SemSearch 2008, CEUR Workshop Proceedings, ISSN 1613-0073, online at CEUR-WS.org/Vol-334/

Concept Search: Semantics Enabled Syntactic Search

Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu

Department of Information Engineering and Computer Science
University of Trento, Italy
{fausto,kharkevi,ilya}@disi.unitn.it

Abstract. Historically, information retrieval (IR) has followed two principally different paths that we call syntactic IR and semantic IR. In syntactic IR, terms are represented as arbitrary sequences of characters and IR is performed through the computation of string similarity. In semantic IR, instead, terms are represented as concepts and IR is performed through the computation of semantic relatedness between concepts. Semantic IR, in general, demonstrates lower recall and higher precision than syntactic IR. However, so far the latter has definitely been the winner in practical applications. In this paper we present a novel approach which allows it to extend syntactic IR with semantics, thus leverage the advantages of both syntactic and semantic IR. First experimental results, reported in the paper, show that the combined approach performs at least as good as syntactic IR, often improving results where semantics can be exploited.

1 Introduction

The goal of information retrieval (IR) is to map a natural language query, which specifies the user information needs, to a set of objects in a given collection, which meet these needs. Most existing systems also compute a numeric score on how relevant each retrieved object is to the query, and order these objects according to the degree of relevance.

Historically, there has been two major approaches to IR that we calls syntactic IR and semantic IR. In syntactic IR, search engines use words or multi-words phrases that occur in documents and queries as atomic elements in document and query representations. The search procedure, used by these search engines, is principally based on the *syntactic matching* of document and query representations. These search engines are known to suffer in general from low precision while being good at recall.

Semantic IR is based on fetching document and query representations through semantic analysis of their contents using natural language processing techniques and then retrieving documents by matching these semantic representations. Differently from syntactic IR, in this approach the *meaning* of words is analyzed and not only their syntactic representations. Semantics-based approaches, in general, allow to reach a higher precision but lower recall than syntactic approaches [11].

In practice, results of semantic IR are inferior to that of syntactic one. In fact, most of the state of the art search engines are based on syntactic IR. There are many reasons for this, where one of them is that techniques based on semantics, to be used properly, need a lot of background knowledge which is in general not available [6].

In this paper we propose a novel approach to IR which extends syntactic IR with semantics, thus addressing the problem of low precision of syntactic IR. We call it *Concept Search* (*C-Search* in short). The main idea is to keep the same machinery which has made syntactic IR so successful, but to modify it so that, whenever possible, syntactic IR is substituted by semantic search, thus improving the system performance. This is why we say that *C-Search* is semantics enabled syntactic search. In principle, our approach allows it to scale on the continuum from purely syntactic search to purely semantic search, performing at least as well as syntactic search and improving over it by taking advantage of semantics when and where possible. Our approach scales as much as syntactic IR can scale because semantics is seamlessly integrated in the syntactic search technology.

The remainder of the paper is organized as follows. In Section 2, we first discuss IR in general and then we discuss syntactic search approach to IR. In Section 3, we discuss semantic IR and introduce semantics enabled syntactic search. In Section 4, we describe how semantic matching of (complex) concepts, the core of semantic search algorithm, can be efficiently implemented using inverted index technology. Section 5 presents some preliminary experimental results. In Section 6, we discuss the state-of-the-art in semantic search and compare our approach with other related approaches. Section 7 summarizes the achieved results and concludes the paper.

2 Syntactic Search

The goal of an information retrieval system is to map a natural language queries Q, which specify user information needs, to a set of documents in the document collection D, which meet these needs, and (optionally) to order these documents according to the degree of relevance. The search S in general can be represented as a mapping function:

$$S : Q \rightarrow D \tag{1}$$

In order to implement an IR System we need to decide (i) what is an atomic element (*Term*) in document and query representations, (ii) which matching techniques (*Match*) are used for matching of document and query terms, (iii) which models (*Model*) are used for document and query representations, for computing query answers and relevance ranking, and (iv) which data structures (*Data_Structure*) are used for document indexing and retrieval. Thus, the IR System is a 4-tuple:

$$IR\_System =< Model, Data\_Structure, Term, Match > \tag{2}$$

SemSearch 2008, CEUR Workshop Proceedings, ISSN 1613-0073, online at CEUR-WS.org/Vol-334/

The Bag of words model, i.e., the model in which the ordering of words in a document is not considered, is the most widely used model for document representation. The Boolean Model, the Vector Space Model, and the Probabilistic Model are the classical examples of models used for computing query answers and relevance ranking [1].

Various index structures, such as Signature File and Inverted Index, are used for efficient retrieval. Inverted Index, which stores a mapping from terms to their locations in documents, is the most popular solution [1].

