
UNIVERSITY
OF TRENTO

**DIPARTIMENTO DI INGEGNERIA E SCIENZA DELL'INFORMAZIONE**

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.disi.unitn.it


# CONCEPT SEARCH


Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu



August 2008

Technical Report # DISI-08-037

Also: in the 6th Annual European Semantic Web Conference 2009 (ESWC2009)

# Concept 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.** In this paper we present a novel approach, called Concept Search, which extends syntactic search, i.e., search based on the computation of string similarity between words, with semantic search, i.e., search based on the computation of semantic relations between concepts. The key idea of Concept Search is to operate on complex concepts and to maximally exploit the semantic information available, reducing to syntactic search only when necessary, i.e., when no semantic information is available. The experimental results show that Concept Search performs at least as well as syntactic search, improving the quality of results as a function of the amount of available semantics.

## 1 Introduction

Historically, there have been two major approaches to information retrieval that we call *syntactic search* and *semantic search*. Syntactic search use words or multi-word phrases as atomic elements in document and query representations. The search procedure is essentially based on the *syntactic matching* of document and query representations. Search engines, exploiting syntactic search, are known to suffer in general from low precision while being good at recall. Semantic search is based on fetching document and query representations through a semantic analysis of their contents using natural language processing techniques and, later, on retrieving documents by matching these semantic representations. The idea is that, differently from syntactic search, semantic search exploits the *meaning* of words, thus avoiding many of the well known problems of syntactic search, e.g., the problems of polysemy and synonymy. Semantics-based approaches, in general, allow to reach a higher precision but lower recall [11].

In this paper we propose a novel approach called *Concept Search* (*C-Search* in short) which extends syntactic search with semantics. The main idea is to keep the same machinery which has made syntactic search so successful, but to modify it so that, whenever possible, syntactic search is substituted by semantic search, thus improving the system performance. This is why we say that *C-Search* is *semantics enabled syntactic search*. Semantics can be enabled along different dimensions, on different levels, and to different extents forming a space of approaches lying between purely syntactic search and fully semantic search. We call this space the *semantic continuum*. In principle, *C-Search* can be tuned to work at any point in the semantic continuum taking advantage of semantics

when and where possible. As a special case, when no semantic information is available, *C-Search* reduces to syntactic search, i.e., results produced by *C-Search* and syntactic search are the same.

The remainder of the paper is organized as follows. In Section 2, we first discuss information retrieval (IR) in general, and then focus on syntactic search. In this section, we provide a general model of IR which will be latter extended to semantic search. In Section 3, we introduce and describe the semantic continuum. In Section 4, we describe how *C-Search* is positioned within the semantic continuum. In Section 5, we show how *C-Search* can be efficiently implemented using inverted index technology. Section 6 presents experimental results. In Section 7, we discuss the state-of-the-art in semantic search and compare our approach with other related approaches. Section 8 concludes the paper.

## 2 Syntactic Search

The goal of an IR system is to map a natural language query $q$ (in a query set $Q$), which specifies a certain user information needs, to a set of documents $d$ in the document collection $D$ which meet these needs, and to order these documents according to their relevance. *IR* can therefore be represented as a mapping function:

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

In order to implement an IR System we need to decide (i) which models (*Model*) are used for document and query representation, for computing query answers and relevance ranking, (ii) which data structures (*Data Structure*) are used for indexing document representations in a way to allow for an efficient retrieval, (iii) what is an atomic element (*Term*) in document and query representations, and (iv) which matching techniques (*Match*) are used for matching document and query terms. Thus, an IR System can be abstractly modelled as the following 4-tuple:

$$IR\_System = \langle Model,\ Data\ Structure,\ Term,\ Match \rangle \tag{2}$$

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 [13]. Conventional search engines rank query results using the cosine similarity from the vector space model with terms weighted by different variations of the *tf-idf* weight measure. Various index structures, such as the *signature file* and the *inverted index*, are used as *data structures* for efficient retrieval. Inverted index, which stores mappings from terms to their locations in documents, is the most popular solution [13]. Finally, in syntactic search, *Term* and *Match* are instantiated as follows:

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

In the simplest case, syntactic matching is implemented as search for equivalent words. These words are often stemmed. Furthermore, some systems perform approximate matching by searching for words with common prefixes or words within a certain edit distance with a given word.

There are several problems which may negatively affect the performance of syntactic search, as discussed below.

**Polysemy**. The same word may have multiple meanings and, therefore, query results, computed by a syntactic search engine, 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 $D1$ (in Figure 1) 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 (see query $Q1$ in Figure 1). An answer for query $Q1$, computed by a syntactic search engine, includes document $D1$, while the correct answer is the empty set.

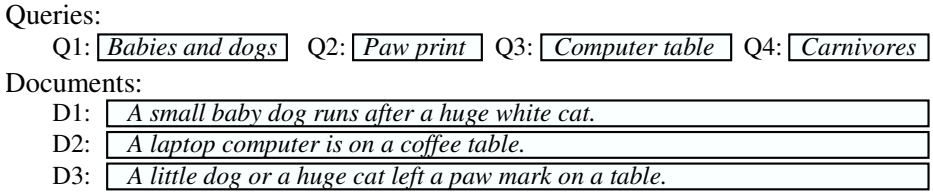


**Fig. 1.** Queries and a document collection

**Synonymy**. Two different words can express the same meaning in a given context, i.e., they can be synonyms. 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 $Q2$ (in Figure 1), computed by a syntactic search engine, is the empty set, while the correct answer includes the document $D3$.

