CONCEPT SEARCH: SEMANTICS ENABLED INFORMATION RETRIEVAL

Fausto Giunchiglia, Uladzimir Kharkevich, Ilya Zaihrayeu

January 2010

Technical Report # DISI-10-004

Concept Search: Semantics Enabled Information Retrieval [1]

Fausto Giunchiglia, Uladzimir Kharkevich, Ilya Zaihrayeu

Department of Information Engineering and Computer Science, University of Trento, Italy

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.

Key words: Semantic Search, Semantic Inverted Index, Semantic Continuum

1 Introduction

Historically, there have been two major approaches to information retrieval that we call *syntactic search* and *semantic search*. Syntactic search uses words or multi-word phrases as atomic elements in document and query representations. The search procedure is essentially based on the *syntactic matching*

Email addresses: `fausto@disi.unitn.it` (Fausto Giunchiglia), `kharkevi@disi.unitn.it` (Uladzimir Kharkevich), `ilya@disi.unitn.it` (Ilya Zaihrayeu).

[1] This article is an extended and updated version of an earlier conference paper [14], where *Concept Search* was originally introduced.

[2] This work has been partially supported by the European Commission - LIVING KNOWLEDGE project, Grant agreement no.: 231126.

of document and query representations. 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 key 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.

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 useful, 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 full 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. Section 5 describes how semantics enabled relevancy ranking is implemented in *C-Search*. In Section 6, we show how *C-Search* can be efficiently implemented using inverted index technology. Section 7 presents experimental results. In Section 8, we discuss the state-of-the-art in semantic search and compare our approach with other related approaches. Section 9 concludes the paper. In Appendix A, we provide a proof of the correctness and the completeness of the concept matching algorithm which is described in Section 4.

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 to q. *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. 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. Finally, in syntactic search, *Term* and *Match* are instantiated as follows:

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

In the simplest case, syntactic matching is implemented as search for identical 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. Readers interested in a detailed discussion of IR systems and their components are referred to [20].

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 may contain documents where the query word is used in a meaning which is different from what the user had in mind when she was defining the query. 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.

Queries:

Q1: *Babies and dogs* Q2: *Paw print* Q3: *Computer table* Q4: *Carnivores*

Documents:

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 table.*

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 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 complex 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 to $Q3$ 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.

From natural language to formal language (NL2FL-axis in Figure 2). To solve the problems related to the ambiguity of natural language, namely, the

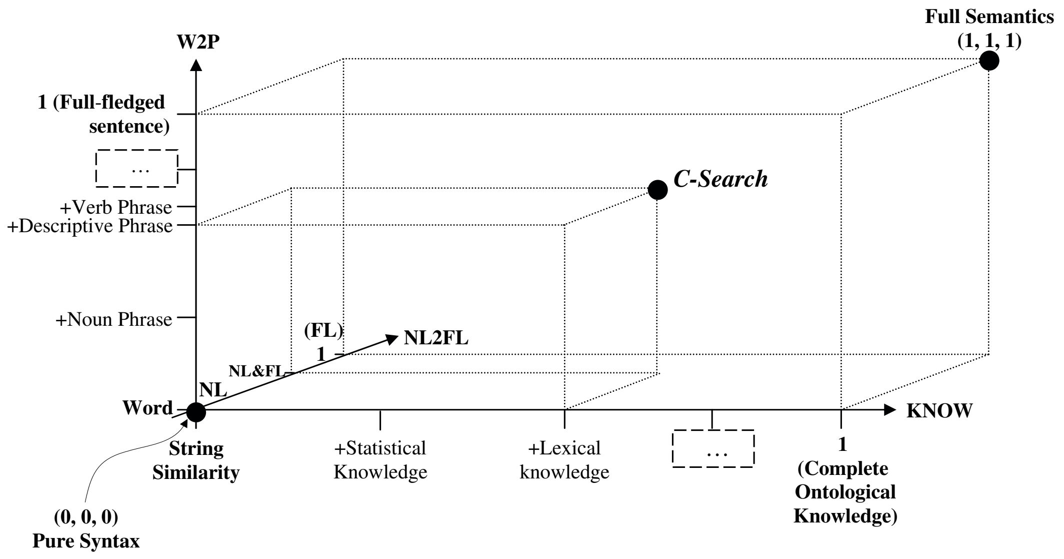

Fig. 2. Semantic Continuum

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 [26]. In the NL2FL-axis, value 0 represents the situation where only words are used, while value 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 [12,16], 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 identifier, 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 [28]. An example of a noun phrase parsing algorithm and its application to document indexing and retrieval is described in [32]. There are approaches in which the conceptual content of noun phrases is also analyzed (e.g., see [1]). In general, concepts can be expressed as more complex phrases than noun phrases (e.g., verb phrases) and possibly as arbitrary complex full-fledged sentences. In the W2P-axis, value 0 represents the situation where only single words are used, while value 1 represents the situation where complex concepts, extracted from full-fledged sentences, 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 (e.g., see [10]), lexical knowledge about synonyms and related words (e.g., see [22]), or ontological knowledge about classes, individuals, and their relationships (e.g., see [23]). In the KNOW-axis, value 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 full semantic search, and all points in between represent search approaches in which semantics is enabled to different extents. The *C-Search* approach can be positioned anywhere in the semantic continuum with the purely syntactic search being its base case, and the full semantic search being the optimal solution, at the moment beyond the available technology.

4 Concept Search

C-Search is implemented according to the model described in 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) [2] language (i.e., a DL language without roles). Complex concepts are computed by analyzing meaning of words and phrases in queries and document bodies.

Single words are converted into atomic concepts uniquely identified as the *lemma-sn*, where *lemma* is the lemma of the word, and *sn* is the sense number in a lexical database such as WordNet [21]. For instance, the word *dog* used in the sense of a domestic dog, which is the first sense in the lexical database, 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 the lexical database. 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 [31], which exploits part-of-speech (POS) tagging information and the lexical database for the disambiguation of words in short noun phrases. Differently from [31] 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 the left senses. If no senses from the lexical database 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 in the phrase. For instance, the noun phrase *A little dog* is translated into the concept $little\text{-}4 \sqcap dog\text{-}1$. Note that in this paper, we adopt the approach described in [15] by defining the extension of a concept as a set of noun phrases which *describe* this concept. For instance, the extension of the concept $little\text{-}4 \sqcap dog\text{-}1$ is the set of all noun phrases about dogs of a small size.

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) that comes from the coordination conjunction *OR* in natural language, we introduce the notion of *descriptive phrase*. We define a 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 the logical disjunction of the formulas corresponding to the noun phrases. For instance, the phrase *A little dog or a huge cat* is translated into the concept $(little\text{-}4 \sqcap dog\text{-}1) \sqcup (huge\text{-}1 \sqcap cat\text{-}1)$. To locate descriptive phrases we, first, follow a standard NLP pipeline to locate noun phrases, i.e., we perform sentence detection, tokenization, POS tagging, and noun phrase chunking. Second, we locate descriptive phrases satisfying Formula 3.

In *C-Search*, every document d is represented as an enumerated sequence of conjunctive components $\sqcap A^d$ (where A^d is an atomic concept from d, e.g., *dog*-1) possibly connected by disjunction symbol "$\sqcup$". For example, in Fig-

Queries:

Q1: $\boxed{\textit{baby-1}}$ **AND** $\boxed{\textit{dog-1}}$ **Q2**: $\boxed{\textit{paw-1} \sqcap \textit{print-3}}$ **Q3**: $\boxed{\textit{computer-1} \sqcap \textit{table-1}}$ **Q4**: $\boxed{\textit{carnivore-1}}$

Documents:

D1: $\boxed{\textit{1}}\boxed{\textit{small-4} \sqcap \textit{baby-3} \sqcap \textit{dog-1}}$ $\boxed{\textit{2}}\boxed{\textit{run}}$ $\boxed{\textit{3}}\boxed{\textit{huge-1} \sqcap \textit{white-1} \sqcap \textit{cat-1}}$

D2: $\boxed{\textit{1}}\boxed{\textit{laptop-1} \sqcap \textit{computer-1}}$ $\boxed{\textit{2}}\boxed{\textit{be}}$ $\boxed{\textit{3}}\boxed{\textit{on}}$ $\boxed{\textit{4}}\boxed{\textit{coffee-1} \sqcap \textit{table-1}}$

D3: $\boxed{\textit{1}}\boxed{\textit{little-4} \sqcap \textit{dog-1}}\boxed{\textit{2}}\boxed{\sqcup}\boxed{\textit{3}}\boxed{\textit{huge-1} \sqcap \textit{cat-1}}$ $\boxed{\textit{4}}\boxed{\textit{leave}}$ $\boxed{\textit{5}}\boxed{\textit{paw-1} \sqcap \textit{mark-4}}$ $\boxed{\textit{6}}\boxed{\textit{on}}$ $\boxed{\textit{7}}\boxed{\textit{table-1}}$

Fig. 3. Document and Query Representations

ure 3 we show the sequences of $\sqcap A^d$ extracted from documents in Figure 1. Rectangles in Figure 3 represent either conjunctive components $\sqcap A^d$ or the disjunction 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:

$$\sqcup \sqcap A^d ::= (\sqcap A^d)\{(\text{“} \sqcup \text{”})(\sqcap A^d)\} \tag{4}$$

For example, the first three positions in the sequence for document $D3$ in Figure 3 represent the concept $(\textit{little}\text{-}4 \sqcap \textit{dog}\text{-}1) \sqcup (\textit{huge}\text{-}1 \sqcap \textit{cat}\text{-}1)$.

