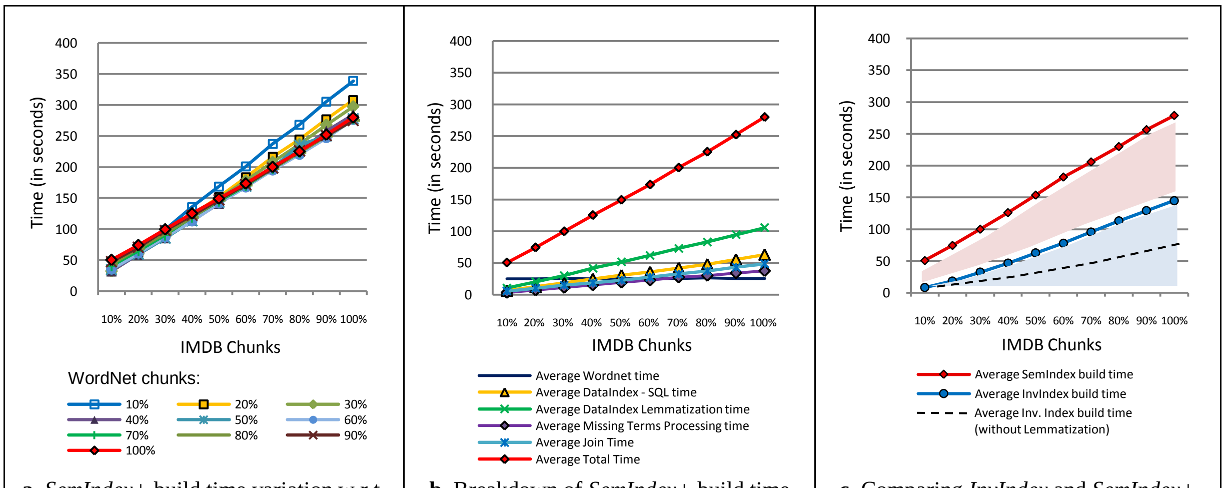
We evaluated the practical usability of our indexing approach by assessing four main criteria: i) index building time, ii) index size and characteristics, iii) query processing time, and iv) the quality of returned results. To do so, we varied the size of the input textual collection Δ by generating different extracts w.r.t. its total size (considering 10%, 20%, …, or 100% of Δ). We also varied the size of the input knowledge base by generating different extracts w.r.t. its total size (considering 10%, 20%, …, or 100% of *KB*). Then, for each doublet <Δ chunk ; *KB* chunk>, we evaluated each of the above four criteria by varying related parameters.

We used the IMBD *movies* dataset[1] as an average-scale[2] input textual collection, including attributes *movie_id* and (*title*, *year*, *plot*, *genre*, *info*) with a total size of around 75 MBytes including more than 7 million data (movie) objects. Three versions of the *movies* data collection were considered: the original (structured) version (cf. Table 1), a flat version where all attribute contents were concatenated in one column (cf. Table 2), and a partly structured version where certain attribute contents were randomly combined or omitted (cf. Table 3). In the remainder of this paper, we utilize the NoSQL version of IMDB *movies* to perform our experiments. WordNet 3.0 has a total size of around 26 Mbytes, including more than 117k synsets (senses). IMBD and WordNet chunk characteristics are summarized in Appendix I. Tests were carried out on a PC with an *Intel I7* system with 2.9 GHz CPU, and 8GB RAM.

7.3. Index Building Time

Fig. 12.a shows the total time required to build *SemIndex+* while varying both IMBD and WordNet chunks. One can realize that the building time is linear in the size of the IMDB chunks on one hand (*x* axis), and linear on the size of the WordNet chunks on the other hand (*y* axis), which underlines quadratic time dependency w.r.t. both of them (which complies with our complexity analysis). We also measured the total time required to build the traditional inverted index (which we note *InvIndex*) while varying IMDB chunk size[3] (cf. Fig. 12.b) and compared results with *SemIndex+*'s (cf. Fig. 12.c). While both indices require linear building time, yet *SemIndex+* requires almost twice (×2) as much build time as *InvIndex*. Also, by disregarding the lemmatization phase in building *InvIndex* (which can be ignored following the data manager's preference: storing words in their actual rather than their original form), then *SemIndex+* build time becomes almost four times (×4) greater than that of *InvIndex*. This is encouraging since even the fastest inverted index creation time is only (at best) four times lesser than the creation time of *SemIndex+*. The reasons for this are: i) the lightweight physical design of *SemIndex+* which can be easily created using fast legacy database technology, and ii) the sheer difference in size between the textual data collection (IMBD *movies*) and the reference knowledge graph (WordNet), which renders the build time of *SemIndex+* mostly dependent on IMBD size.

Regardless of the above, note that the index building process is done offline, prior (in preparation) to the system usage (query evaluation process), and thus does not affect (online) query execution time.



a. *SemIndex+* build time variation w.r.t. input IMDB and WordNet chunk sizes

b. Breakdown of *SemIndex+* build time with WordNet chunk = 100%

c. Comparing *InvIndex* and *SemIndex+* build time using WordNet chunk = 100%

Fig. 12. Breakdown of *SemIndex+* build time, compared with *InvIndex* build time.

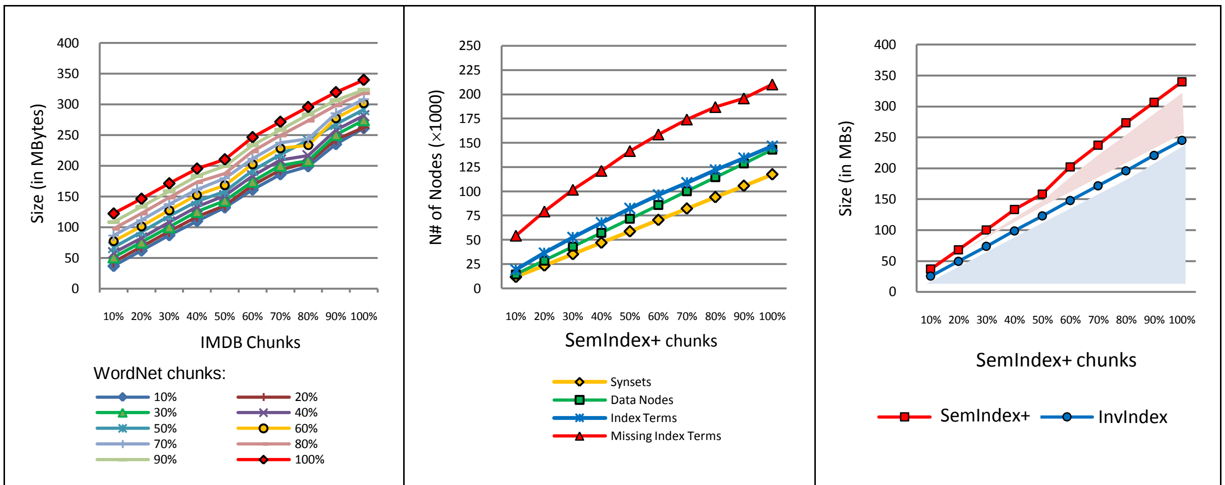
[1] Internet Movie DataBase raw files are available from online http://www.imdb.com/. We used a dedicated data extraction tool (at http://imdbpy.sourceforge.net/) to transform IMDB files into a RDB.

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

[3] Recall that *InvIndex* does not incorporate semantic knowledge and thus is not affected by WordNet chunk size variations.

7.4. Index Size and Characteristics

Regarding *SemIndex+* size, Fig. 13.a shows that the *SemIndex+* graph size varies linearly with the size of the IMDB chunks (*x* axis) and WordNet chunks (*y* axis), which underlines quadratic size dependency w.r.t. both of them (conforming with our complexity analysis). The characteristics of *SemIndex+* chunks are shown in Fig. 13.b (and Appendix I), where each chunk is generated by merging the corresponding <Δ chunk ; *KB* chunk> doublet (for instance, the 10% *SemIndex+* chunk is generated by merging the 10% Δ chunk with the 10% *KB* chunk, and so forth).



a. *SemIndex+* size variation w.r.t. input IMDB and WordNet chunk sizes

b. Breakdown of the number of nodes in the *SemIndex+* graph

c. Comparison with *InvIndex* size

Fig. 13. *SemIndex+* size characteristics and comparison with *InvIndex* size.

First, results show that the number of nodes in the *SemIndex+* graph increases almost linearly w.r.t. *SemIndex+* (and thus IMDB and WordNet) chunks size. Second, one can realize that the number of index nodes resulting from *missing terms* is almost twice that of matching index terms. That is due to the fact that the IMDB *movies* table includes many textual tokens which are not part of the general purpose English language and thus do not appear in WordNet (e.g., terms like "advogado", "advon", "adyeri", "aeer", "moustafa", etc.). We are currently investigating ways to further alleviate the *missing terms* problem, using dedicated language processors and multilingual dictionaries, which will be covered in an upcoming study.