**Complex concepts**. Syntactic search engines fall short in taking into account complex concepts formed by natural language phrases and in discriminating among them. Consider, for instance, document $D2$ (in Figure 1). This document describes two concepts: *a laptop computer* and *a coffee table*. Query $Q3$ (in Figure 1) denotes concept *computer table* which is quite different from both concepts described in $D2$, whereas a syntactic search engine is likely to return $D2$ in response to $Q3$, because both words computer and table occur in this document. The correct answer is the empty set.

**Related concepts**. Syntactic search does not take into account concepts which are semantically related to the query concepts. 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 $Q4$ (in Figure 1), computed by a syntactic search, is the empty set, while the correct answer might include documents $D1$ and $D3$, depending on user information needs and available semantic information.

## 3 The Semantic Continuum

In order to address the problems of syntactic search described in Section 2, we extend syntactic search with semantics. In the following, we identify three dimensions where semantics can improve syntactic search and represent these dimensions in the cartesian space shown in Figure 2.

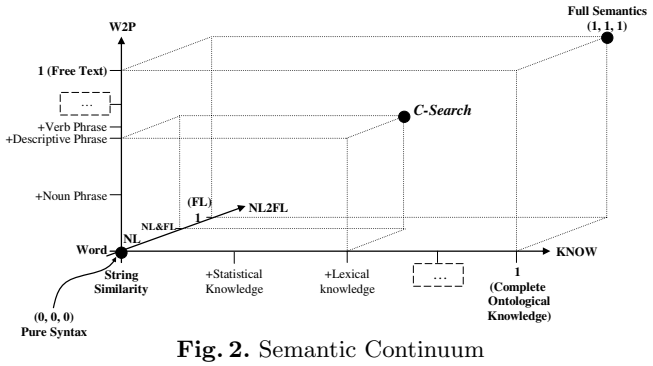


**Fig. 2.** Semantic Continuum

**From natural language to formal language** (NL2FL-axis in Figure 2). To solve the problems related to the ambiguity of natural language, namely, the problems of polysemy and synonymy, we need to move from words, expressed in a natural language, to concepts (word senses), expressed in an unambiguous formal language. An overview of existing approaches to sense based IR is presented in [17]. In the NL2FL-axis, 0 represents the situation where only words are used, while 1 represents the situation where only concepts are used. When we move from words to concepts, it is not always possible to find a concept which corresponds to a given word. The main reason for this problem is the lack of background knowledge [8], i.e., a concept corresponding to a given word may not exist in the lexical database. To address this problem, indexing and retrieval in the continuum are performed by using both syntactic and semantic information, i.e., a word itself is used as a concept, when its denoted concept is not known.

**From words to phrases** (W2P-axis in Figure 2). To solve the problem related to complex concepts, we need to analyze natural language phrases, which denote these concepts. It is well known that in natural language concepts are expressed mostly as noun phrases [19]. An example of a noun phrase parsing algorithm and its application to document indexing and retrieval is described in [23]. There are approaches in which the conceptual content of noun phrases is also analyzed (see e.g. [1]). In general, concepts can be expressed as more complex phrases (e.g., verb phrases) and possibly as a free text. In the W2P-axis, 0 represents the situation where only single words are used, while 1 represents the situation where complex concepts, extracted from a free text, are used.

**From string similarity to semantic similarity** (KNOW-axis in Figure 2). The problem with related concepts can be solved by incorporating knowledge about term relatedness. For instance, it can be statistical knowledge about word co-occurrence (see e.g. [6]), lexical knowledge about synonyms and related words

(see e.g. [14]), or ontological knowledge about classes, individuals, and their relationships (see e.g. [15]). In the KNOW-axis, 0 represents the situation where only string similarity is used during the term matching process, while 1 represents the situation where complete ontological knowledge is used during term matching.

The three-dimensional space contained in the cube (see Figure 2) represents the *semantic continuum* where the origin (0,0,0) is a purely syntactic search, the point with coordinates (1,1,1) is a fully semantic search, and all points in between represent search approaches in which semantics is enabled to different extents. *C-Search* can be positioned anywhere in the semantic continuum with syntactic search being its base case, and semantic search being the optimal solution, at the moment beyond the available technology.

## 4 Concept Search

*C-Search* is implemented according to the model of Equations 1 and 2 from Section 2. In our proposed solution, *C-Search* reuses retrieval models (*Model*) and data structures (*Data Structure*) of syntactic search with the only difference in that now words ($W$) are substituted with concepts ($C$) and syntactic matching of words ($WMatch$) is extended to semantic matching of concepts ($SMatch$). This idea is schematically represented in the equation below:

$$\boxed{\textit{Syntatic Search} \xrightarrow{Term(W \rightarrow C),\ Match(WMatch \rightarrow SMatch)} \textit{C-Search}}$$

Let us consider in details how the words in $W$ are converted into the complex concepts in $C$ and also how the semantic matching $SMatch$ is implemented.

### 4.1 From Words To Complex Concepts ($W \rightarrow C$)

Searching documents, in *C-Search*, is implemented using complex concepts expressed in a propositional Description Logic (DL) language (i.e., a DL language without roles). Complex concepts are computed by analyzing meaning of the words and phrases.