Queries usually are short phrases (i.e., 1-3 words) and, as shown in [31], 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 [31]. 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 $A^q_1, \ldots, A^q_n$, extracted from short queries, should be combined in order to build complex query concepts. To represent this uncertainty we use the following query:

$$(A^q_1 \text{ AND } \ldots \text{ AND } A^q_n) \text{ AND } (A^q_1 \sqcup \cdots \sqcup A^q_n) \tag{5}$$

where the first part $(A^q_1 \text{ AND } \ldots \text{ AND } A^q_n)$, i.e., atomic concepts $A^q_1, \ldots, A^q_n$ connected by using boolean operator AND, 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* to which complex concept (e.g., $A_1^q \sqcap A_2^q$, or $A_1^q \sqcap \cdots \sqcap A_n^q$) the documents in the query answer should actually be relevant to. For instance, for queries *cat huge* and *dog cat* the following C-Search queries will be generated:

$$\textit{cat huge} \Rightarrow \textit{cat}\text{-}1 \text{ AND } \textit{huge}\text{-}1 \text{ AND } \textit{cat}\text{-}1 \sqcup \textit{huge}\text{-}1 \sqcup (\textit{cat}\text{-}1 \sqcap \textit{huge}\text{-}1)$$
$$\textit{dog cat} \Rightarrow \textit{dog}\text{-}1 \text{ AND } \textit{cat}\text{-}1 \text{ AND } \textit{dog}\text{-}1 \sqcup \textit{cat}\text{-}1 \sqcup (\textit{dog}\text{-}1 \sqcap \textit{cat}\text{-}1)$$

Note that in *C-Search* (as it will be discussed later in Section 5) we give a preference to documents which match complex concepts, therefore, in the first example, documents about *cat*-1$\sqcap$*huge*-1 will be ranked higher. Let us assume that in the second example there are no documents about complex concept *dog*-1 $\sqcap$ *cat*-1. In this case, it can be shown that the query results will be the same as the results of the query *dog*-1 AND *cat*-1.

4.2 From Word to Concept Matching (WMatch → 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 [3] . 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 encoded by the latter concept. Formally a query answer $A(C^q, \mathcal{T})$ is defined as follows:

$$A(C^q, \mathcal{T}) = \{d \mid \exists C^d \in d, \text{ s.t. } \mathcal{T} \models C^d \sqsubseteq C^q\} \qquad (6)$$

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 6 states that the answer to a query concept C^q is the set of all documents d, such that, there exists concept 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 [17], and collect those C^d for which semantic matching returns *more specific* ($\sqsubseteq$). However,

[3] 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.

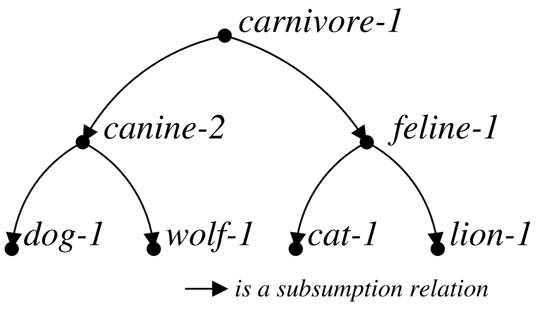

Fig. 4. Example of terminological knowledge base $\mathcal{T}_{WN}$

this approach may become prohibitory expensive as there may be thousands and millions of concepts described in documents. In order to allow for a more efficient computation of $A(C^q, \mathcal{T})$, we propose an 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 4. $\mathcal{T}_{WN}$ 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, i.e., $C^d \equiv \sqcup \sqcap A^d$ and $C^q \equiv \sqcup \sqcap A^q$. For example, possible document and query concepts 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 6, we obtain:

$$A(\sqcup \sqcap A^q, \mathcal{T}_{WN}) = \{d \mid \exists(\sqcup \sqcap A^d) \in d, \text{ s.t. } \mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q\} \quad (7)$$

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

$$\mathbf{C}_{\sqcup\sqcap A^q} = \{\sqcup \sqcap A^d \mid \mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q\} \quad (8)$$

Then Equation 7 can be rewritten as follows:

$$A(\sqcup \sqcap A^q, \mathcal{T}_{WN}) = \{d \mid \exists(\sqcup \sqcap A^d) \in d, \text{ s.t. } (\sqcup \sqcap A^d) \in \mathbf{C}_{\sqcup\sqcap A^q}\} \quad (9)$$

In order to compute set $\mathbf{C}_{\sqcup\sqcap A^q}$, as defined in Equation 8, we need to solve the following subsumption problem

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

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 10 can be reduced to the set of subsumption problems

$$\mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq A^q \tag{11}$$

This problem reduction is obtained by applying the following three equations[4]:

$$\mathcal{T}_{WN} \models \sqcup \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \text{ iff for all } \sqcap A^d \text{ in } \sqcup \sqcap A^d, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \tag{12}$$

$$\mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcup \sqcap A^q \text{ iff there exists } \sqcap A^q \text{ in } \sqcup \sqcap A^q, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcap A^q \tag{13}$$

$$\mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq \sqcap A^q \text{ iff for all } A^q \text{ in } \sqcap A^q, \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq A^q \tag{14}$$

Notice that the second part of each equation is the same as the first part of the equation that follows, and that the first part of Equation 12 and the last part of Equation 14 are exactly Equations 10 and 11. This proves that the above problem reduction is correct.

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

$$\mathbf{C}^{\sqcap}_{C} = \{\sqcap A^d \mid \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq C\}, \text{ where } C \in \{A^q, \sqcap A^q, \sqcup \sqcap A^q\} \tag{15}$$

Given Equations 12, 13, and 14, the query answer $A(C^q, \mathcal{T}_{WN})$, as defined in Equation 9, can be computed by using Algorithm 1. The algorithm consists of the five principle phases which are described below:

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 $\mathbf{C}^{\sqcap}_{\sqcap A^q}$, i.e., a set of all $\sqcap A^d$, such that, $\sqcap A^d \sqsubseteq \sqcap A^q$. As it follows from Equation 14, $\sqcap A^d \in \mathbf{C}^{\sqcap}_{\sqcap A^q}$ only if for every A^q in $\sqcap A^q$, $\sqcap A^d \in \mathbf{C}^{\sqcap}_{A^q}$. To compute $\mathbf{C}^{\sqcap}_{\sqcap A^q}$, 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 13, $\sqcap A^d \in \mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q}$ 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 12, $\sqcup \sqcap A^d \in \mathbf{C}_{\sqcup\sqcap A^q}$ only if all the conjunctive components $\sqcap A^d$

[4] In Appendix A, we prove correctness and completeness of Equation 13 when the knowledge base and complex concepts are as described above. Note, that in general, Equation 13 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 of the form $A_i \sqsubseteq A_j \sqcup A_k$; consider, e.g., $\mathcal{T} = \{A_i \sqsubseteq A_j \sqcup A_k\} \models A_i \sqsubseteq A_j \sqcup A_k$.