In syntactic IR, *Term* and *Match* are instantiated as follows:

– *Term* - a word or a multi-words phrase,
– *Match* - a syntactic matching of words or phrases.

In the simplest case, syntactic matching is computed through search for equivalent (possibly stemmed [14]) words. Some systems approximate matching by search for words with common prefixes or words within a certain edit distance with a given word.

Let us consider the document collection shown in Figure 1.

D1 :	A small baby dog runs after a huge white cat. . . .
D2 :	A laptop computer is on a coffee table. . . .
D3 :	A little dog or a huge cat left a paw mark on a computer table. . . .

Fig. 1. A document collection

In Figure 2, we show examples of four queries, which are submitted to this document collection.

Q1 :	Babies and dogs	Q3 :	Table computer
Q2 :	Paw print	Q4 :	Carnivores

Fig. 2. Queries

An example of syntactic IR using Inverted Index technology is given in Figure 3. The two parts of an Inverted Index are: *Dictionary*, i.e., a list of terms used for document indexing; and posting lists (*Postings*), where every posing list is associated with a term and consists of documents in which this term occur. The query processing in Inverted Index is separated into two main steps: (i) to locate terms in dictionary which match query terms, and (ii) to search Inverted Index with these terms. Consider, for example, processing a query *table computer*.
First, for each query term we identify those terms in dictionary that match this term (*table* $\rightarrow$ $\{$*table*$\}$ and *computer* $\rightarrow$ $\{$*computer*$\}$). Second, we search inverted

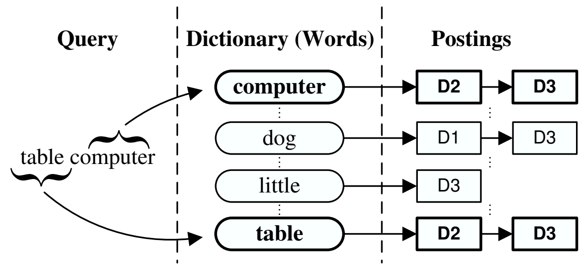

Fig. 3. Inverted Index in classical syntactic search

index with computed dictionary terms (*table* $\rightarrow$ $\{D_2, D_3\}$ and *computer* $\rightarrow$ $\{D_2, D_3\}$). And finally, we take the intersection of document sets, found for every query terms, as an answer to the query (D_2 and D_3 in our example).

There are several problems which negatively affect the performance of syntactic search. These problems are discussed bellow:

Polysemy. The same word may have multiple meanings (see Figure 4) and, therefore, in syntactic search, query results may contain documents where the query word is used in a meaning which is different from what the user had in mind. For instance, a document which talks about *baby* in the sense of a very young mammal is irrelevant if the user looks for documents about *baby* in the sense of a human child who has not yet begun to walk or talk. An answer for query $Q1$, computed by syntactic search engine, includes document $D1$, while the correct answer is an empty set.

Synonymy. Two different words can express the same meaning in a given context, i.e., they can be synonyms (see Figure 5). Syntactic search approaches do not explicitly take synonymous words into account. For instance, words *mark* and *print* are synonymous when used in the sense of a visible indication made on a surface, however, only documents using word *print* will be returned if the user query was exactly this word. An answer for query $Q2$, computed by syntactic search engine, is an empty set, while the correct answer includes document $D3$.

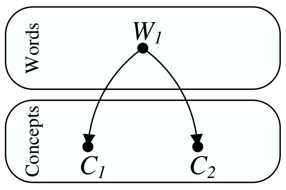

Fig. 4. Polysemy

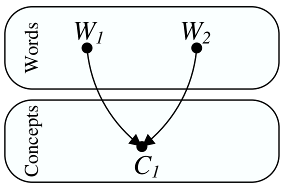

Fig. 5. Synonymy

Complex concepts. State-of-the-art syntactic search engines fall short in taking into account complex concepts formed by natural language phrases and in discriminating among them (see Figure 6). For instance, phrases *computer table* and *table computer* denote two quite different concepts, whereas a conventional search engine is very likely to return similar results if they are submitted as queries. Moreover, the results of these queries may contain documents irrelevant to both of them, e.g., a document, containing a sentence *A laptop computer is on a coffee table*, being irrelevant to both of our queries, is likely to be found as an answer to these queries. An answer for query $Q3$, computed by syntactic search engine, includes documents $D2$ and $D3$, while the correct answer is an empty set.
Related concepts. Syntactic search does not take into account concepts which are closely related to the query concept (see Figure 7). For instance, a user looking for *carnivores* might not only be interested in documents which talk about carnivores but also in those which talk about the various kinds of carnivores such as *dogs* and *cats*. An answer for query $Q4$, computed by syntactic search engine, is an empty set, while the correct answer includes documents $D1$ and $D3$.