Single words are converted into atomic concepts uniquely identified as *lemma-sn*, where *lemma* is the lemma of the word, and *sn* is the sense number in WordNet. For instance, the word *dog* used in the sense of a domestic dog, which is the first sense in WordNet, is converted into the atomic concept *dog*-1. The conversion of words into concepts is performed as follows. First, we look up and enumerate all meanings of the word in WordNet. Next, we perform word sense filtering, i.e., we discard word senses which are not relevant in the given context. In order to do this, we follow the approach presented in [22], which exploits POS tagging information and WordNet lexical database for disambiguation of words in short noun phrases. Differently from [22] we do not use the disambiguation technique which leaves only the most probable sense of the word, because of its low accuracy. If more than one sense is left after the word sense filtering step

then we keep all these senses. If no senses from WordNet are found then *lemma* is used as the identifier for the atomic concept. In this case, *C-Search* is reduced to syntactic search.

Complex concepts are computed by extracting phrases and by analyzing their meaning. Noun phrases are translated into the logical conjunction of atomic concepts corresponding to the words. For instance, the noun phrase *A little dog* represents a concept, whose extension is the set of all dogs of a small size. It is translated into the concept *little*-4 $\sqcap$ *dog*-1.

Concepts in natural language can be described ambiguously. For instance, the phrase *A little dog or a huge cat* represents a concept which encodes the fact that it is unknown whether the *only* animal described in the document is *a little dog* or *a huge cat*. In order to support complex concepts which encode uncertainty (partial information) coming from the coordination conjunction *OR* in natural language, we introduce the notion of *descriptive phrase*. We define descriptive phrase as a set of noun phrases, representing alternative concepts, connected by *OR*:

$$descriptive\_phrase ::= noun\_phrase \ \{OR \ noun\_phrase\} \tag{3}$$

Descriptive phrases are translated into logical disjunction of formulas corresponding to the noun phrases. For instance, phrase *A little dog or a huge cat* is translated into concept (*little*-4 $\sqcap$ *dog*-1) $\sqcup$ (*huge*-1 $\sqcap$ *cat*-1). To locate descriptive phrases 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. The first step allows as to locate noun phrases. As a second step, that we call descriptive phrase chunking, we locate descriptive phrases satisfying Formula 3. Our implementation is based on GATE [4].

Queries usually are short phrases (i.e., 1-3 words) and, as shown in [22], standard NLP technology, primarily designed to be applied on full-fledged sentences, is not effective enough in this application scenario. An atomic concept, in a query, can be computed incorrectly, because of the selection of a wrong part-of-speech tag. In order to address this problem, for *short queries*, we use a POS-tagger which is specifically trained on short phrases [22]. On the other hand, for *long queries* (i.e., 4 words or more), we use the standard NLP technology.

Even if atomic concepts are computed correctly, complex concepts can be erroneously computed. One of the reasons is that a complex concept can be represented as a sequence of words without following the grammar for noun phrases. For instance, the query *cat huge* is converted into two atomic concepts *cat*-1 and *huge*-1, while the correct concept might be *cat*-1 $\sqcap$ *huge*-1. Another reason is that a query describing more than one concept, without properly separating them, can be recognized as a single complex concept. For instance, the query *dog cat* is converted into the concept *dog*-1 $\sqcap$ *cat*-1, while the user might be actually looking for a document describing both animals, i.e., *dog*-1 and *cat*-1. The examples described above show that, in general, it is unknown how atomic concepts $A_1^q, \ldots, A_n^q$, extracted from short queries, should be combined in order to build complex

query concepts. To represent this uncertainty we use the following query

$$(A_1^q \text{ AND } \ldots \text{ AND } A_n^q) \text{ AND } (A_1^q \sqcup \cdots \sqcup A_n^q) \tag{4}$$

where the first part ($A_1^q$ AND ... AND $A_n^q$) encodes the fact that it is *known* that the query answer should contain documents which are relevant to all the atomic concepts in the query. The second part, i.e., the complex concept $A_1^q \sqcup \cdots \sqcup A_n^q$, can be equivalently rewritten as $(A_1^q) \sqcup (A_2^q) \sqcup \cdots \sqcup (A_1^q \sqcap A_2^q) \sqcup \cdots \sqcup (A_1^q \sqcap \cdots \sqcap A_n^q)$ and, therefore, encodes the fact that it is *unknown* which complex concept (e.g., $A_1^q \sqcap A_2^q$, or $A_1^q \sqcap \cdots \sqcap A_n^q$) should be actually described.

### 4.2 From Word to Concept Matching ($WMatch \rightarrow SMatch$)

In *C-Search*, we allow the search of 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.[1] For example, the extension of concept (*little*-4 $\sqcap$ *dog*-1) $\sqcup$ (*huge*-1 $\sqcap$ *cat*-1) is a subset of the extension of concept *carnivore*-1. Therefore, documents describing the former concept should be returned as answers to the query describing the latter concept. Formally a query answer $A(C^q, \mathcal{T})$ is defined as follows:

$$A(C^q, \mathcal{T}) = \{d \mid \exists C^d \in d, \text{ s.t. } \mathcal{T} \models C^d \sqsubseteq C^q\} \tag{5}$$

where $C^q$ is a complex query concept extracted from the query $q$, $C^d$ is a complex document concept extracted from the document $d$, and $\mathcal{T}$ is a terminological knowledge base (the background knowledge) which is used in order to check if $C^d$ is more specific then $C^q$. Equation 5 states that the answer to a query concept $C^q$ is the set of all documents $d$, such that, there exists concepts $C^d$ in $d$ which is more specific than the query concept $C^q$.