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

```
1:  $\mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q} \leftarrow \emptyset$
2:  for all $\sqcap A^q$ in $\sqcup \sqcap A^q$ do
3:      $i \leftarrow 0$
4:      $\mathbf{C}^{\sqcap}_{\sqcap A^q} \leftarrow \emptyset$
5:      for all $A^q$ in $\sqcap A^q$ do
6:          $\mathbf{C}^{\sqcap}_{A^q} \leftarrow \{\sqcap A^d \mid \mathcal{T}_{WN} \models \sqcap A^d \sqsubseteq A^q\}$ |Phase 1
7:          if i = 0 then
8:              $\mathbf{C}^{\sqcap}_{\sqcap A^q} \leftarrow \mathbf{C}^{\sqcap}_{A^q}$
9:          else
10:             $\mathbf{C}^{\sqcap}_{\sqcap A^q} \leftarrow \mathbf{C}^{\sqcap}_{\sqcap A^q} \cap \mathbf{C}^{\sqcap}_{A^q}$
11:         end if
12:         $i \leftarrow i + 1$
13:     end for                                          (lines 5–13: Phase 2)
14:     $\mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q} \leftarrow \mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q} \cup \mathbf{C}^{\sqcap}_{\sqcap A^q}$
15: end for                                              (lines 2–15: Phase 3)
16: $\mathbf{C}_{\sqcup\sqcap A^q} \leftarrow \{\sqcup \sqcap A^d \mid$ all $\sqcap A^d$ in $\sqcup \sqcap A^d$ belong to $\mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q}\}$ |Phase 4
17: $A \leftarrow \{d \mid$ there exists $\sqcup \sqcap A^d$ in $d$, s.t., $\sqcup \sqcap A^d$ belongs to $\mathbf{C}_{\sqcup\sqcap A^q}\}$ |Phase 5
```

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 $A(\sqcup \sqcap A^q, \mathcal{T}_{WN})$ as defined in Equation 9, i.e., by collecting all the documents which contain concepts from $\mathbf{C}_{\sqcup\sqcap A^q}$.

The next section reports different approaches to how the phases above can actually be implemented and their output on a running example.

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

$$\begin{aligned} A(q_i \text{ AND } q_j, \mathcal{T}_{WN}) &= A(q_i, \mathcal{T}_{WN}) \cap A(q_j, \mathcal{T}_{WN}) \\ A(q_i \text{ OR } q_j, \mathcal{T}_{WN}) &= A(q_i, \mathcal{T}_{WN}) \cup A(q_j, \mathcal{T}_{WN}) \\ A(q_i \text{ NOT } q_j, \mathcal{T}_{WN}) &= A(q_i, \mathcal{T}_{WN}) \ / \ A(q_j, \mathcal{T}_{WN}) \end{aligned}$$

For instance, the query answer for query *baby*-1 AND *dog*-1 (in Figure 3) is computed as follows: $A(\textit{baby-}1 \text{ AND } \textit{dog-}1, \mathcal{T}_{WN}) = A(\textit{baby-}1, \mathcal{T}_{WN}) \cap A(\textit{dog-}1, \mathcal{T}_{WN}) = \emptyset \cap \{D1, D3\} = \emptyset$

5 Relevance Ranking

In order to compute the relevance of documents, in *C-Search*, standard IR ranking techniques are adapted. In syntactic search, ranking of documents is usually performed by calculating frequencies $f(w^q, d)$ of appearance of query words w^q in a document d and then by applying different scoring functions $S(w^q, d) = S(f(w^q, d), d)$, which depend on $f(w^q, d)$ and, in general, can also depend on many other parameters, e.g., length of d. In order to adapt such techniques for ranking query results in *C-Search*, $f(w^q, d)$ is replaced by the following function:

$$f'(A^q, w^q, d) = P(A^q, w^q) \cdot \sum_{A^d \sqsubseteq A^q} SS(A^q, A^d) \cdot P(A^d, w^d) \cdot f(A^d, w^d, d) \tag{16}$$

which takes into account frequencies $f(A^d, w^d, d)$ of all the atomic document concepts A^d which are related to A^q and described in d. Moreover, $f'(A^q, w^q, d)$ takes into account the fact that not all the atomic concepts A^q are equally important to query concept A^q. To measure this importance the following three parameters are used: $SS(A^q, A^d)$ - a measure of semantic similarity between concepts A^d and A^q; $P(A^q, w^q)$ - a coefficient which is proportional to the probability of that atomic concept A^q is correctly assigned to a word w^q in the query; and $P(A^d, w^d)$ - a coefficient which is proportional to the probability that an atomic concept A^d is correctly assigned to a word w^d in the document. Informally, $f'(A^q, w^q, d)$ is higher for: (i) concepts A^d which are closer in the meaning to the concept A^q, (ii) a concept A^q which is more likely to be a correct sense of a word w^q in a query q, and (iii) concepts A^d which are more likely to be correct senses for words w^d in a document d.

As a measure of semantic similarity $SS(A^q, A^d)$ the following formula is used [5]:

$$SS(A^q, A^d) = \frac{1}{10^{dist(A^q, A^d)}} \tag{17}$$

where $dist(A^q, A^d)$ is a distance between concepts A^q and A^d in the concept hierarchy from $\mathcal{T}_{WN}$. To estimate the coefficients $P(A, w)$, we use the following formula:

$$P(A, w) = \frac{freq(A, w)}{maxFreq(w)} \tag{18}$$

where $freq(A, w)$ is a number provided by WordNet which shows how frequently the specified word w is used to represent the meaning A (incremented by one in order to avoid zero values), $maxFreq(w)$ is a maximum $freq(A, w)$ for w.

As an example, let us compute value $f'(A^q, w^q, d)$ for a concept *canine*-2 in a document $D1$ from Figure 3. The only more specific concept for a concept *canine*-2 in $D1$ is a concept *dog*-1. Given that the distance $dist(canine\text{-}2, dog\text{-}1)$ is equal to one (see Figure 4), $SS(canine\text{-}2, dog\text{-}1)$ will be equal to 10^{-1}.

[5] Note that other measures of semantic similarity (e.g., see [4]) can also be used. It is a part of our future work to analyze the performance of different semantic similarity measures.

Assume that concepts *canine*-2 and *dog*-1 are the only concepts assigned to words *canine* and dog respectively. In this case, $P(\textit{canine}\text{-}2, \textit{canine}) = P(\textit{dog}\text{-}1, \textit{dog}) = 1$. A word *dog* appears in document $D1$ only once, therefore, $f(\textit{dog}\text{-}1, \textit{dog}, D1) = 1$. Finally, $f'(\textit{canine}\text{-}2, \textit{canine}, D1) = 1 * 10^{-1} * 1 * 1 = 10^{-1}$.

Given the scores $S(A^q, d)$ for atomic concepts A^q (computed by a syntactic relevancy ranking technique with $f(w^q, d)$ replaced by $f'(A^q, w^q, d)$), a score for a complex concept $\sqcup \sqcap A^q$ is computed by using the following equation.

$$S(\sqcup \sqcap A^q, d) = \sum_{\sqcap A^q \in \sqcup \sqcap A^q} isAns(\sqcap A^q, d) \cdot size^2(\sqcap A^q) \quad \cdot \sum_{A^q \in \sqcup \sqcap A^q} S(A^q, d) \quad (19)$$

where $isAns(\sqcap A^q, d)$ is a function which has value 1 when document d describes at least one conjunctive component $\sqcap A^d$ which is more specific than the given conjunctive component $\sqcap A^q$ from the query concept $\sqcup \sqcap A^q$; otherwise, it has value 0. Function $size(\sqcap A^q)$ returns the number of atomic concepts in conjunctive component $\sqcap A^q$. Informally, $S(\sqcup \sqcap A^q, d)$ is higher for those documents d which are answers to more complex concepts $\sqcap A^q$ in $\sqcup \sqcap A^q$.

6 Concept Search via Inverted Indexes

In the section, we will show how document representations (e.g., see Figure 3) can be indexed and retrieved by using (different modifications of) inverted indexes.