In addition, we have also measured the characteristics and size of *InvIndex* in comparison with *SemIndex+* (cf. Fig. 13.c). Results show that *SemIndex+*'s size is larger only by (almost) 1/3rd of the size of *InvIndex*. This increase in size is less pronounced than the increase in build time of *SemIndex+* (which was 4 times larger) compared with *InvIndex*. This is due to the difference in sizes between the textual data collection (IMBD *movies*) and the knowledge graph (WordNet) used: WordNet (≈ 26 MBytes) is almost 1/3rd the size of IMDB (≈ 75 MBytes), which reflect in the sizes of *SemIndex+* (coupling IMDB with WordNet) and *InvIndex* (referencing IMDB only).

7.5. Query Processing Time

To test the performance of *SemIndex+*, we formulated different kinds of queries categorized following four main criteria shown in Table 4: i) *unrelated queries* with *varying number of attributes*, ii) *unrelated queries without attributes*, iii) *expanded queries* with *varying number of attributes*, and iv) *expanded queries without attributes*.

The first query group *Q1* consists of queries with varying numbers of selection terms (keywords) from 1-to-5, where all terms are different and all queries are unrelated, such that the number of attributes per query varies from 1-to-3. The second query group *Q2* consists of queries made of unrelated terms without attributes (i.e., queries targeting the whole textual contents of the searched objects). Query groups *Q3* and *Q4* are comparable to *Q1* and *Q2* respectively, except that queries are related such that each query expands its predecessor by adding an additional selection term to the latter. In other words, groups *Q1* and *Q3* consist of NoSQL keyword queries whereas *Q2* and *Q4* consist of "traditional" unstructured keyword queries.

Each query was tested on every one of the 100 combinations of *SemIndex+* generated by combining the different chunks of the IMDB *movies* data collection (10%, 20%, …, 100%) with every chunk of WordNet (10%, 20%, …, 100%), at link distance threshold values varying from $\ell = 1$ to 5 (i.e., increasing semantic coverage).

7.5.1. *SemIndex+ Query Processing Time*

On the one hand, the graph in Fig. 14.a plots *SemIndex+*'s query execution time averaged over all queries, considering different IMBD and WordNet chunk sizes, with a fixed number of query terms *k* and a fixed link distance threshold ℓ.

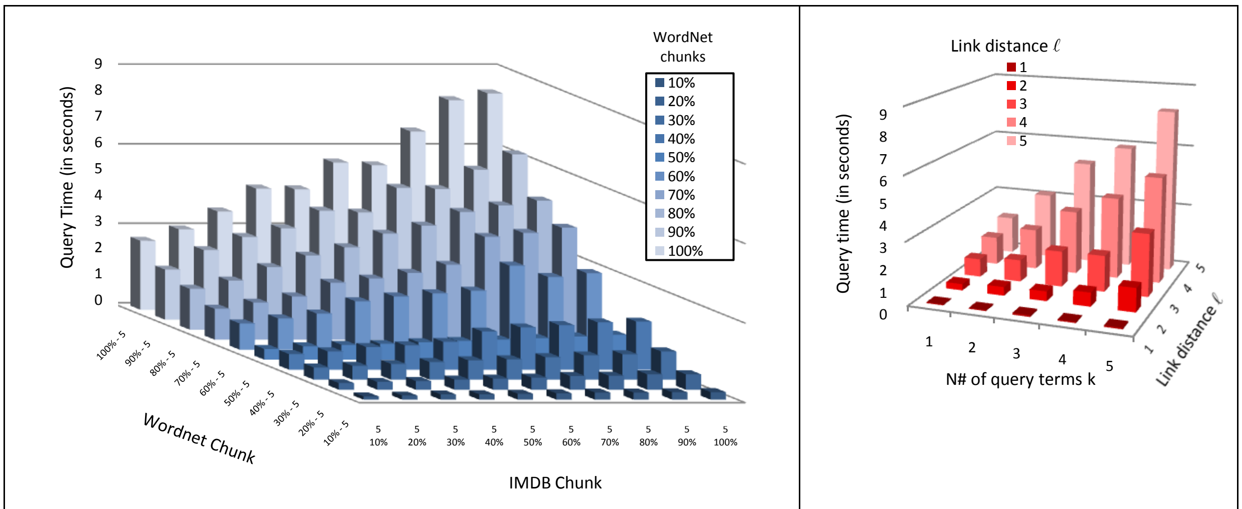
Results show that query execution is linear in both IMBD and WordNet chunk sizes, and thus is quadratic w.r.t. both of them (verifying our complexity analysis). On the other hand, the graphs in Fig. 14.b highlight the effects of varying the number of query terms k and varying link distance ℓ w.r.t. fixed IMDB and WordNet chunk sizes. One can see that processing time is linear w.r.t. the number of query terms, and quadratic w.r.t. link distance, which corresponds to the time complexity of *SemIndex*+'s querying algorithm in navigating the edges (pairs of nodes) of the *SemIndex+* graph.

Table 4. Sample test queries used in our experiments.

Group *Q1* *Unrelated queries* with *varying number of attributes*		**Group *Q2*** *Unrelated queries without attributes*[1]	
Q1_1	$\sigma_{title\in}$ ("time") Δ_{IMDB}[2]	Q2_1	$\sigma_{\perp\in}$ ("music") Δ_{IMDB}
Q1_2	$\sigma_{title\in}$ ("love", "date") Δ_{IMDB}	Q2_2	$\sigma_{\perp\in}$ ("romance", "dinner") Δ_{IMDB}
Q1_3	$\sigma_{title\in}$ ("fly", "power") $\wedge$ *plot*$\in$ ("man") Δ_{IMDB}	Q2_3	$\sigma_{\perp\in}$ ("teen", "party", "home") Δ_{IMDB}
Q1_4	$\sigma_{title\in}$ ("robot", "human") $\wedge$ *plot*$\in$ ("war", "world") Δ_{IMDB}	Q2_4	$\sigma_{\perp\in}$ ("west", "cowboy", "peacekeeper", "sheriff") Δ_{IMDB}
Q1_5	$\sigma_{title\in}$ ("mafia", "kill") $\wedge$ *plot*$\in$ ("mob", "hit") $\wedge$ $\perp\in$ ("family") Δ_{IMDB}	Q2_5	$\sigma_{\perp\in}$ ("trip", "road", "city", "group", "fun") Δ_{IMDB}

Group *Q3* *Expanded queries* with *varying number of attributes*		**Group *Q4*** *Expanded queries without attributes*[1]	
Q3_1	$\sigma_{title\in}$ ("auto") Δ_{IMDB}	Q4_1	$\sigma_{\perp\in}$ ("car") Δ_{IMDB}
Q3_2	$\sigma_{title\in}$ ("auto", "muscle") Δ_{IMDB}	Q4_2	$\sigma_{\perp\in}$ ("car", "explosion") Δ_{IMDB}
Q3_3	$\sigma_{title\in}$ ("auto", "muscle") $\wedge$ *plot*$\in$ ("classic") Δ_{IMDB}	Q4_3	$\sigma_{\perp\in}$ ("car", "explosion", "race") Δ_{IMDB}
Q3_4	$\sigma_{title\in}$ ("auto", "muscle") $\wedge$ *plot*$\in$ ("classic", "speed") Δ_{IMDB}	Q4_4	$\sigma_{\perp\in}$ ("car", "explosion", "race", "guns") Δ_{IMDB}
Q3_5	$\sigma_{title\in}$ ("auto","muscle") $\wedge$ *plot*$\in$ ("classic","speed") $\wedge$ $\perp\in$ ("thrills") Δ_{IMDB}	Q4_5	$\sigma_{\perp\in}$ ("car", "explosion", "race", "guns", "shootout") Δ_{IMDB}

Note that varying the number of attributes did not show any detectable impact on query execution time since, they almost do not affect the size of *SemIndex+* neither do they affect the complexity of query search algorithm (compared with the relatively huge numbers of data and index nodes involved).



a. Varying IMDB and WordNet chunk sizes, with fixed number of query terms k=5 and fixed link distance threshold ℓ = 5

b. Varying k and ℓ, with fixed IMDB and WordNet chunk sizes (=100% each)

Fig. 14. Query execution time, using *SemIndex+*'s *SI_PSS* algorithm, averaged over all queries.

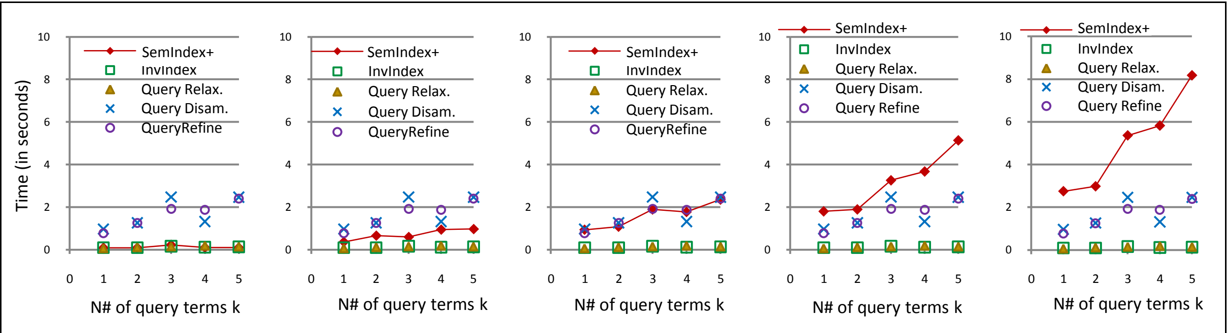
7.5.2. Comparing Query Processing Time with Alternative Solutions

We ran the same querying tasks using four alternative querying approaches adapted from the literature: *InvIndex* [62], *QueryRelax* [64], *QueryDisam* [7], and *QueryRefine* [67] (cf. Appendix III), and compared the obtained query time

[1] Recall that $\perp$ designates a data object's combined textual content from all attributes.