During query processing we need to compute $A(C^q, \mathcal{T})$ for every query concept $C^q$ in the query. One approach is to sequentially iterate through each concept $C^d$, compare it to the query concept $C^q$ using semantic matching [9], and collect those concepts for which semantic matching returns *more specific* ($\sqsubseteq$). However, this approach may become prohibitory expensive as there may be thousands and millions of concepts described in documents. In order to allow for the efficient computation of $A(C^q, \mathcal{T})$, we propose a new approach described below.

Let us assume, as it is the case in the current implementation, that $\mathcal{T}$ consists of the terminological knowledge base $\mathcal{T}_{WN}$ generated from WordNet and extended by words (represented as concepts) for which no senses in WordNet are found. One small fragment of $\mathcal{T}_{WN}$ is represented in Figure 3. $\mathcal{T}_{WN}$ can be thought of as an acyclic graph, where links represent subsumption axioms in the form $A_i \sqsubseteq A_j$, with $A_i$ and $A_j$ atomic concepts.

Concepts $C^d$ and $C^q$, are created by translating descriptive phrases into propositional DL formulas (see Section 4.1 for details). The resulting concepts are disjunctions ($\sqcup$) of conjunctions ($\sqcap$) of atomic concepts ($A$) without negation,

[1] This could be easily generalized to any set of semantically related concepts. The impact of this choice onto the system performance is part of the future work.

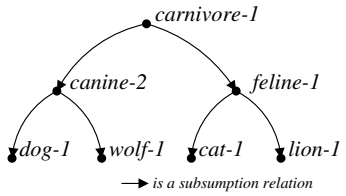


**Fig. 3.** Example of terminological knowledge base $\mathcal{T}_{WN}$

i.e., $C^d \equiv \sqcup \sqcap A^d$ and $C^q \equiv \sqcup \sqcap A^q$. For example, a possible document concept and query concept are:

$$C^d \equiv (little\text{-}4 \sqcap dog\text{-}1) \sqcup (huge\text{-}1 \sqcap cat\text{-}1)$$
$$C^q \equiv (small\text{-}4 \sqcap canine\text{-}2) \sqcup (large\text{-}1 \sqcap feline\text{-}1)$$

By substituting $C^d$ with $\sqcup\sqcap A^d$, $C^q$ with $\sqcup\sqcap A^q$, and $\mathcal{T}$ with $\mathcal{T}_{WN}$ in Equation 5, we obtain:

$$A(\sqcup \sqcap A^q, \mathcal{T}_{WN}) = \{d \mid \exists(\sqcup \sqcap A^d) \in d, \text{ s.t. } \mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q\} \quad (6)$$

Let by $\mathbf{C}_{\sqcup\sqcap A^q}$ denote the set of all complex document concepts $\sqcup \sqcap A^d$ (in all documents $d$), which are equivalent to or more specific than $\sqcup \sqcap A^q$, in formulas

$$\mathbf{C}_{\sqcup\sqcap A^q} = \{\sqcup \sqcap A^d \mid \mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q\} \quad (7)$$

Then Equation 6 can be rewritten as follows:

$$A(\sqcup \sqcap A^q, \mathcal{T}_{WN}) = \{d \mid \exists(\sqcup \sqcap A^d) \in d, \text{ s.t. } (\sqcup \sqcap A^d) \in \mathbf{C}_{\sqcup\sqcap A^q}\} \quad (8)$$

In order to compute set $\mathbf{C}_{\sqcup\sqcap A^q}$, as defined in Equation 7, we need to solve the following subsumption problem

$$\mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \quad (9)$$

Given that $\mathcal{T}_{WN}$ consists only of subsumption axioms between atomic concepts, and that concepts $\sqcup \sqcap A^d$ and $\sqcup \sqcap A^q$ do not contain negations, the problem in Equation 9 can be reduced to the set of subsumption problems

$$\mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq A^q \quad (10)$$

This problem reduction is obtained by sequentially applying the following three equations:[2]

$$\mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \text{ iff for all } \sqcap A^d \text{ in } \sqcup\sqcap A^d, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \quad (11)$$

$$\mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \text{ iff there exists } \sqcap A^q \text{ in } \sqcup\sqcap A^q, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcap A^q \quad (12)$$

$$\mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcap A^q \text{ iff for all } A^q \text{ in } \sqcap A^q, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq A^q \quad (13)$$

Notice that the second part of each equation is the same as the first part of the following equation, and that the first part of Equation 11 and the last part of

[2] Note, that in general, Equation 12 cannot be applied. One such case is when negation is allowed, a counterexample is $\models A_i \sqsubseteq A_j \sqcup \neg A_j$. A second case is when $\mathcal{T}$ contains axioms in the form $A_i \sqsubseteq A_j \sqcup A_j$; consider , e.g., $\mathcal{T} = \{A_i \sqsubseteq A_j \sqcup A_j\} \models A_i \sqsubseteq A_j \sqcup A_j$.

Equation 13 are exactly Equations 9 and 10. This proves that the above problem reduction is correct.