6.1 Approach 1: C-Search via a Record Level Inverted Index

In this section, we describe how the document representations (see Figure 3) can be indexed and retrieved by using a (record level) inverted index (as it was proposed in [13]). In the 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 the postings for term t:

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

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

In *Approach 1*, inverted indexes are used to:

Dictionary (t)	Posting lists (P(t))
little-4	$[\langle$*little-4*$\sqcap$*dog-1*$, 1\rangle; \langle$*small-4*$\sqcap$*baby-1*$\sqcap$*dog-1*$, 1\rangle]$
dog-1	$[\langle$*little-4*$\sqcap$*dog-1*$, 1\rangle; \langle$*small-4*$\sqcap$*baby-1*$\sqcap$*dog-1*$, 1\rangle]$
huge-1	$[\langle$*huge-1*$\sqcap$*cat-1*$, 1\rangle; \langle$*huge-1*$\sqcap$*white-1*$\sqcap$*cat-1*$, 1\rangle]$
cat-1	$[\langle$*huge-1*$\sqcap$*cat-1*$, 1\rangle; \langle$*huge-1*$\sqcap$*white-1*$\sqcap$*cat-1*$, 1\rangle]$

Fig. 5. Concept $\sqcap$-index

Dictionary (t)	Posting lists (P(t))
little-4$\sqcap$*dog-1*	$[\langle($*little-4*$\sqcap$*dog-1*$) \sqcup ($*huge-1*$\sqcap$*cat-1*$), 1\rangle]$
huge-1$\sqcap$*cat-1*	$[\langle($*little-4*$\sqcap$*dog-1*$) \sqcup ($*huge-1*$\sqcap$*cat-1*$), 1\rangle]$

Fig. 6. Concept $\sqcup$-index

Dictionary (t)	Posting lists (P(t))
small-4$\sqcap$*baby-1*$\sqcap$*dog-1*	$[\langle D1, 1\rangle]$
huge-1$\sqcap$*white-1*$\sqcap$*cat-1*	$[\langle D1, 1\rangle]$
$($*little-4*$\sqcap$*dog-1*$) \sqcup ($*huge-1*$\sqcap$*cat-1*$)$	$[\langle D3, 1\rangle]$

Fig. 7. Document index

(1) index conjunctive components $\sqcap A^d$ by their atomic concepts A^d. We call the resulting index the *concept $\sqcap$-index.* Concept $\sqcap$-index stores a mapping from each atomic concept to a set of all the conjunctive components which contain this concept. In Figure 5, we show a fragment of a concept $\sqcap$-index.

(2) index complex concepts $\sqcup \sqcap A^d$ by their conjunctive components $\sqcap A^d$. We call the resulting index the *concept $\sqcup$-index.* Concept $\sqcup$-index stores a mapping from each conjunctive component $\sqcap A^d$ to a set of complex concepts $\sqcup \sqcap A^d$ which contain this component. In Figure 6, we show a fragment of a concept $\sqcup$-index.

(3) index documents by (complex) concepts described in the documents. We call the resulting index the *document index.* The document index stores a mapping from each (complex) concept to a set of all the documents which describe this concept. In Figure 7, we show a fragment of a document index.

Now we will see how Algorithm 1 in Section 4.2 can be implemented in *Approach 1* given that concept $\sqcap$- and $\sqcup$- indexes as well as document index were constructed.

Phase 1 To compute the set $\mathbf{C}^{\sqcap}_{A^q}$ (line 6 in Algorithm 1), first, we search the knowledge base $\mathcal{T}_{WN}$ for a set $\mathbf{C}^{A}_{A^q}$ of atomic concepts A which are equivalent to or more specific than A^q. For example (see Figure 4),

$$\begin{aligned}\mathbf{C}^{A}_{canine\text{-}2} &= \{canine\text{-}2, dog\text{-}1, wolf\text{-}1, \dots\}\\ \mathbf{C}^{A}_{feline\text{-}1} &= \{feline\text{-}1, cat\text{-}1, lion\text{-}1, \dots\}\\ \mathbf{C}^{A}_{little\text{-}4} &= \{little\text{-}4, \dots\}\end{aligned}$$

Second, we collect all the conjunctive components $\sqcap A^d$ in $\mathbf{C}^{\sqcap}_{A^q}$ by searching in the concept $\sqcap$-index with atomic concepts from $\mathbf{C}^{A}_{A^q}$. For instance (see Figure 5),

$$\begin{aligned}\mathbf{C}^{\sqcap}_{canine\text{-}4} &= \{little\text{-}4 \sqcap dog\text{-}1, small\text{-}4 \sqcap baby\text{-}1 \sqcap dog\text{-}1\}\\ \mathbf{C}^{\sqcap}_{feline\text{-}1} &= \{huge\text{-}1 \sqcap cat\text{-}1, huge\text{-}1 \sqcap white\text{-}1 \sqcap cat\text{-}1\}\\ \mathbf{C}^{\sqcap}_{little\text{-}4} &= \{little\text{-}4 \sqcap dog\text{-}1, small\text{-}4 \sqcap baby\text{-}1 \sqcap dog\text{-}1\}\end{aligned}$$

Phase 2 Compute the set $\mathbf{C}^{\sqcap}_{\sqcap A^q}$ (lines 5-13 in Algorithm 1). For instance,

$$\mathbf{C}^{\sqcap}_{little\text{-}4 \sqcap canine\text{-}1} = \{little\text{-}4 \sqcap dog\text{-}1, small\text{-}4 \sqcap baby\text{-}1 \sqcap dog\text{-}1\}$$

Phase 3 Compute the set $\mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q}$ (lines 2-15 in Algorithm 1). For instance,

$$\begin{aligned}\mathbf{C}^{\sqcap}_{canine\text{-}2 \sqcup feline\text{-}1} =&\{little\text{-}4 \sqcap dog\text{-}1, small\text{-}4 \sqcap baby\text{-}1 \sqcap dog\text{-}1,\\ &huge\text{-}1 \sqcap cat\text{-}1, huge\text{-}1 \sqcap white\text{-}1 \sqcap cat\text{-}1\}\end{aligned}$$

Phase 4 We compute the set $\mathbf{C}_{\sqcup\sqcap A^q}$ (line 16 in Algorithm 1) by searching in the concept $\sqcup$-index with conjunctive components from $\mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q}$. Note that we search only for those concepts $\sqcup \sqcap A^d$ which have all their conjunctive components $\sqcap A^d$ in $\mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q}$ and discard other concepts. For instance (see Figure 6),

$$\begin{aligned}\mathbf{C}_{canine\text{-}2 \sqcup feline\text{-}1} =&\{small\text{-}4 \sqcap baby\text{-}3 \sqcap dog\text{-}1, huge\text{-}1 \sqcap white\text{-}1 \sqcap cat\text{-}1,\\ &(little\text{-}4 \sqcap dog\text{-}1) \sqcup (huge\text{-}1 \sqcap cat\text{-}1)\}\end{aligned}$$

Phase 5 The query answer (line 17 in Algorithm 1) is computed by searching in the document index with complex concepts from $\mathbf{C}_{\sqcup\sqcap A^q}$. For instance (see Figure 7),

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

The described above approach has several potential problems. First, the size of an inverted index dictionary in concept $\sqcup$-index and document index, in the worst case, is exponential with respect to the size of terminology $\mathcal{T}_{WN}$. Second, the search time can be lengthy. If the query concepts A^q are very general, than, in phases 1,4, and 5, the sets $\mathbf{C}^{A}_{A^q}$, $\mathbf{C}^{\sqcap}_{\sqcup\sqcap A^q}$, and $\mathbf{C}_{\sqcup\sqcap A^q}$ can contain many (in principle, all) related concepts. Consequently, the search time, which is growing linearly with the number of related (complex) concepts, can exceed an acceptable limit.

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. 8. Positional Inverted Index

6.2 Approach 2: C-Search via a Word Level Inverted Index

In this section, we describe how the document representations (see Figure 3) can be indexed and retrieved by using a positional (word level) inverted index (as it was proposed in [14]). In a positional inverted index, differently from a record level inverted index, a posting list P(t) additionally contains all the positions of term t within a document.

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

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 *Approach 2*, we adopt a positional inverted index to index conjunctive components $\sqcap A^d$ by all more general or equivalent atomic concepts from $\mathcal{T}_{WN}$. For example, in Figure 8 we show a fragment of the positional inverted index created by using the document representations in Figure 3. The inverted index dictionary, in *Approach 2*, consists of atomic concepts from $\mathcal{T}_{WN}$ (e.g., concepts *baby*-3 and *canine*-2 in Figure 8), and symbol "$\sqcup$" (e.g., the first term in Figure 8). Note that differently from *Approach 1*, the size of the dictionary in this case is the same as the size of $\mathcal{T}_{WN}$. 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 $D1$ 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 in documents.

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 8),

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

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 4. For instance, for complex query concept $canine$-2 $\sqcup$ $feline$-1, we will find the following complex document concepts:

$$\begin{aligned} \langle D1, 1, [1]\rangle &\Rightarrow \boxed{1 \mid small\text{-}4 \sqcap baby\text{-}3 \sqcap dog\text{-}1} \\ \langle D1, 1, [3]\rangle &\Rightarrow \boxed{3 \mid huge\text{-}1 \sqcap white\text{-}1 \sqcap cat\text{-}1} \\ \langle D3, 1, [1, 2, 3]\rangle &\Rightarrow \boxed{1 \mid little\text{-}4 \sqcap dog\text{-}1}\ \boxed{2 \mid \sqcup}\ \boxed{3 \mid huge\text{-}1 \sqcap cat\text{-}1} \end{aligned}$$

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

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

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 [20].