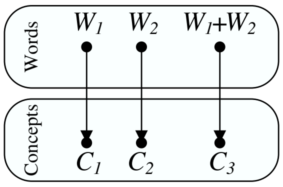

Fig. 6. Complex concepts

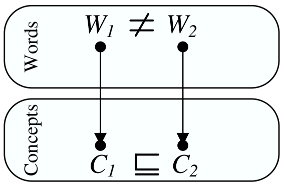

Fig. 7. Related concepts

3 Semantics Enabled Syntactic Search

In semantic search, *Term* and *Match* elements of the model, described in Formula 2, are instantiated as follows:

– *Term* - an atomic or a complex concept,
– *Match* - semantic matching of concepts.

Where concepts are computed, for example, by mapping words to concepts in a lexical database such as WordNet [13]. Semantic matching can be implemented by using semantic matching approach described in [7–9]. The main idea of semantic matching is to compare meanings (concepts) and not words, as in syntactic matching. For example, phrase *A little dog or a huge cat* syntactically is very different from a word *carnivores* but semantically they denote related concepts.

Because we build on top of standard syntactic search technology, classical information retrieval models and data structures can be fully reused in semantic search with the difference in that now words (W) are replaced with concepts (C) and syntactic matching of words ($WMatch$) is replaced with semantic matching of concepts ($SMatch$).

$$\boxed{\textit{Syntatic Search} \xrightarrow{(W \to C),\ (WMatch \to SMatch)} \textit{Semantic Search}}$$

Note that semantic search can solve the problems related to the ambiguity of natural language, namely, the problems of polysemy and synonymy, because concepts are unambiguous by definition.

In this paper we propose an approach in which semantic search is build on top of syntactic search. We call it semantics enabled syntactic search (*C-Search*). In our approach, we extend the classical syntactic search approach with semantics as follows:

– Indexing and searching documents is done using complex concepts. Complex concepts are computed by extracting multi-word phrases (that function as a single unit in the syntax of a sentence) and then by analyzing the meaning of these phrases. For example, phrase *A little dog or a huge cat* is converted into concept C(*A little dog or a huge cat*) which then is used as a single term during document indexing and retrieval. Note that because we analyze multi-word phrases we solve the problem related to complex concepts discussed in Section 2.
– The notion of complex concepts allows us to represent uncertainty (partial information) coming from the coordination conjunction “OR” in natural language. For instance, phrase *A little dog or a huge cat* represents a concept which encodes the fact that it is unknown if *a little dog* or *a huge cat* is actually described in the document. Note that classical syntactic search is not capable of representing this kind of uncertainty and, therefore, of taking it into account during indexing and retrieval.
– Searching for documents describing concepts which are semantically related to query concepts. We assume that when a user is searching for a concept she is also interested in more specific concepts. For example, the extension of concept C(*A little dog or a huge cat*) is a subset of the extension of concept C(*carnivores*). Therefore, documents describing the former concept should be returned as answers to the query describing the later concept. In our approach, semantic matching is used in order to implement a search for related (complex) concepts. It allows us to solve the problem with related concepts discussed in Section 2.
– Semantic continuum. When we move from words to concepts in semantic search it is not always possible to find a concept which corresponds to a given word. The main reason for this problem is lack of background knowledge, i.e., a concept corresponding to a given word may not exist in the lexical database. In this case, in our approach, semantic search is reduced to an underlying syntactic search, i.e., we index and retrieve by words and not by concepts. This means that *C-Search* should perform at least as good as classical syntactic search.

An example of semantics enabled syntactic search using Inverted Index technology is given in Figure 8. Analogously to syntactic search, the query processing in semantics enabled Inverted Index is separated into two main steps: (i) to locate terms (which can be concepts or words) in dictionary which match query terms, and (ii) to search Inverted Index with these terms. Note that the second step is identical to that of syntactic search. First step may require semantic matching of (complex) concepts in a query to (complex) document concepts stored in the Inverted Index dictionary (see Section 4). Consider, for example, processing a query *mark of canine or feline*.