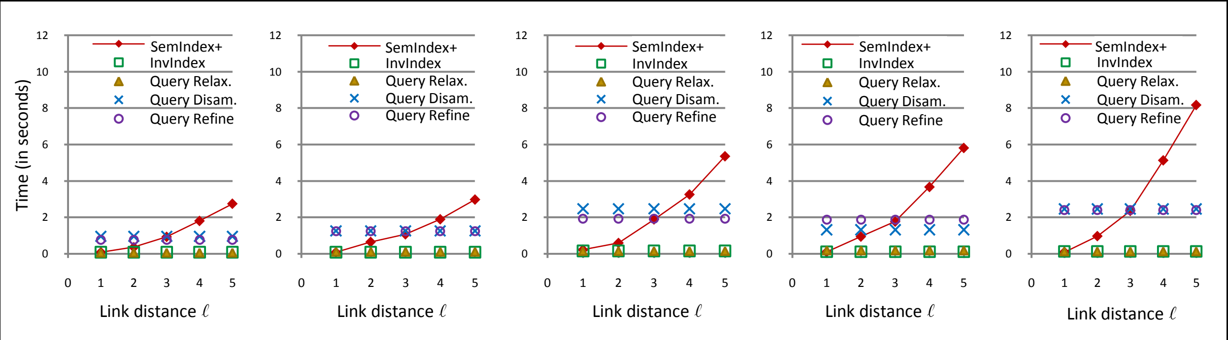
[2] Recall that we utilize the NoSQL version of IMDB *movies* to perform our experiments (cf. Section 7.2).

results with *SemIndex+*'s querying algorithm. Fig. 15 and Fig. 16 show average query execution time, plotted by varying the number of query terms *k* (in Fig. 15) and *SemIndex+* link distance threshold *ℓ* (in Fig. 16). First, results show that *SemIndex+* and its alternatives have very close query time levels when link distance *ℓ* is small (*ℓ*=1 and *ℓ*=2), such that *SemIndex+* time increases as link distance increases (whereas the other algorithms' query time is naturally invariant w.r.t. variations in *ℓ*, cf. Fig. 16). Second, Fig. 15 shows that query time for *SemIndex+* and all three alternative solutions (namely *QueryDisam* and *QueryRefine*) almost linearly increases with the number of query terms *k*[1], such that the pace of *SemIndex+*'s time increase is relatively low with small link distances (*ℓ*=1 and *ℓ*=2) and the pace augments when reaching higher link distance thresholds (*ℓ*=3-to-5). Third, one can realize that the most time consuming alternative approaches are *QueryDisam* and *QueryRefine*: i) *QueryDisam* and *QueryRefine* are more time consuming than *SemIndex+* when the latter is run with link distances ($ℓ \leq 3$), but they are both surpassed by *SemIndex+* when the latter is run with higher link distances (*ℓ*=4 and *ℓ*=5). Here, *QueryDisam*'s computational overhead is due to the complexity of the query keywords' semantic disambiguation process, whereas *QueryRefine*'s overhead[2] is due to post-processing the first round of query results in order to refine/rewrite query keywords accordingly. Algorithms *InvIndex* and *QueryRelax* share very close time levels and are the most efficient among the five solutions (including *SemIndex+*), running most queries almost instantaneously (under 0.14 seconds). This is expected since both approaches perform traditional syntactic query processing (and do not involve computationally expensive semantic processing) with one difference: *InvIndex* runs on the original query terms, whereas *QueryRelax* runs on an expansion of the original terms (adding the terms' synonymous words to the original keyword query).



a. Link distance *ℓ* = 1 **b.** Link distance *ℓ* = 2 **c.** Link distance *ℓ* = 3 **e.** Link distance *ℓ* = 4 **f.** Link distance *ℓ* = 5

Fig. 15. Comparing *SemIndex+* query time with four alternative solutions while varying the number of query terms *k* and fixing link distance *ℓ* (the latter affecting *SemIndex+* only).



a. N# of query terms *k* = 1 **b.** N# of query terms *k* = 2 **c.** N# of query terms *k* = 3 **d.** N# of query terms *k* = 4 **e.** N# of query terms *k* = 5

Fig. 16. Comparing *SemIndex+* query time with four alternative solutions while varying link distance threshold *ℓ* (affecting *SemIndex+*) and fixing the number of query terms *k*.

7.5.3. Discussion

By comparing *SemIndex+*'s query execution time with existing alternative solutions (*InvIndex*, *QueryRelax*, *QueryDisam*, and *QueryRefine*), we can highlight various observations: i) *SemIndex+* is more computationally

[1] Both *InvIndex* and *QueryRelax* time levels increase with *k*, even though these are not clearly visible in the graphs of Fig. 15 and Fig. 15 due to their scale. The reader can refer to the actual data behind the graphs at: http://sigappfr.acm.org/Projects/SemIndex/

[2] *QueryRefine*'s time shown in Fig. 16 and Fig. 17 does not encompass the time it took the testers to manually choose the new query terms (which we did not consider to be part of the algorithm itself), but only considers actual algorithm (CPU and SQL) execution time.

expensive than syntactic solutions such as *InvIndex* and *QueryRelax*, ii) Involving query disambiguation (*QueryDisam*) is clearly computationally expensive (due to the complexity of the word sense disambiguation process applied on the query keywords) and hinders query performance, iii) Involving query refinement (through *QueryRefine*, producing the first round of results, allowing the user to refine query keywords, and then producing the second round of results) adds computational overhead, iv) The time to navigate the semantic graph, following the allowed link distance ℓ, remains the foremost determining factor in *SemIndex+* query execution time. *SemIndex+* executes faster than *QueryDisam* and *QueryRefine* with low ℓ (≤ 3) but then requires more time than the latter when ℓ increases (to ℓ =4 and 5), v) *SemIndex+* search can benefit from parallelization, and can execute even faster when run on more powerful parallel processing systems (such as advanced multi-core, cluster, or grid computing platforms). We omit the latter experimental results here for ease of presentation and report the detailed evaluation of parallelization and its impact on *SemIndex+* to a later dedicated study.

7.6. Query Result Evaluation

7.6.1. Result Quality Evaluation Metrics

In addition to evaluating *SemIndex+*' efficiency (processing time), we also evaluated its effectiveness (result quality), i.e., evaluating the interestingness of semantic-aware answers from the user's perspective. To do so, we collected the results of our test queries and mapped them against user feedback (user judgments, utilized as *golden truth*) evaluating the quality of the answers produced by the system by computing *precision*, *recall*, *f-value*, and *mean average precision* metrics commonly utilized in IR evaluation [65]. Formally:

$$PR = \frac{a}{a+b} \in [0,1] \qquad R = \frac{a}{a+c} \in [0,1] \qquad F\text{-}Value = \frac{2 \times PR \times R}{PR + R} \in [0,1] \qquad \textbf{(10)}$$

where *a* is the number of retrieved data objects that indeed correspond to the query's result list (correctly retrieved), *b* is the number of retrieved data objects that do not correspond to the query's result list (wrongly retrieved), and *c* is the number of data objects that are not retrieved, although they correspond to the query's result list (data objects that should have been retrieved). *F-value* represents the harmonic mean of *precision* and *recall*, such that high *precision* and *recall*, and thus high *f-value* characterize good retrieval quality [65]. Also, we employed *mean average precision* (*MAP*) to evaluate the ranking of relevant results w.r.t. non-relevant ones in the query result list:

$$MAP = \frac{\sum_{j=1..n} (PR[j] \times rel[j])}{N} \in [0,1] \qquad \textbf{(11)}$$

where *n* be the number of hits (i.e., returned data objects) in the query result list, *PR*[*j*] is *precision* at hit *j*, *rel*[j] is equal to 1 if the *j*th data object in the result list is relevant and 0 otherwise, and *N* = *a*+*c* is the total number of data objects in the data collection which are relevant for the query. *MAP* is maximum, i.e., = 1, when the system retrieves all relevant data objects (i.e., *recall* = 1) and ranks them perfectly: all of them appearing before non-relevant data objects in the query list (i.e., *precision* at *N*th hit =1); and decreases as more non-relevant data objects are introduced before relevant ones in the result list.

Ten test subjects (six master students, and four doctoral students, who were not part of the system development team) were involved in the experiment as human judges. Testers were asked to evaluate the quality of the top 100 results (movie objects returned) per query (produced by *SemIndex+* and its 3 alternatives). Here, only queries consisting of two keywords or more were considered, given that 1 single term queries (e.g., *Qi_1*) were deemed two fuzzy and coarse-grained for the testers to judge their results[1]. Query results were randomized before being shown to testers. Manual relevance ratings (in the form of integers $\in$ {-1, 0, 1}, i.e., {*not relevant*, *neutral*, *relevant*}) were acquired for each query answer. Then, we quantified inter-tester agreement, by computing pair-wise correlation scores[2] among testers for each of the rated query answers, and subsequently selected the top 100 hundred answers per query having the highest average inter-tester correlation scores[3], which we utilized as the experiment's *golden truth*.