The main problem with *Approach 2*, is that the size of the index can be relatively big. If s is the maximum number of concepts which are assigned to each word after the word sense filtering step and *depth* is the depth of the knowledge base $\mathcal{T}_{WN}$ (i.e., the number of edges on the longest path from the concept to a root concept in $\mathcal{T}_{WN}$), then, in the worst case, a semantic inverted index in *Approach 2* will contain $s * depth$ times more postings than an inverted index in syntactic search for the same document collection. For example, if $depth = 16$ (as it is the case in WordNet) and $s = 3$, then, in the worst case, semantic inverted index will contain 48 times more postings than the syntactic inverted index.

6.3 Approach 3: C-Search with a Minimum Index Size

In this section, we propose *Approach 3* which is a modification of *Approach 2*, such that, the size of an inverted index is minimized.

First, in *Approach 3*, positions of conjunctive components are indexed only by atomic concepts which are contained in the conjunctive components $\sqcap A^d$ (and not by all the more general atomic concepts as it was done in *Approach 2*). Algorithm 1, in this case, is implemented in the same way as in *Approach 2* apart from *phase 1*. Now, in *phase 1*, we first search the knowledge base $\mathcal{T}_{WN}$ for a set $\mathbf{C}^{A}_{A^q}$ of atomic concepts A which are equivalent to or more specific than A^q. Second, the 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 lists $P(A^q)$ for atomic concepts A^q in $\mathbf{C}^{A}_{A^q}$ and merging them.

Second, all the atomic concepts which have been assigned for a word, in *Approach 3*, are stored in a single posting (and not in separate postings as it was done in *Approach 2*). An inverted index which supports payloads is used. A payload is metadata that can be stored together with each occurrence of a term[6]. In a positional inverted index with payloads, a posting list P(t) is represented as follows:

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

where *payload* is a sequence of bytes of an arbitrary length which is associated with the term t and which can be used to codify additional information about t at the position *position* in the document d.

The inverted index dictionary, in *Approach 3*, consists only of word lemmas (as in syntactic search) and the symbol "$\sqcup$". Payloads are used to store sense

[6] `http://lucene.apache.org/java/2_4_0/api/org/apache/lucene/index/Payload.html`

numbers *sn* in WordNet. Each payload is seen as a bit array, where the size of the array is equal to the number of possible senses for a given lemma and the positions of bits which are set to one are used to represent active sense numbers *sn* (note that in general all the bits can be set to one if no senses were filtered out). For instance, if we take the word *dog* which has 7 senses in WordNet, then the posting list $P(dog)$ created by using the document representations in Figure 3 will be as follows:

$$P(dog) = [\langle D1, 1, [1, 1000000] \rangle; \langle D3, 1, [1, 1000000] \rangle]$$

During the retrieval, the posting list P(A^q =*lemma-sn*) can be computed, first, by fetching posting list P(*lemma*) and then by filtering out all the positions where *sn* bit is not set to one.

In this approach, the size of the inverted index dictionary as well as the number of postings are almost the same as in the inverted index in syntactic search (note that the symbol "$\sqcup$" is the only additional term). Payloads take some additional space, but it can also be minimized. For instance, we can store a payload in a posting only when its value is different from the one in the preceding posting and use the payload value from the preceding posting otherwise.

$$P(dog) = [\langle D1, 1, [1, 1000000] \rangle; \langle D3, 1, [1] \rangle]$$

Here the assumption is that the same word tends to have the same meaning within the same document [11]. If it is the case, then only a few additional bytes will be stored for each document.

Similarly to *Approach 1*, the potential problem of *Approach 3* is that the search time can increase if we search for a very general atomic concept A^q. Note, however, that differently from *Approach 1*, we need to consider only related atomic concepts, and not all the complex concepts, which, in the worst case, are exponentially many.

6.4 Approach 4: C-Search with a Hybrid Index

Approach 3 can perform well if atomic concepts A^q in a query have only a few more specific concepts in $\mathcal{T}_{WN}$, but it can become inefficient otherwise. In Table 1, we show a statistics for a number of more specific concepts in WordNet. As we can observe from Table 1, only 909 out of 100303 concepts (i.e., less than 1%) are very general, i.e., have more than 100 more specific atomic concepts. For instance, concepts *mammal-1*, *animal-1*, and *entity-1* have 9472, 12953, and 76439 more specific atomic concepts respectively. These 909 concepts form a tree (where $\top$ is a root) which we call a *'cap'* of the knowledge base.

Table 1
Statistics for the number of more specific concepts

Number **N** of more specific concepts	Number of concepts with **N** more specific concepts	Number of concepts (%)
N<=10	95264	94.98
10<N<=100	4130	4.12
100<N<=1000	745	0.74
1000<N<=10000	136	0.13
10000<N<=100000	28	0.03

In this section, we propose *Approach 4* which combines Approaches 2 and 3, where *Approach 2* is used only for concepts in the *cap* and *Approaches 3* is used for the rest of the concepts. Let us consider how it is done in detail. First, all the concepts are indexed by using *Approach 2*. Second, each concept which is not in the cap is additionally indexed by the most specific concept(s) from the cap, which is more general than the given concept.

During the retrieval, in order to compute the posting list $P(A^q)$ for a concept A^q which is in the cap, first, we follow *Approach 3*, where the set $\mathbf{C}^A_{A^q}$ consists only of atomic concepts A which are equivalent to or more specific than A^q and are in the cap. Second, we follow *Approach 2* by using the most specific atomic concepts from $\mathbf{C}^A_{A^q}$ (i.e., we use only those concepts which don't have more general concepts in $\mathbf{C}^A_{A^q}$). The results of both approaches are then merged. For concepts which are not in the cap we just follow *Approach 3*. Note that for all the concepts inside/outside the cap *Approach 3* is used for a relatively small number of atomic concepts.

If s is the number of senses, then in the worst case the index will contain $2 * s$ times more postings (and not $s * depth$ as in *Approach 2*) than in the syntactic search approach. Moreover, the search time in *Approach 4* can be always kept relatively small for both very specific and very general concepts (which is not the case in *Approach 3*).

6.5 Approach 5: Approximated C-Search

As it was discussed in Section 5, only those atomic document concepts A^d are scored high which are not very distant from the query concepts A^q (see Formula 16) in the concept hierarchy of $\mathcal{T}_{WN}$. Therefore, if we use only the closest concepts, the quality of results returned by *C-Search* should not be affected much. Moreover, as it was discussed in Sections 6.1-6.4, by using fewer related concepts, we can decrease the search time.

Approach 5 is a modified version of *Approach 3* which approximates the results of *C-Search* by using not all but only more specific concepts within distance *dist* from the atomic concept A^q. Also, in *Approach 5*, we limit the number of atomic concepts which can be assigned for each word in a query, by selecting only the s most probable senses. The influence of parameters *dist* and s on a quality and a performance of *C-Search* is discussed in Section 7.3.

7 Evaluation

In order to evaluate the proposed approaches, we built six IR systems. One system is an instantiation of syntactic search and is build on top of Lucene [7], an open source IR toolkit used in many search applications [8]. 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 Lucene default implementation of the cosine similarity from the vector space model was used for relevancy ranking [9]. Other five systems are semantics enabled versions of Lucene, implemented following the approaches described in Sections 6.1-6.5. WordNet 2.1 was used as a lexical database in all these systems. GATE [8] was used in order to locate descriptive phrases (see Section 4.1). Relevancy ranking, in these systems, was implemented by modifying Lucene default score function[9] as it was described in Section 5.