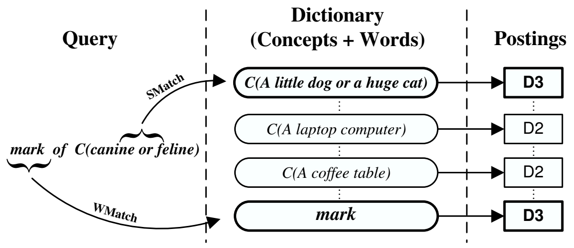
Query
Dictionary (Concepts + Words)
Postings
mark of C(canine or feline)
SMatch
WMatch
C(A little dog or a huge cat)
C(A laptop computer)
C(A coffee table)
mark
D3
D2
D2
D3

Fig. 8. Inverted Index in *C-Search*

Assume that words *canine* and *feline* present in our lexical database and word *mark* does not. In this case, phrase *canine or feline* will be converted into a complex concept $C(canine\ or\ feline)$ defined as a set of all fissiped mammals with non-retractile claws and typically long muzzles, or lithe-bodied roundheaded fissiped mammals with retractile claws, and word *mark* will not be changed. Modified query is processed as follows. First, for each query term, i.e., for word *mark* and for concept $C(canine\ or\ feline)$, we identify those terms in dictionary that match these query terms ($mark \xrightarrow{WMatch} \{mark\}$ and $C(canine\ or\ feline) \xrightarrow{SMatch} \{C(A\ little\ dog\ or\ a\ huge\ cat)\}$). Second, we search inverted index with computed dictionary terms ($mark \rightarrow \{D_3\}$ and $C(A\ little\ dog\ \ or\ a\ huge\ cat) \rightarrow \{D_3\}$). And finally, we take the intersection of document sets, found for every query term, as an answer to the given query (D_3 in our example).

4 Concept Indexing

In this section, we discuss how we implement the semantic matching of (complex) query concepts C^q to related (complex) document concepts C^d stored in the Inverted Index dictionary. Let $C_{ms}(C^q)$ be a set of all (complex) document concepts C^d matching (complex) query concept C^q, i.e., a set of all C^d, which are equivalent or more specific (ms) than the given C^q.

$$C_{ms}(C^q) = \{C^d \mid C^d \sqsubseteq C^q\} \tag{3}$$

During the query processing we need to compute set $C_{ms}(C^q)$ for every query concept C^q in the query. One approach to computing this set is to sequentially iterate through each concept C^d, compare it to the given query concept C^q by using semantic matching [7–9] technique, and collect those concepts for which semantic matching returns *more specific* ($\sqsubseteq$) relation. This approach may become prohibitory expensive as there may be thousands and millions of concepts stored in the document index dictionary. In this section we show how Inverted Index technology can be used in order to allow search for concepts in $C_{ms}(C^q)$, as efficient as Inverted Index technology can allow.

It is known, that in natural language, concepts are expressed as noun phrases [17]. In order to support complex concepts which encode uncertainty (see Section 3), we introduce the notion of descriptive phrase, where descriptive phrase is a set of noun-phrases, representing alternative concepts, connected by coordinating conjunction "OR":

$$descriptive\_phrase ::= noun\_phrase \; \{OR \; noun\_phrase\} \tag{4}$$

Descriptive phrases are converted into concepts expressed in Propositional Description Logic language L^C by following the approach described in [5]. Complex document concepts extracted from descriptive phrases are DNF formulas of atomic concepts representing words

$$C^d = \sqcup \sqcap A^d \tag{5}$$

For instance, descriptive phrase *A little dog or a huge cat* is converted into the following complex concept.

$$C_1^d(A\ little\ dog\ or\ a\ huge\ cat) = (A(little) \sqcap A(dog)) \sqcup (A(huge) \sqcap A(cat))$$

where $A(w)$ is an atomic concept corresponding to the word w.

Let $\mathbf{C^{DNF}}$ be the set of all complex document concepts and $\mathbf{C}^{\sqcap}$ be the set of conjunctive clauses from which concepts in $\mathbf{C^{DNF}}$ are composed. For instance, concept C_1^d belongs to $\mathbf{C^{DNF}}$ and its conjunctive clauses, i.e., concepts $C_2 = A(little) \sqcap A(dog)$ and $C_3 = A(huge) \sqcap A(cat)$, belong to $\mathbf{C}^{\sqcap}$.

Assume that query concept is converted into CNF

$$C^q = \sqcap \sqcup A^q \tag{6}$$

Recall also that if A,B, and C are concepts, then:

$$\begin{aligned} (A \sqcup B) \sqsubseteq C &\iff A \sqsubseteq C \text{ and } B \sqsubseteq C \\ A \sqsubseteq (B \sqcap C) &\iff A \sqsubseteq B \text{ and } A \sqsubseteq C \end{aligned} \tag{7}$$