If by $\mathbf{C}^{\sqcap}_{C}$ we denote a set of all conjunctive components $\sqcap A^d$, which are equivalent to or more specific than concept $C$, i.e.,

$$\mathbf{C}^{\sqcap}_{C} = \{\sqcap A^d \mid \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq C\}, \text{ where } C \in \{A^q, \sqcap A^q, \sqcup \sqcap A^q\} \tag{14}$$

Given Equations 11, 12, and 13, the query answer $A(C^q, \mathcal{T}_{WN})$, as defined in 8, can be computed by using the algorithm in the figure below. The five phases

**Algorithm 1** Calculate $A(\sqcup \sqcap A^q, \mathcal{T}_{WN})$

```
1: C^⊓_{⊔⊓A^q} ← ∅
2: for all ⊓A^q in ⊔ ⊓ A^q do                                          ┐
3:    i ← 0                                                            │
4:    C^⊓_{⊓A^q} ← ∅                                                   │
5:    for all A^q in ⊓A^q do                                           │
6:       C^⊓_{A^q} ← {⊓A^d | T_WN ⊨ ⊓A^d ⊑ A^q}   | Phase 1           │
7:       if i = 0 then                                                 │ Phase 2
8:          C^⊓_{⊓A^q} ← C^⊓_{A^q}                                     │
9:       else                                                          │
10:         C^⊓_{⊓A^q} ← C^⊓_{⊓A^q} ∩ C^⊓_{A^q}                        │
11:      end if                                                        │
12:      i ← i + 1                                                     │
13:   end for                                                          │
14:   C^⊓_{⊔⊓A^q} ← C^⊓_{⊔⊓A^q} ∪ C^⊓_{⊓A^q}                          │
15: end for                                                            ┘ Phase 3
16: C_{⊔⊓A^q} ← {⊔ ⊓ A^d | all ⊓A^d in ⊔ ⊓ A^d belong to C^⊓_{⊔⊓A^q}}   | Phase 4
17: A ← {d | there exists ⊔ ⊓ A^d in d, s.t., ⊔ ⊓ A^d belongs to C_{⊔⊓A^q}}   | Phase 5
```

of the algorithm can be explained as follows (the next section reports how these phases are actually implemented and their output on a running example):

**Phase 1** (line 6) We compute $\mathbf{C}^{\sqcap}_{A^q}$, i.e., the set of all $\sqcap A^d$, such that, $\sqcap A^d \sqsubseteq A^q$.

**Phase 2** (lines 5-13) We compute set $\mathbf{C}^{\sqcap}_{\sqcap A^q}$, i.e., a set of all $\sqcap A^d$, such that, $\sqcap A^d \sqsubseteq \sqcap A^q$. As it follows from Equation 13, $\sqcap A^d \in \mathbf{C}^{\sqcap}_{\sqcap A^q}$ only if for every $A^q$ in $\sqcap A^q$, $\sqcap A^d \in \mathbf{C}^{\sqcap}_{A^q}$. To compute the set $\mathbf{C}^{\sqcap}_{\sqcap A^q}$, we intersect sets $\mathbf{C}^{\sqcap}_{A^q}$ for all $A^q$ in $\sqcap A^q$.

**Phase 3** (lines 2-15) We compute the set $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$, i.e., the set of all $\sqcap A^d$, such that, $\sqcap A^d \sqsubseteq \sqcup \sqcap A^q$. As it follows from Equation 12, $\sqcap A^d \in \mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$ if $\sqcap A^d \in \mathbf{C}^{\sqcap}_{\sqcap A^q}$ at least for one $\sqcap A^q$ in $\sqcup \sqcap A^q$. To compute the set $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$, we take the union of all the sets $\mathbf{C}^{\sqcap}_{\sqcap A^q}$ for all $\sqcap A^q$ in $\sqcup \sqcap A^q$.

**Phase 4** (line 16) We compute the set $\mathbf{C}_{\sqcup \sqcap A^q}$, i.e., the set of all complex document concepts $\sqcup \sqcap A^d$, such that, $\sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q$. As it follows from Equation 11, $\sqcup \sqcap A^d \in \mathbf{C}_{\sqcup \sqcap A^q}$ only if all the conjunctive components $\sqcap A^d$ in $\sqcup \sqcap A^d$ belong to $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$. To compute the set $\mathbf{C}_{\sqcup \sqcap A^q}$, we collect all $\sqcup \sqcap A^d$ which consist only from conjunctive components $\sqcap A^d$ in $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$.

**Phase 5** (line 17) We compute the query answer $A(\sqcup \sqcap A^q, \mathcal{T}_{WN})$ as defined in Equation 8, i.e., by collecting all the documents which contain concepts from $\mathbf{C}_{\sqcup \sqcap A^q}$.

## 5 Concept Search via Inverted Indexes

In *C-Search*, every document is represented as an enumerated sequence of conjunctive components $\sqcap A^d$ possibly connected by symbol "$\sqcup$". For example, in Figure 4 we show the sequences of $\sqcap A^d$ extracted from documents in Figure 1. Rectangles in Figure 4 represent either conjunctive components $\sqcap A^d$ or the dis-

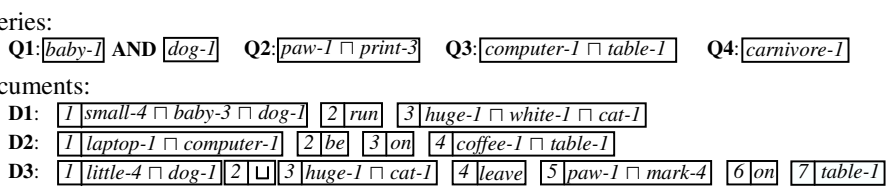


**Fig. 4.** Document and Query Representations

junction symbol "$\sqcup$", a number in a square at the left side of a rectangle represents the *position* of the rectangle in the whole sequence. Note, that symbol "$\sqcup$" is used to specify that conjunctive components $\sqcap A^d$ connected by this symbol form a single disjunctive concept $\sqcup \sqcap A^d$, namely:

$$\sqcup \sqcap A^d ::= (\sqcap A^d)\{(\text{“}\sqcup\text{”})(\sqcap A^d)\} \tag{15}$$

For example, the first three positions in the sequence for document $D3$ in Figure 4 represent the concept $(little\text{-}4 \sqcap dog\text{-}1) \sqcup (huge\text{-}1 \sqcap cat\text{-}1)$.