[1] A great many movies can be retrieved as answers for query *Q1_1*: $\sigma_{title \in (\text{"time"})} \Delta_{\text{IMDB}}$ given single term "*time*" alone is too broad for human testers to make sense of the query. The same goes for the other three single term queries: *Q2_1*, *Q3_1*, and *Q4_1* (Table 4).

[2] Using *Pearson Correlation Coefficient* (*PCC*), producing scores $\in$ [-1, 1] such that: -1 designates that one tester's ratings is a decreasing function of the other tester's ratings (i.e., answers deemed relevant by one tester are deemed irrelevant by the other, and vice versa), 1 designates that one tester's ratings is an increasing function of the other tester's ratings (i.e., answers are deemed relevant/irrelevant by testers alike), and 0 means that tester ratings are not correlated.

[3] Having average inter-tester *PCC* score $\geq$ 0.5.

7.6.2. Comparing SemIndex+ Result Quality with Alternative Solutions

Fig. 17 shows the *precision*, *recall*, *f-value*, and *MAP* results obtained with *SemIndex+*'s querying algorithm and alternative solutions: *InvIndex* [62], *QueryRelax* [64], *QueryDisam* [7], and *QueryRefine* [67]. Results averaged per link distance ℓ and number of query terms *k* are provided in Table 5. These highlight several observations.

1) ***Precision*** and ***recall*** with ***link distance***: One can realize that *precision* levels with *SemIndex+* while fluctuating, generally increase with link distance (ℓ) until reaching $\ell = 3$ or $\ell = 4$ where *precision* starts to decrease toward $\ell = 5$. However, one can realize that *recall* levels steadily increase with ℓ (with reduced fluctuation compared with *precision*). On the 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 $\ell > 1$). On the other hand, this also shows that over-navigating the *SemIndex+* graph to link terms with semantically related ones located as far as $\ell \geq 3$ 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 $\ell = 5$ 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.)[1]. Many such examples occurred when running multiple term queries such as *Q1_4* (consisting of terms "*robot*", "*human*", "*war*", "*world*")[2], where movies like *The Taking of Pelham One Two Three*[3] and *Showtime*[4] (among others) where returned as results by *SemIndex+*'s querying algorithm when reaching ℓ=5. Such results were deemed not relevant by the testers since they do not correspond to the semantics of the query.

2) ***Precision*** and ***recall*** with ***the number of query terms***: Here, one can realize that *precision* levels tend to stagnate or even decrease when increasing the number of terms *k* – with queries having low link distance thresholds ($\ell \leq$ 2-or-3) ; whereas *precision* tends to increase with the increase of *k* – with queries having higher link distances ($\ell \geq$ 3-or-4). A similar behavior can be seen with *recall* levels: using more query terms (increasing *k*) produces lesser results when link distance is low. For instance, running query *Q1_4* with link distance ℓ=1 requires the returned movie objects to contain exact occurrences of all 4 query terms: "*robot*", "*human*", "*war*" and "*world*", hence in our case producing zero (no) results whatsoever with all four *SemIndex+* algorithms. Yet, as link distance increases, more and more (semantically related) results are retrieved (i.e., 0, 5, 253, and 1363 results were produced by *SemIndex+* as answers for query *Q1_4* with ℓ=2, 3, 4, and 5 respectively). In other words, as ℓ increases, so do the chances of producing more results with (lower and especially) higher *k* values, which in turn improves both *precision* (retrieving more relevant results) and *recall* (missing out less relevant results).

3) Regarding ***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* as mentioned and discussed in the previous paragraph. 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.

4) Comparison with **alternative solutions**: First, considering all four metrics, one can realize that *SemIndex+* performs similarity to alternative solutions at lower link distances ($\ell \leq 2$), and then increasingly surpasses the latter as ℓ increases ($\ell \geq 3$). This emphasizes the central impact of link distance in improving *SemIndex+* performance (highlighted above). Second, considering results compiled over all link distances combined, one can clearly realize that *SemIndex+* surpasses its alternatives considering both *f-value* and *MAP*, i.e., in both result quality and ranking. *InvIndex* naturally produced the worst *f-value* and *MAP* results since it is not semantic aware and only returns exact (syntactic) matches to query terms.

[1] 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.

[2] Query *Q1_4*: $\sigma_{title \in (\text{“robot”, “human”}) \wedge plot \in (\text{“war”, “world”})} \Delta_{\text{IMDB}}$

[3] 2009 movie starring Denzel Washington and John Travolta, about a train hijacking in New York city.

[4] 2002 comedy movie starring Eddy Murphy and Robert De Niro, about police officers starring in a reality TV show.

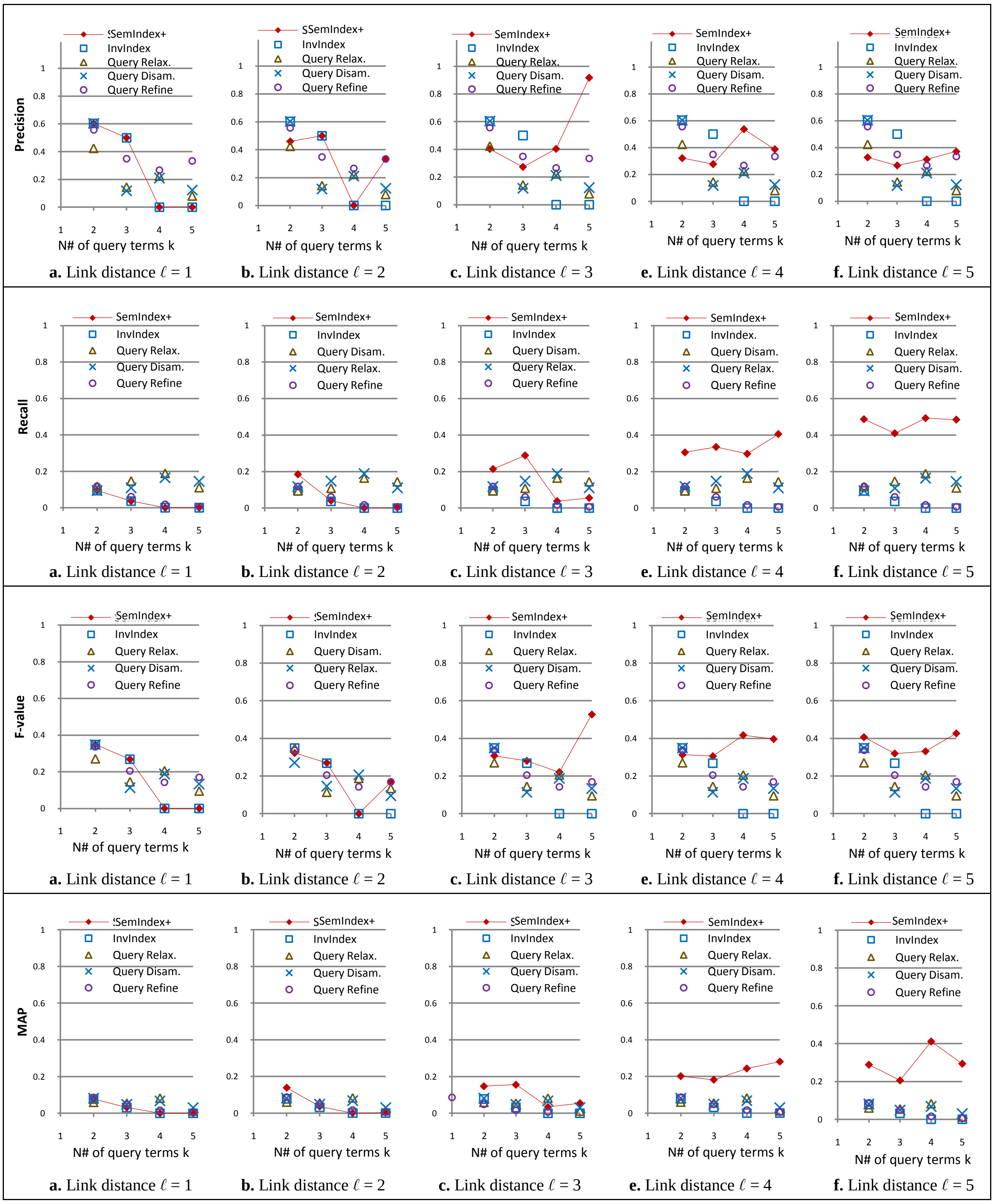


Fig. 17. Comparing *precision* (PR), *recall* (R), *f-value*, and *mean average precision* (*MAP*) results obtained using *SemIndex+*'s querying versus alternative solutions.