Given 5, 6, and 7, Formula 3 can be rewritten as follows:

$$
\begin{aligned}
C_{ms}(C^q) &= \{C^d \in \mathbf{C^{DNF}} \mid (\sqcup \sqcap A^d) \sqsubseteq (\sqcap \sqcup A^q)\} \\
&= \{C^d \in \mathbf{C^{DNF}} \mid \forall(\sqcup A^q) \in C^q, \forall(\sqcap A^d) \in C^d, (\sqcap A^d) \sqsubseteq (\sqcup A^q)\} \\
&= \bigcap_{\sqcup A^q \in C^q} \{C^d \in \mathbf{C^{DNF}} \mid \forall(\sqcap A^d) \in C^d, (\sqcap A^d) \sqsubseteq (\sqcup A^q)\} \\
&= \bigcap_{\sqcup A^q \in C^q} C_{ms}(\sqcup A^q)
\end{aligned} \tag{8}
$$

where by $C_{ms}(\sqcup A^q)$ we denote the set of all concepts in $\mathbf{C^{DNF}}$ which are equivalent to or more specific than disjunctive clause $\sqcup A^q$:

$$C_{ms}(\sqcup A^q) = \{C^d \in \mathbf{C^{DNF}} \mid \forall(\sqcap A^d) \in C^d, (\sqcap A^d) \sqsubseteq (\sqcup A^q)\} \tag{9}$$

Formula 9 can be rewritten as follows:

$$C_{ms}(\sqcup A^q) = \{C^d \in \mathbf{C^{DNF}} \mid \forall(\sqcap A^d) \in C^d, (\sqcap A^d) \in C^{\sqcap}_{ms}(\sqcup A^q)\} \tag{10}$$

where by $C^{\sqcap}_{ms}(\sqcup A^q)$ we denote the set of all conjunctive clauses in $\mathbf{C}^{\sqcap}$ which are equivalent to or more specific than the given disjunctive clause $(\sqcup A^q)$:

$$C^{\sqcap}_{ms}(\sqcup A^q) = \{\sqcap A^d \in \mathbf{C}^{\sqcap} \mid (\sqcap A^d) \sqsubseteq (\sqcup A^q)\} \tag{11}$$

Set $C_{ms}(\sqcup A^q)$ (see Formula 10) consists of complex concepts $C^d \in \mathbf{C^{DNF}}$ which have all its conjunctive clauses $\sqcap A^d$ in $C^{\sqcap}_{ms}(\sqcup A^q)$. In order to allow fast computation of $C_{ms}(\sqcup A^q)$ at query time, every concept $C^d \in \mathbf{C^{DNF}}$ containing more than one conjunctive clause is indexed (at indexing time) by its conjunctive clauses in the index which we call the concept $\sqcup$-index. Concept $\sqcup$-index stores a mapping from each conjunctive clause to a set of all concepts $C^d \in \mathbf{C^{DNF}}$ which contain this conjunctive clause (*conjunctive_clause* $\rightarrow$ {*dnf_concept*}). In Figure 9 we show a fragment of a concept $\sqcup$-index for concept C^d_1.

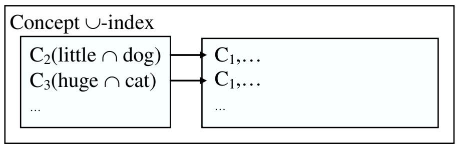

Fig. 9. Concept $\sqcup$-index

Now let us consider set $C^{\sqcap}_{ms}(\sqcup A^q)$ (see Formula 11). Notice that from WordNet we can extract only relations between atomic concepts (e.g., $A \sqsubseteq B$).

Therefore, using WordNet as our background knowledge, we can prove that $(\sqcap A^d) \sqsubseteq (\sqcup A^q)$ only if $\exists A^q, \exists A^d$, such that $A^d \sqsubseteq A^q$. Taking this into account, Formula 11 can be rewritten as follows:

$$\begin{aligned} C^{\sqcap}_{ms}(\sqcup A^q) &= \{\sqcap A^d \in \mathbf{C}^{\sqcap} \mid \exists A^q, \exists A^d, \text{ s.t. } A^d \sqsubseteq A^q\} \\ &= \bigcup_{A^q \in \sqcup A^q} \{\sqcap A^d \in \mathbf{C}^{\sqcap} \mid \exists A^d, \text{ s.t. } A^d \sqsubseteq A^q\} = \bigcup_{A^q \in \sqcup A^q} C^{\sqcap}_{ms}(A^q) \end{aligned} \tag{12}$$

where by $C^{\sqcap}_{ms}(A^q)$ we denote the set of all conjunctive clauses $\sqcap A^d \in \mathbf{C}^{\sqcap}$ which are equivalent to or more specific than the given atomic concept A^q:

$$C^{\sqcap}_{ms}(A^q) = \{\sqcap A^d \in \mathbf{C}^{\sqcap} \mid \exists A^d, \text{ s.t. } A^d \sqsubseteq A^q\} \tag{13}$$