The resulting document representations (see Figure 4) are indexed using a positional inverted index. In a positional inverted index, as used in syntactic search, there are two parts: the *dictionary*, i.e., a set of terms ($t$) used for indexing; and a set of posting lists P(t). A posting list P(t) is a list of all postings for term $t$:

$$P(t) = [\langle d, freq, [position]\rangle]$$

where $\langle\, d,\ freq,\ [position]\rangle$ is a posting consisting of a document $d$ associated with term $t$, the frequency $freq$ of $t$ in $d$, and a list $[position]$ of positions of $t$ in $d$.

In *C-Search*, we adopt a positional inverted index to index conjunctive components $\sqcap A^d$ by all more general or equivalent atomic concepts from $\mathcal{T}_{WN}$[3]. For example, in Figure 5 we show a fragment of the positional inverted inverted index created by using the document representations in Figure 4. The inverted index dictionary, in *C-Search*, consists of atomic concepts from $\mathcal{T}_{WN}$ (e.g., concepts *baby*-3 and *canine*-2 in Figure 5), and symbol "$\sqcup$" (e.g., the first term in Figure 5). The posting list $P(A)$ for an atomic concept $A$ stores the positions of conjunctive components $\sqcap A^d$, such that, $\sqcap A^d \sqsubseteq A$. For instance, P(*canine-2*) $= [\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$, which means that at first position in documents $D1$

[3] In [7], we showed how searching for complex concepts can be implemented by indexing documents directly by these concepts. The problem of this method is that the size of the inverted index vocabulary, in the worst case, is exponential with respect to size of terminology $\mathcal{T}$. In the current paper, we propose an approach in which we index complex concepts by atomic concepts using the positional information in the inverted index. The size of the vocabulary in this case is the same as the size of $\mathcal{T}$.

| *Dictionary* (t) | Posting lists (P(t)) |
|---|---|
| $\sqcup$ | $[\langle D3, 1, [2]\rangle]$ |
| *baby-3* | $[\langle D1, 1, [1]\rangle]$ |
| *canine-2* | $[\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$ |
| *carnivore-1* | $[\langle D1, 2, [1, 3]\rangle; \langle D3, 2, [1, 3]\rangle]$ |
| *computer-1* | $[\langle D2, 1, [1]\rangle]$ |
| *feline-1* | $[\langle D1, 1, [3]\rangle; \langle D3, 1, [3]\rangle]$ |
| *leave* | $[\langle D3, 1, [4]\rangle]$ |
| *little-4* | $[\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$ |

**Fig. 5.** Positional Inverted Index

and $D3$ there are conjunctive components (i.e., *small*-4 $\sqcap$ *baby*-3 $\sqcap$ *dog*-1 and *little*-4 $\sqcap$ *dog*-1) which are more specific than *canine-2*. The posting list $P(\sqcup)$ stores the positions of the symbol "$\sqcup$".

Now, let us see how Algorithm 1 in Section 4.2 can be implemented by using the positional information of conjunctive components $\sqcap A^d$ stored in the inverted index. Notice that below instead of conjunctive components themselves we work only with their positions.

**Phase 1** Positions of conjunctive components $\sqcap A^d$ in the set $\mathbf{C}^{\sqcap}_{A^q}$ (line 6 in Algorithm 1) are computed by fetching the posting list $P(A^q)$ for an atomic concept $A^q$. For instance (see Figure 5),

$$\mathbf{C}^{\sqcap}_{little\text{-}4} = [\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$$
$$\mathbf{C}^{\sqcap}_{carnivore\text{-}1} = [\langle D1, 2, [1, 3]\rangle; \langle D3, 2, [1, 3]\rangle]$$

**Phase 2** The intersection of the sets of conjunctive components (line 10 in Algorithm 1) is implemented by the intersection of corresponding posting lists. For instance,

$$\mathbf{C}^{\sqcap}_{little\text{-}4 \sqcap carnivore\text{-}1} = [\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$$

**Phase 3** The union of the sets of conjunctive components (line 14 in Algorithm 1) is implemented by uniting corresponding posting lists. For instance,

$$\mathbf{C}^{\sqcap}_{canine\text{-}2 \sqcup feline\text{-}1} = [\langle D1, 2, [1, 3]\rangle; \langle D3, 2, [1, 3]\rangle]$$

**Phase 4** Every concept in set $\mathbf{C}_{\sqcup \sqcap A^q}$ (line 16 in Algorithm 1) should consists only from the conjunctive components in $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$. In order to find the positions of such concepts, we take the union of the posting lists for $\mathbf{C}^{\sqcap}_{\sqcup \sqcap A^q}$ with the posting list for the symbol "$\sqcup$". Then we filter out all the positions which does not comply with the pattern defined in Equation 15. For instance, for complex query concept *canine*-2 $\sqcup$ *feline*-1, we will find the following complex document concepts:

$\langle D1, 1, [1]\rangle \Rightarrow$ $\boxed{1 \mid small\text{-}4 \sqcap baby\text{-}3 \sqcap dog\text{-}1}$ $\quad \langle D1, 1, [3]\rangle \Rightarrow$ $\boxed{3 \mid huge\text{-}1 \sqcap white\text{-}1 \sqcap cat\text{-}1}$