7.7. Discussion

To sum up, we evaluate in Table 6 the ratio (expressed in percentage) of increase in *query execution time*, as well as the ratio (percentage) of increase in *query result quality* (considering average *MAP* scores) when using *SemIndex+*'s querying algorithm versus alternative solutions (detailed results are provided in Appendix I):

$$\eta_{Efficiency}(SemIndex+, Alt) = \frac{QueryTime_{SemIndex+} - QueryTime_{Alt}}{QueryTime_{Alt}} \qquad \eta_{Effectiveness}(SemIndex+, Alt) = \frac{MAP_{SemIndex+} - MAP_{Alt}}{MAP_{Alt}} \quad \textbf{(12)}$$

Results in Table 6 highlight various observations: i) *SemIndex+* querying requires an average 1701.74%, 1622.48%, 26.99%, and 21.83% more processing time than *InvIndex*, *QueryRelax*, *QueryDisam*, and *QueryRefine* respectively; ii) *SemIndex+* improves query result quality levels by 543.67%, 472.64%, 238.31%, and 156.31% compared with *InvIndex*, *QueryRefine*, *QueryRelax*, and *QueryDisam* respectively; iii) *SemIndex+* is costlier in computation time compared with its alternatives, nonetheless, it is also clearly and significantly more effective in producing higher quality results; iv) *SemIndex+* can be most clearly appreciated when compared with *QueryDisam* where its improvement in query result quality (156.31%) clearly surpasses by 5.82 times its increased query execution time (26.88%), such that the time "effort" put in query execution time more than quintupled the system's increase in query result quality. Even more pronounced, *SemIndex+*'s (21.83%) increase in query time w.r.t. *QueryRefine* produced a significant 21.65 times increase in result quality (472.64%).

Table 5. Average *PR*, *R*, *f-value*, and *MAP* (Appendix I)

	PR	R	F-value	MAP
SemIndex+	0.3636	0.2085	0.2815	0.1393
InvIndex	0.2758	0.0327	0.1543	0.0273
QueryRelax	0.2179	0.1412	0.1796	0.0527
QueryDisam	0.2639	0.1281	0.1960	0.0570
QueryRefine	0.3762	0.0508	0.2135	0.0394

Table 6. Average $\eta_{Efficiency}$ and $\eta_{Effectiveness}$ (Appendix I)

	$\eta_{Efficiency}$×100	$\eta_{Effectiveness}$×100
InvIndex	1701.74%	543.67%
QueryRelax	1622.48%	238.31%
QueryDisam	26.88%	156.31%
QueryRefine	21.83%	472.64%

8. Conclusion

This paper introduces *SemIndex+*, a framework for semantic-aware DB indexing and querying of unstructured (free-text), structured (relational), and partly-structured (NoSQL) textual data. At the indexer level, *SemIndex+* creates a hybrid graph 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. The index is extended to handle varying multi-attribute data collections (using attribute-sensitive indexers), handling terms with missing semantic connections (i.e., *missing terms*), and introducing a model for weighting *SemIndex+* entries (i.e., the graph's nodes and edges). At the query processing level, the framework provides a parallelized (multithreaded) querying algorithm, coupled with a dedicated relevance scoring measure allowing to retrieve and rank relevant query answers. Our theoretical study and empirical evaluation highlight interesting observations: i) *SemIndex+*'s structure can be built in average linear time, and its size is of average linear space w.r.t. the sizes of the input data and knowledge sources used, ii) query processing time is linear in the size of the *SemIndex+* structure, and varies linearly w.r.t. to the number query terms (keywords) as well as the link distance threshold designating the breadth of the *SemIndex+* graph to be covered during querying, iii) following the chosen link distance threshold, *SemIndex+* is more or less costly in query processing time compared with alternative solutions (i.e., inverted index search, query relaxation, query disambiguation, and query refinement), nonetheless, iv) it is usually and significantly more effective in producing semantic-aware and higher quality results, such that *SemIndex+*'s improvement in query result quality clearly surpasses its increased query execution time.

We are currently exploring various techniques to improve *SemIndex+* query processing time. First, we have started testing the querying algorithm on a parallel processing platform (using the open source Apache Hadoop[1]). Preliminary results show a significant reduction in query execution time as the number of concurrent threads (multi-cores) increases toward matching the number of query terms and staring index nodes. We are also investigating DB index partitioning techniques (horizontal [84], vertical [1], and graph-based [48]) to distribute *SemIndex+* on multiple sites in order to allow more parallelization. We also plan to investigate query-driven optimization techniques, such as multi-term indexing [86] (i.e., associating more than one single term with every entry in the seed inverted index, (<*term1, term2, ..., termN*> *docIDs*[]) which could be useful to reduce processing overhead for intersecting the inverted lists of multi-term queries, as well as incremental query result fetching [75] (returning successive subsets of the partial results until the final result is "good" enough, such that the minimum number of responses or the minimum range satisfying the query are found). Extending *SemIndex+* to perform incremental result fetching could prove to be efficient by limiting the breadth and depth of the *SemIndex+* graph navigated by the querying algorithm.

Acknowledgements

This study is partly funded by the National Council for Scientific Research – Lebanon (CNRS-L), the Lebanese American University (LAU), the Fulbright Visiting Scholar program (sponsored by the US Department of State), and the Research Support Foundation of the State of Sao Paulo (FAPESP). We would also like to thank LAU graduates Christian Kallas and Marc Al Assad who helped implement different components of the *SemIndex+* prototype system.

[1] http://hadoop.apache.org/

References

[1] Abadi D.J. et al., *SW-Store: a vertically partitioned DBMS for Semantic Web data management.* VLDB Journal, 2009. 18(2): 385-406.

[2] Agarwal M.K., Ramamritham K., and Agarwal P., *Generic Keyword Search over XML Data.* International Conference on Extended DataBase Technology (EDBT'16), 2016. pp. 149-160.

[3] Agirre E. and Edmonds P., *Word Sense Disambiguation: Algorithms and Applications.* Dordrecht: ISBN 978-1-4020-4809-8, 2006.

[4] Agrawal S., et al., *Exploiting Web Search Engines to Search Structured Databases.* World Wide Web Conference (WWW'09), 2009. pp. 501-510.

[5] Algergawy A., Nayak R., and Saake G., *Element Similarity Measures in XML Schema Matching.* Elsevier Information Sciences, 2010. 180(24): 4975-4998

[6] Allan J., et al., *Frontiers, Challenges, and Opportunities for Information Retrieval* Report from SWIRL 2012 the Second Strategic Workshop on Information Retrieval - SIGIR Forum 2012. 46(1): 2-32.

[7] Allan J. and H. Raghavan, *Using Part-of-Speech Patterns to Reduce Query Ambiguity.* In 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2002. pp. 307-314, Tampere, Finland: ACM Press, New York.

[8] Andreasen T., et al., *Conceptual Indexing of Text Using Ontologies and Lexical Resources,* . Inter. Conf. on Flexible Query Answering Systems (FQAS'09) 2009. pp 323-332.

[9] Baeza-Yates R. and Ribeiro-Neto B., *Modern Information Retrieval: The Concepts and Technology behind Search.* ACM Press Books, Addison-Wesley Professional, 2nd Ed., 2011. p. 944.

[10] Bao Z., et al., *A Query Refinement Framework for XML Keyword Search.* World Wide Web 2017. 20(6):1469-1505.

[11] Bast H. and Buchhold B., *An Index for Efficient Semantic Full-Text Search.* Proceedings of the 22nd ACM International Conference on information & knowledge Management (CIKM '13), 2013. pp. 369-378

[12] Baziz M., et al., *An Information Retrieval Driven by Ontology: from Query to Document Expansion.* Large Scale Semantic Access to Content: Text, Image, Video, and Sound (RIAO'07), 2007. pp. 301-313.

[13] Bednar Peter, Sarnovsky M., and Demko V., *RDF vs. NoSQL databases for the Semantic Web applications.* IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI'14), 2014. pp. 361-364.

[14] Bergamaschi S., et al., *Combining User and Database Perspective for Solving Keyword Queries over Relational Databases.* Information Systems, 2016. 55: 1-19.

[15] Blanco R., Mika P., and Vigna S., *Effective and Efficient Entity Search in RDF data.* In International Semantic Web Conference (ISWC'11), 2011. pp. 83–97.

[16] Budanitsky A. and Hirst G., *Evaluating WordNet-based Measures of Lexical Semantic Relatedness.* Computational Linguistics, 2006. 32(1): 13-47.

[17] Burton-Jones A., et al., *A Heuristic-Based Methodology for Semantic Augmentation of User Queries on the Web.* In Proceedings ot the International Conference on Conceptual Modeling (ER'03), 2003. pp. 476–489.

[18] Calvo H., Gelbukh A.F., and Kilgarriff A., *Distributional Thesaurus Versus WordNet: A Comparison of Backoff Techniques for Unsupervised PP Attachment.* Conf. on Computational Linguistics and Natural Language Processing (CICLing'05) 2005. pp. 177-188.

[19] Carpineto C. and Romano G., *A Survey of Automatic Query Expansion in Information Retrieval,* . ACM Computing Survey, ACM New York, NY, USA, 2012. 44(1):1.

[20] Carpineto C., Romano G., and Giannini V., *Improving Retrieval Feedback with Multiple Term-Ranking Function Combination.* ACM Transactions on Information Systems (TOIS), 2002. 20(3):259-290

[21] Chandramouli K., et al., *Query Refinement and user Relevance Feedback for contextualized image retrieval.* 5th International Conference on Visual Information Engineering (VIE), 2008. pp. 453 - 458.