7.1 Quality Evaluation

In this section, we compared the quality of results returned by Lucene and by *C-Search. Approach 3* was used for this evaluation but, given that the approaches 1, 2 and 4 implement the same algorithm, the results returned by these approaches should be comparable. As a data-set for our experiments, we used the TREC ad-hoc document collection [10] (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

[7] `http://lucene.apache.org/java/docs/index.html`
[8] `http://wiki.apache.org/lucene-java/PoweredBy`
[9] `http://lucene.apache.org/java/2_4_0/api/org/apache/lucene/search/Similarity.html`
[10] `http://trec.nist.gov/data/test_coll.html`

Table 2
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%)

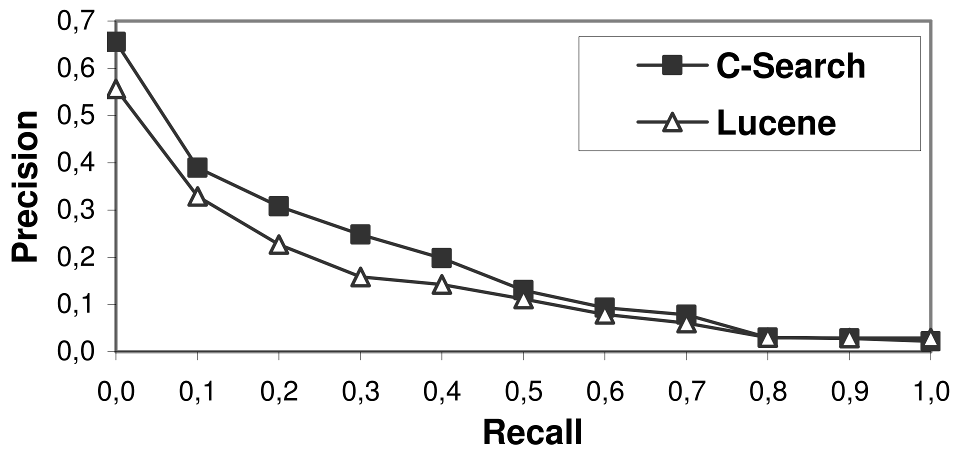

Fig. 9. Recall-Precision Graph (TREC6)

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

First, in Table 2, we report the evaluation results for the two systems and further, in Figures 9,10, and 11 we provide recall-precision graphs, i.e., we plot precision as a function of recall, for these systems.

The experiments show that, on TREC ad-hoc data sets, *C-Search* performs better than the purely syntactic search, which supports the underlying assumption of our approach. In particular, from Table 2 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 to the users more relevant documents at the top of the list. From Figures 9, 10, and 11, we observe that the recall-

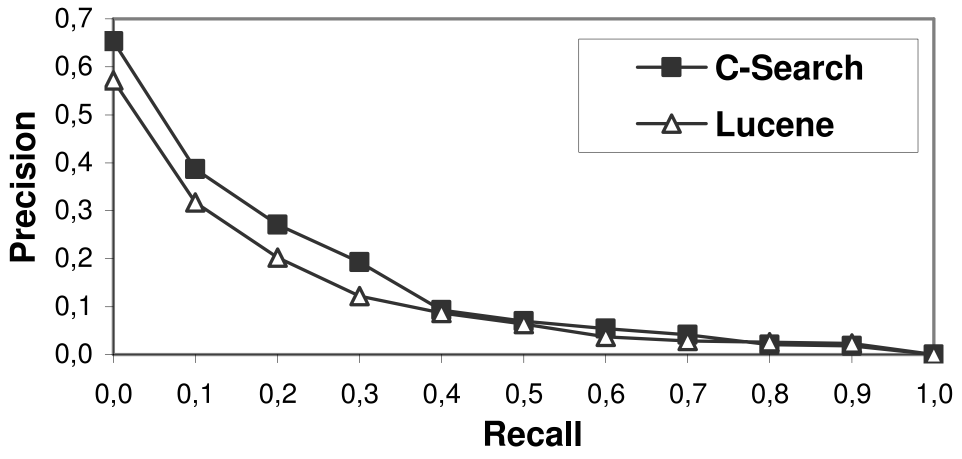

Fig. 10. Recall-Precision Graph (TREC7)

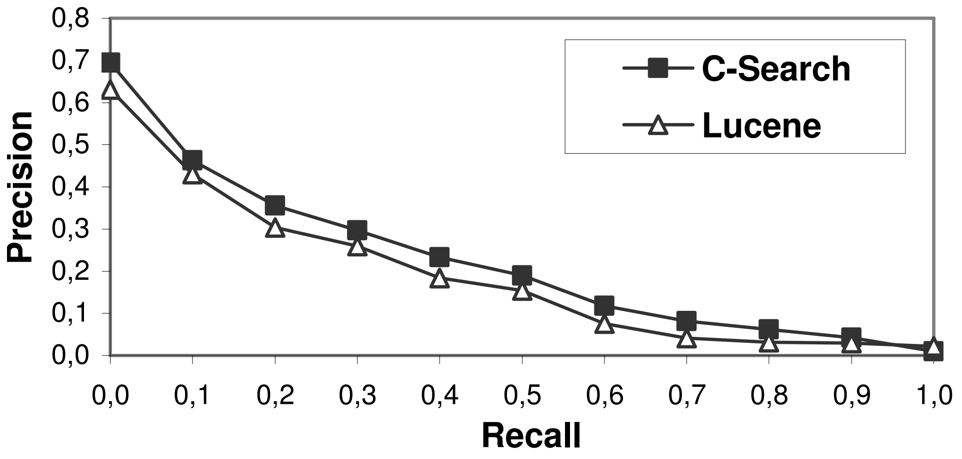

Fig. 11. Recall-Precision Graph (TREC8)

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.

7.2 Performance Evaluation

In this section, we compared an index size and a search time of different versions of *C-Search*. TREC was used as a document collection. Three query sets, with queries consisting of: (i) 1 word, (ii) 2 words, and (iii) 3 words were generated by randomly selecting a set of 1000 queries from the AOL query log [24] for each query set. Queries which contain punctuation, special symbols, or boolean operators (e.g., '+', ' ', and '?'); queries which contain the words shorter than 3 letters; and queries which didn't have any results were filtered out. All the experiments described in this section were run on a machine with the following parameters:

- CPU : Intel(R) Core(TM)2 Duo T7500 @2.20GHz

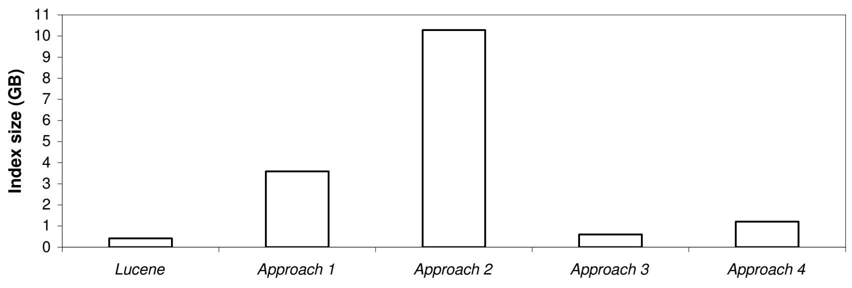

Fig. 12. Size of the inverted index

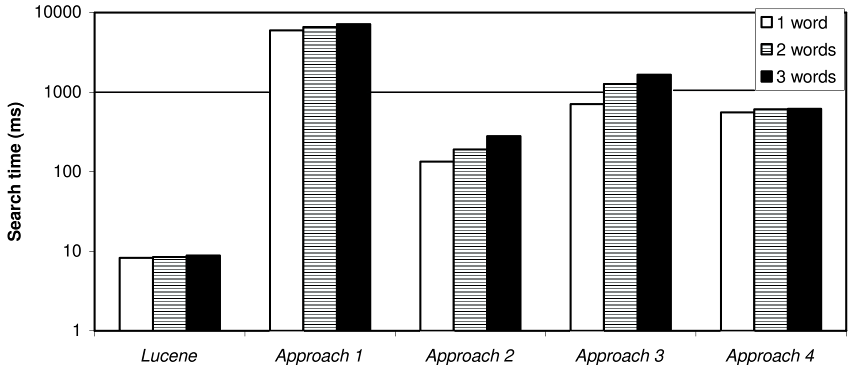

Fig. 13. Search time

- RAM : 3GB
- HD : 250GB @ 5400 RPM
- OS : Windows XP (SP3)

In Figure 12, we report on the size of the inverted indexes created by *Lucene* and by *C-Search* (approaches 1-4). In Figure 13, we report an average search time per query in milliseconds (ms) [11] . As we can observe from Figure 13, *Approach 2* is the the fastest among of *C-Search* approachs. It can provide answers in less than a second, namely in a time which is acceptable for the user to wait. The main reason why *Approach 2* is still much slower than *Lucene* is that, in *C-Search*, we need to analyze positions of atomic concepts and not just the number of their occurrences. Note that the number of positions which need to be analyzed can be much bigger than the number of relevant documents (especially for a very general query concept). The large size of the index in Approach 2 (see Figure 12) is also due to the large amount of positions which need to be stored (see Section 6.2 for details). On the contrary, in *Approach 3*, the size of the index is the smallest among all the other *C-Search* approaches and the average search time is within a few seconds. *Approach 4* improves the search time of *Approach 3* at the cost of doubling its index size.