$\langle D3, 1, [1, 2, 3]\rangle \Rightarrow$ $\boxed{1 \mid little\text{-}4 \sqcap dog\text{-}1}$ $\boxed{2 \mid \sqcup}$ $\boxed{3 \mid huge\text{-}1 \sqcap cat\text{-}1}$

**Phase 5** The query answer (line 17 in Algorithm 1) is computed by collecting the documents from all the postings. For instance,

$$A(\textit{canine}\text{-}2 \sqcup \textit{feline}\text{-}1, \mathcal{T}_{WN}) = \{D1, D3\}$$

If $n$ is a number of atomic concepts $A^q$ in the query concept $\sqcup \sqcap A^q$, then to compute $A(C^q, \mathcal{T}_{WN})$ it takes $n$ posting list merges (i.e., intersections and unions). Note that, in a positional inverted index, the same number of posting list merges is required to process a phrase query consisting of $n+1$ words [13].

In *C-Search*, query concepts $C^q$ can be combined into more complex queries $q$ by using the boolean operators AND and NOT. Query answer $A(q, \mathcal{T}_{WN})$ in this case is computed by recursively applying the following rules:

$$A(q_i \text{ AND } q_j, \mathcal{T}_{WN}) = A(q_i, \mathcal{T}_{WN}) \cap A(q_j, \mathcal{T}_{WN})$$
$$A(q_i \text{ NOT } q_j, \mathcal{T}_{WN}) = A(q_i, \mathcal{T}_{WN}) \ / \ A(q_j, \mathcal{T}_{WN})$$

For instance, the query answer for query *baby*-1 AND *dog*-1 (in Figure 4) is computed as follows: $A(\textit{baby}\text{-}1 \text{ AND } \textit{dog}\text{-}1, \mathcal{T}_{WN}) = A(\textit{baby}\text{-}1, \mathcal{T}_{WN}) \cap A(\textit{dog}\text{-}1, \mathcal{T}_{WN}) = \emptyset \cap \{D1, D3\} = \emptyset$

Relevance of documents, in *C-Search*, is computed by adopting the cosine similarity from the vector space model. Concepts are weighted by *tf-idf* weight measure modified to take into account the semantic relatedness of concepts.

## 6 Evaluation

In order to evaluate our approach, we built two IR systems. The first system is an instantiation of syntactic search and is build on top of Lucene[4], an open source IR toolkit used in many search applications[5]. Standard tokenization and English Snowball stemmer were used for document and query preprocessing. The AND operator was used as a default boolean operator in a query. The second system is a semantics enabled version of Lucene, implemented following the methodology described in Sections 4 and 5.

As a data-set for our experiments, we used the TREC ad-hoc document collection[6] (disks 4 and 5 minus the Congressional Record documents) and three query sets: TREC6 (topics 301-350), TREC7 (topics 351-400) and TREC8 (topics 401-450). Only the title for each topic was used as a query. The whole data-set consists of 523,822 documents and 150 queries. In the evaluation we used the standard IR measures and in particular the mean average precision (MAP) and precision at K (P@K), where K was set to 5, 10, and 15. The average precision for a query is the mean of the precision obtained after each relevant document is retrieved (using 0 as the precision for not retrieved documents which are relevant). MAP is the mean of the average precisions for all the queries in the test collection. P@K is the percentage of relevant documents among the top K ranked documents. MAP is used to evaluate the overall accuracy of IR system,

[4] `http://lucene.apache.org/java/docs/index.html`
[5] `http://wiki.apache.org/lucene-java/PoweredBy`
[6] `http://trec.nist.gov/data/test_coll.html`

while P@K is used to evaluate the utility of IR system for users who only see the top K results.

First, in Table 1, we report the evaluation results for the two systems and further, in figure 6, we provide recall-precision graphs, i.e., we plot precision as a function of recall, for these systems.

**Table 1.** Evaluation results

| **TREC6 (301-350)** | | | | |
|---|---|---|---|---|
| | MAP | P@5 | P@10 | P@15 |
| *Lucene* | 0.1361 | 0.3200 | 0.2960 | 0,2573 |
| *C-Search(Lucene)* | 0.1711(+25.7%) | 0.3920(+22.5%) | 0.3480(+17.6%) | 0.3000(+16.6%) |
| **TREC7 (351-400)** | | | | |
| | MAP | P@5 | P@10 | P@15 |
| *Lucene* | 0.1138 | 0.3560 | 0.3280 | 0.3000 |
| *C-Search(Lucene)* | 0.1375(+20.8%) | 0.4200(+18.0%) | 0.3680(+12.2%) | 0.3427(+14.2%) |
| **TREC8 (401-450)** | | | | |
| | MAP | P@5 | P@10 | P@15 |
| *Lucene* | 0.1689 | 0.4320 | 0.4000 | 0.3573 |
| *C-Search(Lucene)* | 0.2070(+22.6%) | 0.4760(+10.2%) | 0.4280(+7.0%) | 0.4013(+12.3%) |

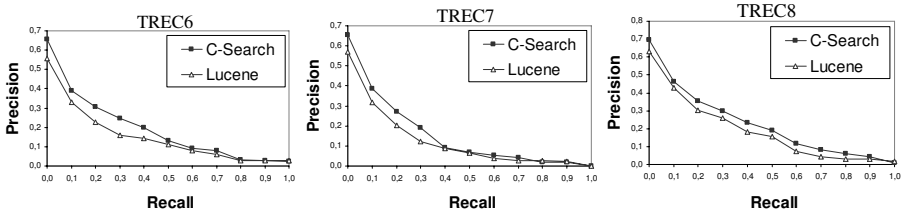


**Fig. 6.** Recall-Precision Graphs

The experiments show that, on TREC ad-hoc data sets, *C-Search* performs better than purely syntactic search, which supports the underlying assumption of our approach. In particular, from Table 1 we observe that *C-Search* improves precision P@K for all K in all three TREC data sets. This is coherent with the intuition that semantics improve on precision. Notice that it means that we are able to show users more relevant documents at the top of the list. From Figure 6 we observe that the recall-precision graphs for *C-Search* are above those for Lucene, which means that the improvement in precision achieved by *C-Search* does not decrease recall.