[22] Charbel N., et al., *Resolving XML Semantic Ambiguity.* International Conference on Extending Database Technology (EDBT'15), 2015. Brussels, Belgium, pp 277-288.

[23] Chbeir R., et al., *SemIndex: Semantic-Aware Inverted Index.* 18th East-European Conference on Advanced Databases and Information Systems (ADBIS'14), 2014. pp. 290-307.

[24] Chen L.J. and Papakonstantinou Y., *Supporting top-K keyword Search in XML Databases.* International Conference on Data Engineering (ICDE'10), 2010. pp. 689-700.

[25] Cheng T., Yan X., and Chang K. C., *EntityRank: searching entities directly and holistically.* Proceedings of the 33rd international conference on Very Large Data Bases (VLDB'07), 2007. pp. 387-398.

[26] Chu E., et al., *A relational approach to incrementally extracting and querying structure in unstructured data.* Proceedings of the 33rd international conference on Very Large Data Bases (VLDB '07), 2007. pp. 1045-1056

[27] Cimiano P., Handschuh S., and Staab S., *Towards the Self-Annotating Web.* In Proceedings of the International World Wide Web Conference (WWW'04), 2004. pp. 462-471.

[28] Cormen T.H., et al., *Introduction to Algorithms (3rd Ed.).* MIT Press and McGraw-Hill. , 2009.

[29] Cui H., et al., *Probabilistic Query Expansion Using Query Logs.* In Proceedings of the 11th International World Wide Web Conference (WWW'02), 2002. Honolulu, Hawaii, pp. 325-332.

[30] Das S., e.a., *Making unstructured data sparql using semantic indexing in oracle database.* In Proceedings of 29th IEEE ICDE Conf., 2012. pp. 1405–1416

[31] Davies M., *The Corpus of Contemporary American English as the first reliable monitor corpus of English.* Literary & Linguistic Computing, 2010. 25(4): 447-464.

[32] de Lima E.F. and Pedersen J.O., *Phrase Recognition and Expansion for Short, Precision biased Queries based on a Query Log.* In International ACM SIGIR Conference on Research and Development in Information Retrieval, 1999. pp. 145-152, Berkeley, CA.

[33] DeCandia G. et al., *Dynamo: Amazon's Highly Available Key-Value Store.* Proc. ACM SIGOPS Symp. Operating Systems Principles (SOSP 07), vol. 41, ACM Press, 2007. pp. 205–220.

[34] Devitt A. and Vogel C., *The Topology of WordNet: Some Metrics.* Proceedings of the Second Global Wordnet Conference (GWC), 2004. pp. 106-111.

[35] Ding B., et al., *Finding top-k min-cost connected trees in databases.* International Conference on Data Engineering (ICDE'07), 2007.

[36] Duchet P., *Hypergraphs.* In Graham R., Grotschel M., Lovasz L., eds.: Handbook of Combinatorics, 1995. Elsevier Science B.V., Amsterdam, pp. 381-432.

[37] Egozi O., Markovitch S., and Gabrilovich E., *Concept-Based Information Retrieval Using Explicit Semantic Analysis.* ACM Transactions on Information Systems 2011. 29(2):8.

[38] Francis W. N. and Kucera H., *Frequency Analysis of English Usage.* Houghton Mifflin, Boston, 1982.

[39] Furnas G., et al., *The vocabulary problem in human-system communication.* . Communications of the ACM, 1987. 30(11):964–971.
[40] Gao X. and Qiu J., *Supporting Queries and Analyses of Large-Scale Social Media Data with Customizable and Scalable Indexing Techniques over NoSQL Databases.* IEEE/ACM Inter. Symposium on Cluster Computing and the Grid (CCGRID'14), 2014, 587-590.
[41] Gauch S., Ravindran D., and Chandramouli A., *KeyConcept: Conceptual Search and Pruning Exploiting Concept Relationships.* Journal of Intelligent Systems, 2010. 19(3): 265-288
[42] Giunchiglia F., Kharkevich U., and Zaihrayeu I., *Concept Search.* In ESWC - Semantics and Big Data, 2009. pp. 429–444.
[43] Greenberg J., *Automatic Query Expansion via Lexical-Semantic Relationships.* Journal of the American Society for Information Science, 2001. 52(5):402–415.
[44] Grootjen F. and Van Der Weide T.P., *Conceptual query expansion.* Data Knowledge Engineering, 2006. 56:174–193.
[45] Hoffart J., et al., *YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia.* Artif. Intell., 2013. 194: 28-61.
[46] Hopfield J. and Tank D., *Neural Computation of Decisions in Optimization Problems.* . Biological Cybernetics, 1985. 52(3):52–141.
[47] Hristidis V., Gravano L., and Papakonstantinou Y., *Efficient IR-style keyword search over relational databases.* Proceedings of the International Conference on Very Large Data Bases (VLDB'03), 2003. pp. 850-861.
[48] Huang J. and Abadi D., *LEOPARD: Lightweight Edge-Oriented Partitioning and Replication for Dynamic Graphs.* PVLDB, 2016. 9(7):540-551.
[49] Huawei-Hadoop, *HIndex.* https://github.com/Huawei-Hadoop/hindex, 2014. Accessed June 2018.
[50] Hudec M., *An approach to fuzzy database querying, analysis and realization.* Comput. Sci. Inf. Syst., 2009. 6(2): 127-140.
[51] Jones R., et al., *Generating Query Substitutions.* Inter. Conf. on World Wide Web, WWW '06, 2006. pp. 387–396, New York, ACM.
[52] Kamvar M. and Baluja S., *A Large Scale Study of Wireless Search Behavior: Google Mobile Search.* In Proceedings of the SIGCHI Conference on Computer Human Interaction, 2006. pp. 701–709, Montreal, Canada.
[53] Kathuria A., et al., *Classifying the User Intent of Web Queries using K-means Clustering.* Internet Research, 2010. 20(5): 563-581.
[54] Kumar S., Rana R.K., and Singh P., *Ontology based Semantic Indexing Approach for Information Retrieval System.* International Journal of Computer Applications, 2012. Volume 49– No.12.
[55] L'Hadj L.S., Boughanem M., and Amrouche K., *Enhancing Information Retrieval through Concept-based Language Modeling and Semantic Smoothing.* Journal of the Association for Information Science and Technology (JASIST), 2016. 67(12): 2909-2927.
[56] Lester N., Zobel J., and Williams H., *Efficient online index maintenance for contiguous inverted lists.* Information Processing and Management, 2006. 42(4):916- 933.
[57] Li F. and J. H.V., *Constructing an Interactive Natural Language Interface for Relational Databases.* Proceedings of the VLDB Endowment, 2014. pp. 73-84.
[58] Li Y., Yang H., and Jagadish H.V., *Term Disambiguation in Natural Language Query for XML.* In Proceedings of the International Conference on Flexible Query Answering Systems (FQAS), 2006. LNAI 4027, pp. 133–146.
[59] Liu Y., et al., *Using WordNet to Disambiguate Word Senses for Text Classification.* International Conference on Computational Science (ICCS'07), 2007. pp 781-789.
[60] Luo Y., et al., *Spark: top-k keyword query in relational databases.* Proceedings of the 2007 ACM International Conference on Management of Data (SIGMOD-07), 2007. pp. 115-126.
[61] Mahapatra A.K. and Biswas S., *Inverted Index: Types and techniques.* International journal of Computer science Issues,, 2011. 8(4):1.
[62] Manning C.D., Raghavan P., and Schütze H., *Introduction to Information Retrieval.* Cambridge University Press, 2008. Ch. 1. Boolean Retrieval - A First Take at Building an Inverted Index, https://nlp.stanford.edu/IR-book/
[63] Markowetz A. and Yang Y., *Keyword search on relational data streams.* International Conference on Management of Data (SIGMOD'07), 2007. pp. 605–616.
[64] Martinenghi D. and Torlone R., *Taxonomy-based relaxation of query answering in relational databases.* VLDB Journal, 2014. 23(5):747-769.
[65] McGill M., *Introduction to Modern Information Retrieval.* 1983. McGraw-Hill, New York.
[66] Miller G.A. and Fellbaum C., *WordNet Then and Now.* Language Resources and Evaluation, 2007. 41(2): 209-214.
[67] Mishra C. and Koudas N., *Interactive Query Refinement* Inter. Conf. on Extending Database Technology (EDBT'09), 2009, 862-873.
[68] Mitra M., Singhal A., and Buckley C., *Improving Automatic Query Expansion.* In the in 21st International Conference on Research & Development on Information Retrieval, 1998. Melbourne, Australia, pp. 206-214.
[69] Navigli R., *Word Sense Disambiguation: a Survey.* ACM Computing Surveys, 2009. 41(2):1–69.
[70] Navigli R., et al., *Extending and Enriching WordNet with OntoLearn.* Proc. of The Second Global Wordnet Conference 2004 (GWC'04), 2004. pp. 279–284.
[71] Navigli R. and Crisafulli G., *Inducing Word Senses to Improve Web Search Result Clustering.* In Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, 2010. pp. 116–126, MIT, USA.
[72] Navigli R. and Velardi P., *An Analysis of Ontology-based Query Expansion Strategies.* In proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI'03), 2003. pp. 42-49.
[73] Nihalani N., Silakari S., and Motwani M., *Natural language Interface for Database: A Brief review.* International Journal of Computer Science Issues, 2011. 8(2):600-608.
[74] Philippe C.-M. et al., *NoSQL Databases for RDF: An Empirical Evaluation.* Inter. Semantic Web Conf. (ISWC'13), 2013. pp 310-325.
[75] Reynolds P. and Vahdat A., *Peer-to-Peer Keyword Search: A Retrospective.* Middleware, 2013. pp. 485-496.
[76] Salton G. and C. Buckley, *Improving Retrieval Performance by Relevance Feedback.* Journal of the American Society for Information Science, 1990. 41(4):288–297.
[77] Schoefeggera K., et al., *A survey on socio-semantic information retrieval.* Computer Science Review, 2013. 8:25–46.
[78] Schuetze H. and Pedersen J. O., *Information Retrieval based on Word Senses.* In Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval. , 1995. pp. 161–175.
[79] Seo C., et al., *An efficient inverted index technique for xml documents using RDBMS.* Information & Software Technology, 2003. 45(1):11-22.
[80] Sinh Hoa Nguyen, 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).
[81] Tekli J., *An Overview on XML Semantic Disambiguation from Unstructured Text to Semi-Structured Data: Background, Applications, and Ongoing Challenges.* IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE), 2016. 28(6): 1383-1407.
[82] Tekli J., et al., *Building Semantic Trees from XML Documents.* Elsevier Journal of Web Semantics (JWS): Science, Services and Agents on the World Wide Web, 2016. 37–38:1–24.
[83] Tekli J., et al., *Minimizing User Effort in XML Grammar Matching.* Elsevier Information Sciences Journal, 2012. 210:1-40.
[84] Thomson A. et al., *Calvin: fast distributed transactions for partitioned database systems.* Inter. ACM SIGMOD Conf., 2012. pp. 1-12.