Formula 13 can be rewritten as follows:

$$C^{\sqcap}_{ms}(A^q) = \{\sqcap A^d \in \mathbf{C}^{\sqcap} \mid \exists A^d, \text{ s.t. } A^d \in A_{ms}(A^q)\} \tag{14}$$

where by $A_{ms}(A^q)$ we denote a set of all atomic concepts A^d which are equivalent to or more specific than the given atomic concept A^q:

$$A_{ms}(A^q) = \{A^d \mid A^d \sqsubseteq A^q\} \tag{15}$$

Set $C^{\sqcap}_{ms}(A^q)$ (see Formula 14) consists of conjunctive clauses $\sqcap A^d \in \mathbf{C}^{\sqcap}$ with at least one its atomic concept A^d in $A_{ms}(A^q)$. In order to allow fast computation of $C^{\sqcap}_{ms}(A^q)$ at query time, conjunctive clauses $C^{\sqcap}$, containing more than one atomic concept, are indexed (at indexing time) by them in the index which we call the concept $\sqcap$-index. Concept $\sqcap$-index stores a mapping from each atomic concept to a set of all conjunctive clauses in $\mathbf{C}^{\sqcap}$ which contains this concept (*atomic_concept* $\rightarrow$ {*conjunctive_clause*}). In Figure 10, we show a fragment of a concept $\sqcap$-index which indexes conjunctive clauses of concept C^d_1, i.e., it indexes concepts C_2 and C_3.

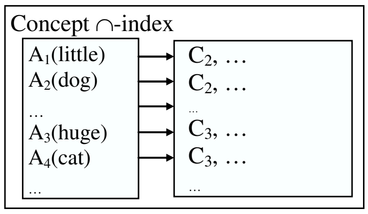

Fig. 10. Concept $\sqcap$-index

Now we will describe how concept retrieval, i.e., computation of $C_{ms}(C^q)$, can be performed given that concept $\sqcap$- and $\sqcup$- indices were constructed. As an example of query concept we will consider the following concept.

$$C_1^q \equiv A(canine) \sqcup A(feline)$$

Set $C_{ms}(C^q)$ is computed in the following six steps:
1. Query concept is converted into CNF. For example, concept C_1^q is already in CNF, so it will not be changed.
2. For every atomic concept A^q we search the lexical database for all atomic concepts which are equivalent to or more specific than A^q, i.e., we compute set $A_{ms}(A^q)$ (see Formula 15). For example, $A_{ms}(A(canine)) = \{A(dog), A(wolf), \dots\}$ and $A_{ms}(A(feline)) = \{A(cat), A(lion), \dots\}$.
3. For every atomic concept A^q we compute set $C^{\sqcap}_{ms}(A^q)$ (see Formula 14), i.e., a set of all conjunctive clauses which are equivalent to or more specific than A^q. Sets $C^{\sqcap}_{ms}(A^q)$ are computed by searching concept $\sqcap$-index with atomic concepts in $A_{ms}(A^q)$. For example, $C^{\sqcap}_{ms}(A(canine)) = \{C_2, \dots\}$ and $C^{\sqcap}_{ms}(A(feline)) = \{C_3, \dots\}$.
4. For every disjunctive clause $\sqcup A^q$ we compute a set $C^{\sqcap}_{ms}(\sqcup A^q)$ (see Formula 12), i.e., a set of all conjunctive clauses which are equivalent to or more specific than disjunctive clause $\sqcup A^q$. We compute $C^{\sqcap}_{ms}(\sqcup A^q)$ by taking the union of all the sets $C^{\sqcap}_{ms}(A^q)$:

$$C^{\sqcap}_{ms}(\sqcup A^q) = \bigcup_{A^q \in \sqcup A^q} C^{\sqcap}_{ms}(A^q) \tag{16}$$

For example, $C^{\sqcap}_{ms}(A(canine) \sqcup A(feline)) = \{C_2, C_3, \dots\}$.
5. For every disjunctive clause $\sqcup A^q$ we compute set $C_{ms}(\sqcup A^q)$ (see Formula 10), i.e., a set of all complex document concepts in $\mathbf{C^{DNF}}$ which are equivalent to or more specific than disjunctive clause $\sqcup A^q$. Sets $C_{ms}(\sqcup A^q)$ are computed by searching concept $\sqcup$-index with conjunctive clauses in $C^{\sqcap}_{ms}(\sqcup A^q)$. Note that we search only for concepts C^d which have all its conjunctive clauses in $C^{\sqcap}_{ms}(\sqcup A^q)$, and discard other concepts. For example, $C_{ms}(A(canine) \sqcup A(feline)) = \{C_1^d, \dots\}$.
6. We compute $C_{ms}(C^q)$ (see Formula 8) by taking the intersection of all the sets $C_{ms}(\sqcup A^q)$:

$$C_{ms}(C^q) = \bigcap_{\sqcup A^q \in C^q} C_{ms}(\sqcup A^q) \tag{17}$$

For example, concept C_1^q has only one disjunctive clause, therefore, set $C_{ms}(C_1^q)$ is equal to set $C_{ms}(A(canine) \sqcup A(feline))$, i.e., $C_{ms}(C_1^q) = \{C_1^d, \dots\}$.