## 7 Related work

The fact that the syntactic nature of classical IR leads to problems with precision was recognized in the IR community long ago (e.g., see [20]). There were two major approaches to addressing this problem. The first 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 [21]). The second is based on using a linguistic database which is used to associate words in a document corpus with atomic lexical concepts in this database and then to index

these documents with the associated concepts (e.g., see [18]). Our approach is different from both these approaches. In fact, the former approach is still essentially syntactic (and semantics is only implicitly derived with no guarantee of correctness), while in the latter approach only atomic concepts are indexed, whereas *C-Search* allows for indexing of complex concepts and explicitly takes into account possible relations between them. More importantly, our approach *extends* syntactic search and does not replace it as in the latter approach and, therefore, supports the continuum from purely syntactic to fully semantic search.

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 fundamentally different because, similarly to classical IR, the 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]. The matching of document and query representations, in these approaches, is based on query expansion (e.g., see [3]), graph traversal (e.g., see [16]), and RDF reasoning (e.g., see [5]). Differently from these approaches, *C-Search* is based on the semantic matching of *complex concepts*, where semantic matching is implemented by using positional inverted index. Hybrid Search [2] is similar to our approach in that it integrates syntactic search with semantic search. Differently from us, in [2], semantic search is implemented on metadata and is totally separated from syntactic search, implemented on keywords. Instead, in our approach semantic search is seamlessly integrated into syntactic search by substitution words with concepts and at the same time keeping the underlying machinery of syntactic search.

## 8 Conclusion

In this paper we have presented an approach where syntactic search is extended with a semantic layer. The proposed approach performs as good as syntactic search while allowing for an improvement whenever semantics are available and can be exploited. The reported experimental results provide a proof of concept of the quality of proposed solution.

We are still at the beginning and a lot of work still needs to be done. Future work includes:

- the development of more accurate document relevance metrics based on both syntactic and semantic similarity of query and document descriptions;
- the integration of more accurate algorithms for concept identification;
- providing support for queries in which concepts can be associated with a semantic scope such as equivalence, more/less general, disjoint;
- improving the performance of the system. Even if the system is reasonably fast in the sense that it provides answers in much less than a second, namely in a time which is reasonable for the user to wait, it is slower than Lucene and we believe that there is a lot of scope for improving its efficiency.

## References


1. T. Andreasen, P. Anker Jensen, J. Fischer Nilsson, P. Paggio, B. S. Pedersen, and H. E. Thomsen. Content-based text querying with ontological descriptors. *Data & Know. Eng.*, 48(2):199–219, 2004.
2. R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi, and D. Petrelli. Hybrid search: Effectively combining keywords and semantic searches. In *ESWC*, 2008.
3. I. Celino, E. Della Valle, D. Cerizza, and A. Turati. Squiggle: a semantic search engine for indexing and retrieval of multimedia content. In *SEMPS*, 2006.
4. H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.
5. J. Davies and R. Weeks. QuizRDF: Search technology for the semantic web. In *HICSS*, 2004.
6. Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
7. Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu. Concept search: Semantics enabled syntactic search. In *SemSearch2008 workshop at ESWC*, 2008.
8. Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Discovering missing background knowledge in ontology matching. In *Proc. of ECAI*, 2006.
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, CWI, 2007.
11. B. Magnini, M. Speranza, and C. Girardi. A semantic-based approach to interoperability of classification hierarchies: evaluation of linguistic techniques. *COLING'04*.
12. Christoph Mangold. A survey and classification of semantic search approaches. *Int. J. Metadata Semantics and Ontology*, 2(1):23–34, 2007.
13. Christopher Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
14. Dan I. Moldovan and Rada Mihalcea. Using wordnet and lexical operators to improve internet searches. *IEEE Internet Computing*, 4(1):34–43, 2000.
15. Gabor Nagypl. Improving information retrieval effectiveness by using domain knowledge stored in ontologies. *OTM Workshops 2005, LNCS 3762*, 2005.
16. C. Rocha, D. Schwabe, and M. de Aragao. A hybrid approach for searching in the semantic web. In *13th International World Wide Web Conference*, 2004.
17. Mark Sanderson. Retrieving with good sense. *Inf. Retr.*, 2(1):49–69, 2000.
18. Hinrich Schutze and Jan O. Pedersen. Information retrieval based on word senses. In *4th Annual Symposium on Document Analysis and Information Retrieval*, 1995.
19. J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.
20. Christopher Stokoe, Michael P. Oakes, and John Tait. Word sense disambiguation in information retrieval revisited. pages 159–166, 2003.
21. William A. Woods. Conceptual indexing: A better way to organize knowledge. Technical Report TR-97-61, Sun Microsystems Laboratories, USA, 1997.
22. 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.
23. C. Zhai. Fast statistical parsing of noun phrases for document indexing. *Fifth Conference on Applied Natural Language Processing*, pages 312–319, 1997.