[11] Every experiment was run 5 times and the average result was reported.

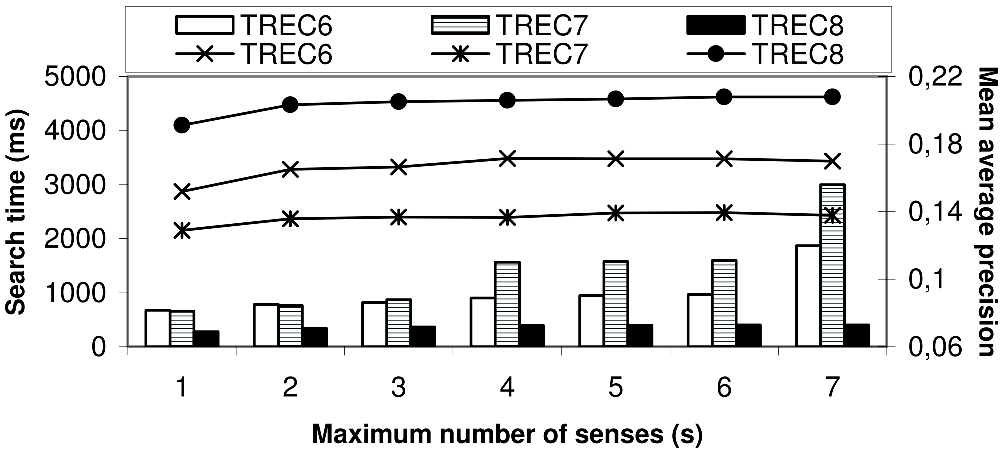

Fig. 14. Influence of a max number of senses for a word on a search time and MAP

7.3 Quality vs. Performance

In this section, we studied the influence of the following two parameters on the quality of results and performance of *Approach 5* (see Section 6.5): (i) s - a maximum number of senses which can be assigned to a word in a query after the word sense filtering step; and (ii) *dist* - a maximum allowed distance between atomic concepts in a query and a document. As a data-set, we used the data-set described in Section 7.1.

Figure 14 shows the influence of the parameter s on the search time (represented as bars) and MAP (represented as broken lines) for *Approach 5* on the three query sets: TREC6, TREC7, and TREC8. As we can see from Figure 14, the quality of results (measured by MAP) returned by the approximated version of *C-Search* is not decreasing much if we use s equal to or bigger than three. At the same time, the search time decreases substantially, namely, if s is reduced from 7 to 3, the search time becomes three times smaller on the TREC7 query set. In Figure 15, we show how the search time and MAP for *Approach 5* are influenced by the parameter *dist* (where s was set to 3). As we can see from Figure 15, the MAP remains almost constant if we keep *dist* equal or bigger than three. If *dist* is set to 3, the search time is decreased around two times, with respect to the case when *dist* is not limited. In total, by using $s = 3$ and $dist = 3$, *Approach 5* can perform 6 times faster than *Approach 3*, while having almost no decrease in the quality of results measured by MAP.

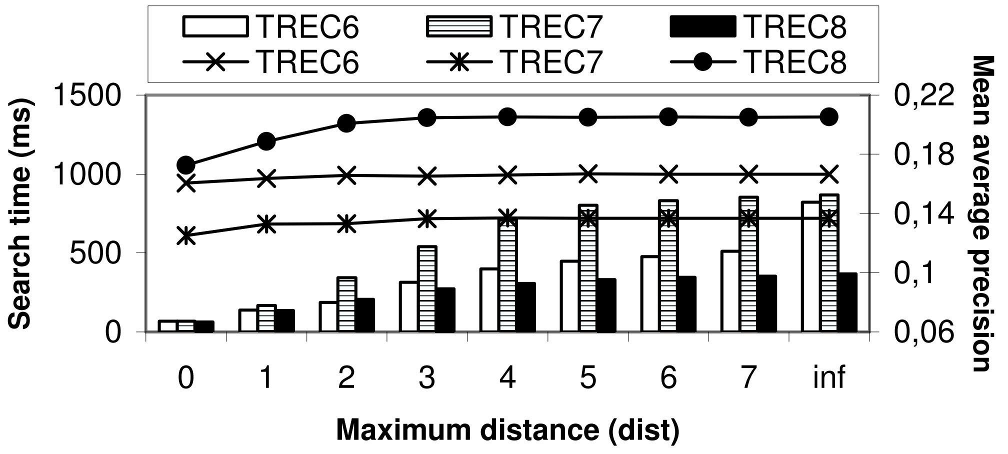

Fig. 15. Influence of a max distance between concepts on a search time and MAP

8 Related work

Semantic search is a research topic that has its roots in the information retrieval (IR) community that proposed first approaches to extending the classical IR with explicit semantics long time ago (e.g., see [7]). Since then many approaches were proposed by the community; however, their core common idea to codify the explicit semantics was in the use of informal knowledge representation structures such as thesauri with no or little formal reasoning support. On the other hand, with the advent of the Semantic Web (SW), many formal frameworks to represent and reason about knowledge were proposed by the SW community with a reasonable level of success in data retrieval applications which, however, required from the user the knowledge of a formal query language. In the recent years, semantic search is gaining an increasing popularity within the SW community (and not only) as a research topic aimed at the exploration of approaches that would leverage the advances of technologies developed by the two communities (see [18,19] for an overview of approaches proposed so far). The work presented in this paper tackles exactly this line of research.

The fact that the syntactic nature of classical IR leads to problems with precision was recognised by the IR community a long time ago (e.g., see [29]). 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 organised in a subsumption hierarchy which is then used to improve the precision of retrieval (e.g., see [30]). The second is based on using a linguistic database in order 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 [27]). 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 and reasoning about complex concepts and explicitly takes into account existing relations between them. More importantly, our approach *extends* syntactic search and does not replace it as the latter approach does and, therefore, supports the continuum from purely syntactic to fully semantic search.

In the SW 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. Our approach is fundamentally different because, similarly to the classical IR, the input query is matched with document contents and not with elements of a knowledge representation structure. The matching of document and query representations, in these approaches, is based on the query expansion (e.g., see [6]), graph traversal (e.g., see [25]), and RDF reasoning (e.g., see [9]). 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.

To the best of our knowledge, there are few approaches that are based on similar ideas to those of *C-Search*. For example, Hybrid Search [3] is similar to *C-Search* in that it combines syntactic search with semantic search. Differently from us, in [3], semantic search is implemented on metadata and is totally separated from syntactic search, implemented on keywords. Another approach, reported in [5], uses classes and instances of an RDF ontology to annotate documents in order to combine ontology-based search with the classical IR. Our approach is different from both [3] and [5] in that it is based on a seamless *integration* of syntactic and semantic kinds of search within a single solution enabled by the proven IR technology based on inverted indexes. In a sense, instead of using a reasoning engine to enable semantics, we integrated semantic reasoning within an inverted index, by taking advantage of the simplifying assumptions that we made about the ontologies used to enable the semantic search (see Section 4.2). Finally, to the best of our knowledge, our approach is the first one that proposes and explains how semantic indexing and retrieval can be performed on complex concepts and not on atomic ones (i.e., for which there is a single class or an instance in the ontology).

9 Conclusion

In this paper we presented a novel approach to IR in which syntactic search is extended with a layer of semantics which enables semantic searching still fully reusing the proven IR technologies such as the inverted index. Noteworthy, the approach is tolerant to the lack of knowledge encoded in the underlying ontology that enables the semantic search, and it improves gracefully as more knowledge becomes available. We demonstrated the proof of concept for the proposed approach by reporting the results of the experiments conducted in different settings. The experiments showed that our approach allowed us to reach noteworthy better results than the classical IR approach. While the first results are promising, more research needs to be done in order to check how much the approach scales in terms of the size of underlying ontology, to develop and compare new semantics-aware document relevant metrics, to explore new natural language processing algorithms and heuristics in order to improve the accuracy of concept identification in queries and in the document corpus.

References

[1] T. Andreasen, P. A. Jensen, J. F. Nilsson, P. Paggio, B. S. Pedersen, H. E. Thomsen, Content-based text querying with ontological descriptors, Data & Know. Eng. 48 (2) (2004) 199–219.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider, The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2003.