[85] Velardi P., et al., *OntoLearn Reloaded: A Graph-Based Algorithm for Taxonomy Induction.* Computational Linguistics, 2013. 39(3): 665-707.
[86] von der Weth C. and Datta A., *Multiterm Keyword Search in NoSQL Systems.* IEEE Internet Computing, 2012. 16(1):34-42
[87] Weeds J., et al., *Characterising measures of lexical distributional similarity.* Int. Conf. on Computational Linguistics (COLING '04), 2004. Article No. 1015.
[88] Wen H., Huang G.S., and L. Z., *Clustering web search results using semantic information* International Conference on Machine Learning and Cybernetics, 2009. 3(1504 - 1509).
[89] Wu P., et al., *Towards keyword-driven analytical processing.* Inter. Conf. on Management of Data (SIGMOD'07), 2007. pp. 617–628.
[90] Yaworsky D., *Word-Sense Disambiguation Using Statistical Models of Roget's Categories Trained on Large Corpora.* Proceedings of the International Conference on Computational Linguistics (Coling), 1992. Vol 2, pp. 454-460. Nantes.
[91] Zhang P., *A Study on Database Fuzzy Query Method in SQL.* Inter. Conf. on Advances in Engineering, 2011. Vol. 24, pp. 340-344.
[92] Zhong S., et al., *A Design of the Inverted Index Based on Web Document Comprehending.* Journal of Computers, 2011. 6(4):664-670.

Appendix I: Experimental Test Data Characteristics and Results

Table 7. Characteristics of IMDB *movies* table chunks.

Chunk %	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Size (in MBs)	7.6183	15.0223	22.1164	29.7024	37.0124	43.8552	52.0411	58.5117	66.7324	74.1111
N# of data objects	14,304	28,608	42,912	57,217	71,521	85,825	100,130	114,434	128,738	143,043
N# of Attributes[1]	70,511	124,139	196,844	204,914	296,858	377,915	446,181	520,983	594,267	671,946
N# of Terms	638,459	1,209,423	1,967,252	2,534,272	3,199,881	4,014,704	4,723,916	5,337,436	6,225,128	7,046,035
Size (in MBs) **of *InvIndex***	24.0366	48.193	70.9162	92.3598	115.9013	138.9912	160.0991	189.8099	214.5981	237.6099

Table 8. Characteristics of WordNet chunks.

Chunk %	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Size (in MBs)	2.7707	3.9466	7.6498	9.5691	12.1641	13.8941	18.2191	19.9491	23.4091	26.0041
N# of Senses (Synsets)	11,738	23,475	35,212	46,949	58,686	70,423	82,160	93,897	105,634	117,371
Avg. Branch[2]	1.4533	1.6257	1.7553	1.9236	2.0697	2.2259	2.3736	2.5285	2.6677	2.8223
Size (in MBs) **of *InvIndex***	3.2031	4.5625	8.8437	11.0625	14.0625	16.0625	21.0625	23.0625	27.0625	30.0625

Table 9. Characteristics of *SemIndex+* chunks.

Chunk %	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Size (in MBs)	36.9219	68.2188	100.2813	133.3281	158.3594	202.4063	237.4688	273.5156	306.5938	339.625
N# of Data Nodes	14304	28608	42912	57217	71521	85825	100130	114434	128738	143043
N# of Attribute Nodes[2]	5	5	5	5	5	5	5	5	5	5
N# of Attribute Node Occurrences	70,511	124,139	196,844	204,914	296,858	377,915	446,181	520,983	594,267	671,946
N# of (Matching) **Index Term Nodes**	19090	36396	52388	67511	82370	96231	108828	122119	134258	146625
N# of (Missing) **Index Terms Nodes**	54165	79174	101594	121078	141534	158663	174111	186930	195897	210279
N# of Sense Nodes (Synsets)	11738	23475	35212	46949	58686	70423	82160	93897	105634	117371
Total N# of Nodes	99302	167658	232111	292760	354116	411147	465234	517385	564532	617323
Avg. Branch	1.7746	4.2493	5.8399	7.5746	8.6564	9.2882	9.9577	10.575	10.9745	11.3173

Table 10. Characteristics of *InvIndex* (w.r.t. IMDB) chunks.

Chunk %	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Size (in MBs)	25.5781	49.6250	73.6719	98.7188	122.7656	147.8125	171.8594	195.8906	220.9375	244.9844
N# of Data Objects	14304	28608	42912	57217	71521	85825	100130	114434	128738	143043
N# of Index Terms	73255	115570	153982	188589	223904	254894	282939	309049	330155	356904

[1] Number of attribute occurrences in all data objects of the IMDB *movies* dataset.

[2] The number of attribute nodes in our current *SemIndex+* graph is equal to 5 (denoting *title*, *year*, *plot*, *genre* and *info)* regardless of chunk size, since attribute nodes are added once to the index (cf. ER model in Fig. 10) following their first occurrence in the dataset, and are then referenced as many times as needed to construct 3-uniform data edges (corresponding to the number of attribute node occurrences).

Table 11. *Precision*, *recall*, *f-value*, and *MAP* results obtained with *SemIndex+* querying versus alternative solutions, averaged per link distance (ℓ) and per number of terms (*k*).

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

	ℓ=1	ℓ=2	ℓ=3	ℓ=4	ℓ=5	*k*=2	*k*=3	*k*=4	*k*=5	Avg.
SemIndex+	0.2758	0.3234	0.5193	0.3805	0.3189	0.4226	0.3632	0.2502	0.4184	**0.3636**
InvIndex	0.2758 (invariant with ℓ)					0.6032	0.5000	0.0000	0.0000	**0.2758**
QueryRelax	0.2179 (invariant with ℓ)					0.4243	0.1440	0.2233	0.0800	**0.2179**
QueryDisam	0.2639 (invariant with ℓ)					0.6032	0.1183	0.2108	0.1233	**0.2639**
QueryRefine	0.3762 (invariant with ℓ)					0.5562	0.3485	0.2667	0.3333	**0.3762**

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

SemIndex+	0.0327	0.0570	0.1487	0.3358	0.4684	0.2570	0.2219	0.1656	0.1895	**0.2085**
InvIndex	0.0327 (invariant with ℓ)					0.0943	0.0365	0.0000	0.0000	**0.0327**
QueryRelax	0.1412 (invariant with ℓ)					0.1178	0.1474	0.1889	0.1108	**0.1412**
QueryDisam	0.1281 (invariant with ℓ)					0.0943	0.1089	0.1644	0.1447	**0.1281**
QueryRefine	0.0508 (invariant with ℓ)					0.0943	0.0365	0.0000	0.0000	**0.0508**

c. Average *f-value* results

SemIndex+	0.1543	0.1902	0.3340	0.3581	0.3711	0.3398	0.2888	0.1937	0.3038	**0.2815**
InvIndex	0.1543 (invariant with ℓ)					0.3488	0.2683	0.0000	0.0000	**0.1543**
QueryRelax	0.1796 (invariant with ℓ)					0.2711	0.1457	0.2061	0.0954	**0.1796**
QueryDisam	0.1960 (invariant with ℓ)					0.3488	0.1136	0.1876	0.1340	**0.1960**
QueryRefine	0.2135 (invariant with ℓ)					0.3488	0.2683	0.0000	0.0000	**0.2135**