Note that steps described above require searching the lexical database, searching inverted indices, computing union and intersection of sets. All these operations are fast in practice and, therefore, the computation of $C_{ms}(C^q)$ is also time efficient.

5 Evaluation

The data-set, used for the evaluation of our approach, was generated from *Home*[1] subtree of DMoz web directory. Documents classified to nodes in the sub-tree are used as a document set, labels of nodes are used as a query set[2], and node-document links represents relevance of documents to queries. The data-set consists of 29506 documents and 890 queries.

To locate descriptive phrases in documents and queries we, first, follow a standard NLP pipeline to locate noun phrases, i.e., we perform sentence detection, tokenization, part-of-speech (POS) tagging, and noun phrase chunking and after that we perform addition step which we call descriptive phrase chunking, where the goal of this step is to locate descriptive phrases, satisfying Formula 4, given that noun phrases are already identified. In particular, we use the GATE [3] infrastructure and resources. Queries usually are short phrases and, as shown in [20], standard NLP technology, primarily designed to be applied on full-fledged sentences, is not effective enough in its application on such phrases. Therefore, for query processing we use a POS-tagger from [20], which is specifically trained on short phrases.

The conversion of descriptive phrases into formulas in L^C was performed as follows. First, for each token in a descriptive phrase, we looked up and enumerated its meaning(s) in WordNet [13]. Next, we performed word sense filtering, i.e., we discard word senses which are not relevant in the given context. In order to do this, we followed the approach presented in [20], which exploits POS tagging information and WordNet lexical database for WSD in short noun phrases. Differently from [20] we did not use the filtering technique which leaves only the most probable sense of the word, because of its low accuracy. Finally, for every descriptive phrase we build a complex concept which encodes the meaning of this phrase. Each word is represented as an atomic concept, noun phrases are translated into logical conjunction of atomic concepts, and descriptive phrases are translated into logical disjunction of formulas for noun phrases.

In order to evaluate our approach, we built two inverted indices. First index was build by using Lucene[3]. Second index was build by using semantics enabled version of Lucene, which was implemented following the methodology described in Sections 3 and 4. Evaluation results for both indexes are reported in Table 1.

Table 1. Evaluation results

	Precision (%)	Recall (%)
Lucene	7.72	20.43
C-Search	8.40	24.69

[1] http://www.dmoz.org/Home/.

[2] Queries were created by concatenation of node's and its parent's labels adding "AND" in between. Queries created from nodes which contained less than 10 or more than 100 documents were eliminated from the query set.

[3] http://lucene.apache.org/java/docs/index.html

After manual inspection of the results, we concluded that the main reason for low precision and recall, achieved by Lucene and C-Search, is low quality of the data-set. Documents in our collection represent web-sites with many interconnected pages, whereas we indexed only root page for each web-site. This leads to low recall because relevant information can be stored on pages other than the root page. Queries in the used data-set are also not always good, for instance, query *purchasing AND new* (created from node New[4]) was associated only with documents about automobiles because nodes *purchasing* and *new* are children of the node *automobiles*, whereas, obviously, information about purchasing of something new can be found in documents from other subtrees. The problem with queries leads to low precision. Nevertheless, in this particular data set, *C-Search* performed better than purely syntactic search, which supports the underlying assumption of our approach.

6 Related work

The fact that the syntactic nature of the classical IR leads to problems with precision was recognized in the IR community long ago (e.g., see [18]). There were two major approaches to addressing this problem: one is based on natural language processing and machine learning techniques in which (noun) phrases in a document corpus are identified and organized in a subsumption hierarchy which is then used to improve the precision of retrieval (e.g., see [19]); and the other is based on a linguistic database which is used to associate words in a document corpus with atomic lexical concepts in the database and then to index these documents by the associated concepts (e.g., see [16]). Our approach is different from these two because the former approach is still essentially syntactic (and semantics is only implicitly derived with no guarantee of correctness) and in the latter approach only atomic concepts are indexed, wherein *C-Search* allows for indexing of complex concepts and explicitly take into account possible relations between them which allows it to compute more accurate query results. More importantly, our approach *extends* syntactic search and not replaces it as it is the case in the latter approach. Therefore, our approach supports a continuum from purely syntactic to fully semantic IR in which indexing and retrieval can be performed at any point of the continuum depending on how much semantic data are available.

In the Semantic Web community, semantic search is primarily seen as the task of querying an RDF graph based on the mapping of terms appearing in the input natural language query to the elements of the graph. An analysis of existing semantic search systems is provided in [10]. Our approach is principally different because, like in classical IR, input query is mapped to document contents and not to elements of a knowledge representation structure. Document retrieval approaches developed in the context of the Semantic Web are surveyed in [12]. Matching of document and query representations, in these approaches, is based on query expansion (e.g., see [2]), graph traversal (e.g., see [15]), and

[4] http://www.dmoz.org/Home/Consumer_Information/Automobiles/Purchasing/New/.

RDF reasoning (e.g., see [4]). Differently from these approaches, in *C-Search*, document and query representations are matched via semantic matching [7–9] of complex concepts, which is implemented by using Inverted Index technology.

7 Conclusions

In this paper we presented an approach in which syntactic IR is extended with a semantics layer which allows it to improve over results of a purely syntactic search. The proposed approach performs as good as syntactic search while allowing for an improvement where semantics is properly integrated. In principle, our approach supports a continuum from purely syntactic to fully semantic IR in which indexing and retrieval can be performed at any point of the continuum depending on how much semantic data are available. The reported experimental results demonstrate the proof of concept of the proposed solution. Future work includes: (i) development of document relevance metrics based on both syntactic and semantic similarity of query and document descriptions; (ii) integration of more accurate algorithms for concept identification during indexing; (iii) comparing the performance of the proposed solution with the state-of-the-art syntactic IR systems using a syntactic IR benchmark; and, (iv) providing support for queries in which concepts can be associated with a semantic scope such as equivalence, more/less general, disjoint.

References

1. Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press / Addison-Wesley, 1999.
2. Irene Celino, Emanuele Della Valle, Dario Cerizza, and Andrea Turati. Squiggle: a semantic search engine for indexing and retrieval of multimedia content. In *SEMPS*, pages 20–34, 2006.
3. H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.
4. John Davies and Richard Weeks. QuizRDF: Search technology for the semantic web. In *HICSS '04: Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4*, page 40112, Washington, DC, USA, 2004. IEEE Computer Society.
5. Fausto Giunchiglia, Maurizio Marchese, and Ilya Zaihrayeu. Encoding classifications into lightweight ontologies. In *Journal on Data Semantics (JoDS) VIII*, Winter 2006.
6. Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Discovering missing background knowledge in ontology matching. In *Proc. of ECAI*, 2006.
7. Fausto Giunchiglia and Mikalai Yatskevich. Element level semantic matching. In *Meaning Coordination and Negotiation workshop, ISWC*, 2004.
8. Fausto Giunchiglia, Mikalai Yatskevich, and Enrico Giunchiglia. Efficient semantic matching. In *Proc. of ESWC*, Lecture Notes in Computer Science. Springer, 2005.

9. Fausto Giunchiglia, Mikalai Yatskevich, and Pavel Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics (JoDS)*, 9:1–38, 2007.
10. M. Hildebrand, J. van Ossenbruggen, and L. Hardman. An analysis of search-based user interaction on the semantic web. Technical Report INS-E0706, Centrum voor Wiskunde en Informatica, MAY 2007.
11. Bernardo Magnini, Manuela Speranza, and Christian Girardi. A semantic-based approach to interoperability of classification hierarchies: evaluation of linguistic techniques. *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, pages 11–33, 2004.
12. Christoph Mangold. A survey and classification of semantic search approaches. *Int. J. Metadata Semantics and Ontology*, 2(1):23–34, 2007.
13. George Miller. *WordNet: An electronic Lexical Database*. MIT Press, 1998.
14. M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
15. C. Rocha, D. Schwabe, and M. de Aragao. A hybrid approach for searching in the semantic web. In *Proceedings of the 13th International World Wide Web Conference*, 2004.
16. Hinrich Schutze and Jan O. Pedersen. Information retrieval based on word senses. In *Fourth Annual Symposium on Document Analysis and Information Retrieval*, 1995.
17. J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.
18. Christopher Stokoe, Michael P. Oakes, and John Tait. Word sense disambiguation in information retrieval revisited. pages 159–166, 2003.
19. William A. Woods. Conceptual indexing: A better way to organize knowledge. 1997.
20. I. Zaihrayeu, L. Sun, F. Giunchiglia, W. Pan, Q. Ju, M. Chi, and X. Huang. From web directories to ontologies: Natural language processing challenges. In *6th International Semantic Web Conference (ISWC 2007)*. Springer, 2007.