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

SemIndex+	0.0273	0.0443	0.0982	0.2264	0.3002	0.1708	0.1222	0.1377	0.1264	**0.1393**
InvIndex	0.0273 (invariant with ℓ)					0.0776	0.0317	0.0000	0.0000	**0.0273**
QueryRelax	0.0527 (invariant with ℓ)					0.0599	0.0558	0.0829	0.0120	**0.0527**
QueryDisam	0.0570 (invariant with ℓ)					0.0814	0.0814	0.0814	0.0814	**0.0570**
QueryRefine	0.0394 (invariant with ℓ)					0.0872	0.0481	0.0162	0.0061	**0.0394**

Table 12. Percentages of increase in *query execution time* and *query result quality*, when using *SemIndex+* versus alternatives solutions.

a. Percentage of increase in *query execution time* ($\eta_{Efficiency} \times 100$)

	ℓ=1	ℓ=2	ℓ=3	ℓ=4	ℓ=5	*k*=2	*k*=3	*k*=4	*k*=5	Avg.
InvIndex	0.59%	503.50%	1273.76%	2600.51%	4195.00%	1231.06%	1393.27%	1321.47%	2020.51%	1701.74%
QueryRelax	-5.90%[1]	464.52%	1185.03%	2426.09%	3917.60%	2032.25%	1127.12%	1480.63%	1286.97%	1622.48%
QueryDisam	-93.07%	-58.42%	-5.34%	86.07%	195.94%	23.07%	6.19%	-8.31%	86.81%	26.88%
QueryRefine	-93.24%	-59.45%	-7.69%	81.45%	188.59%	35.48%	0.08%	5.60%	31.89%	21.83%

b. Percentage of increase in *query result quality* ($\eta_{Effectiveness} \times 100$)

InvIndex	0.00%	61.99%	258.95%	727.99%	997.79%	119.96%	285.04%	1276.85%	1164.46%	543.67%
QueryRelax	-48.08%	-15.90%	86.36%	329.88%	469.95%	184.92%	119.06%	66.02%	952.53%	238.31%
QueryDisam	-52.00%	-22.24%	72.30%	297.44%	426.95%	109.80%	151.04%	103.40%	320.13%	156.31%
QueryRefine	-30.58%	12.46%	149.19%	474.80%	662.09%	95.81%	154.21%	749.38%	1986.36%	472.64%

Appendix II: Missing Terms Linkage Algorithm

Connecting unmapped *searchable term* nodes from $\tilde{G}_{\Delta}.V_i^+$ to $\tilde{G}_{KB}.V_i^+$, which we identify as *missing terms* in $\tilde{G}_{SI}$, can be handled using an adaptation of distributional thesauri construction methods, e.g., [18, 87], to allow mining the syntactic/lexical relatedness between the *missing terms* and the *index terms* in $\tilde{G}_{SI}$. Note that a distributional thesaurus is a thesaurus generated automatically from a given textual corpus (such as the Brown corpus [38], COCA [31], or even the textual collection Δ being indexed), by finding words that co-occur together or that have similar contexts in the corpus. To that end, we introduce the *MissingTerms_Linkage* algorithm in Fig. 18. It accepts as input: the *SemIndex+* graph $\tilde{G}_{SI}$, a reference text corpus *C*, as well as two input parameters: c_1 and c_2 designating respectively the co-occurrence *window size* and the number of *top-ranked terms* needed to identify related terms. For each missing term t_i in $\tilde{G}_{SI}$ (cf. Fig. 18.b, line 1), the algorithm creates a *relatedness vector* $RV(t_i)$ (line 3) to store the co-occurrence

[1] A negative *percentage of increase* underlines a *decrease percentage*.

frequencies of surrounding terms. It identifies a window of size c_1, consisting of c_1 terms occurring to the left and right of the missing term in the reference corpus and which also exist among the index terms of $\widetilde{G}_{SI}$ (line 4), and adds all window term frequencies to the *relatedness vector* (line 5). For example, suppose "*horror*" is a missing term, i.e., it does not appear in the WordNet lexicon extract but appears in object O_1 of the data collection (cf. Fig. 5). If window size c_1 = 2, using the data collection itself Δ_{Part} as reference corpus, then terms "*strange*", "*car*", "*thriller*" and "*cell*" would be in the surrounding window of "*horror*", and hence the relatedness score between "*horror*" and all these terms is increased. Once the vector has been obtained, we normalize vector scores w.r.t. overall maximum term co-occurrence frequency (line 6), and identify the c_2 top-ranked terms of the missing term t_i, which are considered as the most related terms to t_i in $\widetilde{G}_{SI}$ (line 7). Then, a link is created to connect t_i's term node with each top-ranked term t_k node in $\widetilde{G}_{SI}$. These links are represented as index edges in $\widetilde{G}_{SI}.E_i$ labeled: *occurs-with* (cf. Fig. 7 where term "*horror*" links with "*car*", considered as its most related (top-ranked, i.e., highest co-occurrence frequency) term[1]).

```
Algorithm MissingTerms_Linkage

Input:  $\widetilde{G}_{SI}$      // SemIndex+ graph
        C        // Reference text corpus
        $c_1$, $c_2$   // Input parameters: window size and top-ranked terms

Ouput:  $\widetilde{G}_{SI}$     // SemIndex+ graph with missing term links

Begin

For each missing term $t_i$ in $\widetilde{G}_{SI}$                                        1
{                                                                              2
    Create RV($t_i$) from C given $\widetilde{G}_{SI}$    // Relatedness vector for term $t_i$   3
    For each term $t_j$ in window($t_i$, $c_1$, C)                                    4
    { Add Freq($t_j$) to RV($t_i$) }                                              5

    RV($t_i$) = RV($t_i$) / Max(RV($t_i$))        // Normalizing RV($t_i$) scores         6

    $T_i$ = set of $c_2$ top-ranked terms in RV($t_i$)                                 7

    For each term $t_k$ in $T_i$                                                      8
    {  Create link between term nodes $t_i$ and $t_k$ in $\widetilde{G}_{SI}$                    9
       Label the link "occurs-with" }                                           10
    }                                                                          11
    Return $\widetilde{G}_{SI}$                                                           12
End
```

Fig. 18. Pseudocode of the *MissingTerms_Linkage* algorithm.

Note that the effectiveness of algorithm *MissingTerms_Linkage* depends on the number of missing terms, which in turn depends on the semantic coverage and expressiveness of the knowledge base used and its relatedness with the input textual collection (e.g., using a *medical* knowledge base to semantically map terms in a textual collection describing *sports events* will obviously lead to a substantial number of *missing terms* in the resulting *SemIndex+* graph, thus negatively affecting index construction performance). The algorithm's impact on *SemIndex+* querying effectiveness and efficiency will be evaluated in a dedicated future study.

Appendix III: Alternative Algorithms used in our Experimental Study

1. Legacy Inverted Index Search (*InvIndex*)

It's a standard containment keyword-based query [62] that:

1- Retrieves textual identities that contain a set of keywords,
2- Queries the inverted (*term*, *objectIDs*[]) list with every term in the query, to identify objects IDs associated to all (or at least one) query terms (based on the query type at hand: conjunctive or disjunctive),
3- Assigns a score to every potential query answer (data object) considering a predefined relevance ordering scheme, in order to return the results ranked by their order of scores in ascending order.

[1] A *missing term* can link with more than one (top-ranked) related terms, if more than one related terms were ranked with the same maximum co-occurrence frequency with the *missing term*.

2. Query Relaxation (*QueryRelax*)

It consists in expanding the user query to include more interesting (semantically related) words [64], which would help identify more interesting results. The main steps of the algorithm can be described as follows:

1- Perform Part-Of-Speech tagging,
2- Select the most common sense for each token (e.g., based on WordNet's usage frequency, computed based on the Brown text corpus),
3- For each selected sense s_i, include in the query: synonymous terms in the synset that is s_i, as well as the synonymous terms of all senses included in the direct semantic context of s_i [82], i.e., senses that are related to s_i via a semantic relationship (e.g., hypernymy, hyponymy, meronymy, etc.),
4- Run the resulting query using *InvIndex*, as a traditional keyword containment query on the data collection's inverted index (syntactic processing), and return the results to the user.

3. Query Disambiguation (*QueryDisam*)

It consists in applying word sense disambiguation [81] on query keywords, associating every keyword with its proper meaning (i.e., synset in WordNet) in order to execute the query accordingly [7]. The main steps of the algorithm can be described as follows:

1- Perform WSD on query terms using the simplified Lesk algorithm [3],
2- For each of the identified senses, include in the query: the sense's synonymous terms,
3- Run the resulting query using *InvIndex*, and return the results to the user

4. Query Refinement (*QueryRefine*)

It consists in refining the user query to remove certain terms and include more (semantically) descriptive terms, which would help identify more interesting results [67]. The main steps of the algorithm can be described as follows:

1- Run the *InvIndex* algorithm on the original query, and return the query results to the user,
2- For every term of the original query, provide the user with alternative suggestions in the form of a list of semantically related terms: the synonyms of all possible senses (synsets) of the query term,
3- Allow the user to add and remove terms to/from the original query, considering system provided suggestions (step 2) or her own choice of terms,
4- Run the refined query using *InvIndex*, and return the results to the user