[3] R. Bhagdev, S. Chapman, F. Ciravegna, V. Lanfranchi, D. Petrelli, Hybrid search: Effectively combining keywords and semantic searches, in: ESWC, 2008.

[4] A. Budanitsky, G. Hirst, Evaluating wordnet-based measures of lexical semantic relatedness, Computational Linguistics 32 (1) (2006) 13–47.

[5] P. Castells, M. Fernández, D. Vallet, An adaptation of the vector-space model for ontology-based information retrieval, IEEE Trans. Knowl. Data Eng. 19 (2) (2007) 261–272.

[6] I. Celino, E. D. Valle, D. Cerizza, A. Turati, Squiggle: a semantic search engine for indexing and retrieval of multimedia content, in: SEMPS, 2006.

[7] W. B. Croft, User-specified domain knowledge for document retrieval, in: SIGIR, 1986.

[8] H. Cunningham, D. Maynard, K. Bontcheva, 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.

[9] J. Davies, R. Weeks, QuizRDF: Search technology for the semantic web, in: HICSS, 2004.

[10] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, R. A. Harshman, Indexing by latent semantic analysis, Journal of the American Society of Information Science 41 (6) (1990) 391–407.

[11] W. A. Gale, K. W. Church, D. Yarowsky, One sense per discourse, in: HLT '91: Proceedings of the workshop on Speech and Natural Language, 1992.

[12] F. Giunchiglia, B. Dutta, V. Maltese, Faceted lightweight ontologies, in: Conceptual Modeling: Foundations and Applications, 2009.

[13] F. Giunchiglia, U. Kharkevich, I. Zaihrayeu, Concept search: Semantics enabled syntactic search, in: SemSearch2008 workshop at ESWC, 2008.

[14] F. Giunchiglia, U. Kharkevich, I. Zaihrayeu, Concept search, in: ESWC, 2009.

[15] F. Giunchiglia, M. Marchese, I. Zaihrayeu, Encoding classifications into lightweight ontologies, in: Journal on Data Semantics (JoDS) VIII, Winter 2006.

[16] F. Giunchiglia, P. Shvaiko, M. Yatskevich, Discovering missing background knowledge in ontology matching, in: Proc. of ECAI, 2006.

[17] F. Giunchiglia, M. Yatskevich, P. Shvaiko, Semantic matching: Algorithms and implementation., Journal on Data Semantics (JoDS) 9 (2007) 1–38.

[18] M. Hildebrand, J. van Ossenbruggen, L. Hardman, An analysis of search-based user interaction on the semantic web, Tech. Rep. INS-E0706, CWI (2007).

[19] C. Mangold, A survey and classification of semantic search approaches, Int. J. Metadata Semantics and Ontology 2 (1) (2007) 23–34.

[20] C. Manning, P. Raghavan, H. Schtze, Introduction to Information Retrieval, Cambridge University Press, 2008.

[21] G. Miller, WordNet: An electronic Lexical Database, MIT Press, 1998.

[22] D. I. Moldovan, R. Mihalcea, Using wordnet and lexical operators to improve internet searches, IEEE Internet Computing 4 (1) (2000) 34–43.

[23] G. Nagypl, Improving information retrieval effectiveness by using domain knowledge stored in ontologies, OTM Workshops 2005, LNCS 3762.

[24] G. Pass, A. Chowdhury, C. Torgeson, A picture of search, in: InfoScale'06: Proceedings of the 1st international conference on Scalable information systems, ACM, New York, NY, USA, 2006.

[25] C. Rocha, D. Schwabe, M. de Aragao, A hybrid approach for searching in the semantic web, in: 13th International World Wide Web Conference, 2004.

[26] M. Sanderson, Retrieving with good sense, Inf. Retr. 2 (1) (2000) 49–69.

[27] H. Schutze, J. O. Pedersen, Information retrieval based on word senses, in: 4th Annual Symposium on Document Analysis and Information Retrieval, 1995.

[28] J. F. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984.

[29] C. Stokoe, M. P. Oakes, J. Tait, Word sense disambiguation in information retrieval revisited (2003) 159–166.

[30] W. A. Woods, Conceptual indexing: A better way to organize knowledge, Tech. Rep. TR-97-61, Sun Microsystems Laboratories, USA (1997).

[31] I. Zaihrayeu, L. Sun, F. Giunchiglia, W. Pan, Q. Ju, M. Chi, X. Huang, From web directories to ontologies: Natural language processing challenges, in: 6th International Semantic Web Conference (ISWC 2007), Springer, 2007.

[32] C. Zhai, Fast statistical parsing of noun phrases for document indexing, Fifth Conference on Applied Natural Language Processing (1997) 312–319.

A Correctness and Completeness

In order to show that Algorithm 1 in Section 4.2 is sound and complete, we need to prove the following theorem:

Theorem A.1 *Let A' and B' be atomic concepts, and $\mathcal{T}_{WN}$ be a terminological knowledge base which can be represented as an acyclic graph, where nodes are atomic concepts and edges are subsumption axioms in the form $A' \sqsubseteq B'$. Then:*

$$\mathcal{T}_{WN} \models \sqcap A' \sqsubseteq \sqcup \sqcap B' \textit{ iff there exists } \sqcap B' \textit{ in } \sqcup \sqcap B', \textit{s.t.}, \mathcal{T}_{WN} \models \sqcap A' \sqsubseteq \sqcap B' \quad \text{(A.1)}$$

Note that, in Equation A.1, by $\sqcap A$ we denote conjunction ($\sqcap$) of atomic concepts (A) without negation and by $\sqcup \sqcap A$ we denote disjunctions ($\sqcup$) of $\sqcap A$.

Proof It is known, that a subsumption problem with respect to an acyclic terminological knowledge base can be reduced to a subsumption problem with respect to the empty knowledge base [2]:

$$\mathcal{T}_{WN} \models D' \sqsubseteq E' \iff \models D \sqsubseteq E \quad \text{(A.2)}$$

where (complex) concepts D and E are obtained by replacing each occurrence of atomic concept A' in (complex) concepts D' and E' by the conjunction $\sqcap A$ of all atomic concepts A from $\mathcal{T}_{WN}$ which are more general than or equivalent to A'.

Given A.2, we can rewrite Equation A.1 as follows:

$$\models \sqcap A \sqsubseteq \sqcup \sqcap B \text{ iff there exists } \sqcap B \text{ in } \sqcup \sqcap B, \text{s.t.}, \models \sqcap A \sqsubseteq \sqcap B \quad \text{(A.3)}$$

Now, in order to prove Equation A.1, it is enough to prove Equation A.3. In the following we first prove the “if” direction of Equation A.3 and later we demonstrate the proof for the “only if” direction of Equation A.3.

If Recall that disjunction (“$\sqcup$”) is distributive over conjunction (“$\sqcap$”), i.e., if A_1, A_2, and A_3 are concepts than $A_1 \sqcup (A_2 \sqcap A_3) \equiv (A_1 \sqcup A_2) \sqcap (A_1 \sqcup A_3)$. By using the distributive property of disjunction we can convert concept $\sqcup \sqcap B$ from DNF into CNF (we use indexes i,j,k,l in order to enumerate atomic concepts):

$$\sqcup_i \sqcap_j B_{ij} \equiv \sqcap_k \sqcup_l C_{kl} \tag{A.4}$$

Notice, that concepts B_{ij} and C_{kl} in Equation A.4 satisfy the following property:

$$\begin{array}{l} \text{for all the possible combinations } B_1, \ldots, B_I \text{ of atomic concepts } B, \\ \text{where an atomic concept } B_i \text{ is taken from i-th conjunctive clause } \sqcap_j B_{ij} \\ \text{in } \sqcup_i \sqcap_j B_{ij}, \text{there exists disjunctive clause } \sqcup_l C_{kl} \text{ in } \sqcap_k \sqcup_l C_{kl}, \text{ s.t.,} \\ \sqcup_l C_{kl} \text{ is composed from all and only atomic concepts in } \{B_1, \ldots, B_I\}. \end{array} \tag{A.5}$$

Given A.4, subsumption $\models \sqcap A \sqsubseteq \sqcup \sqcap B$ in Equation A.3 can be rewritten as follows:

$$\models \sqcap A \sqsubseteq \sqcap \sqcup C \tag{A.6}$$

A concept can be subsumed by a conjunction of concepts if and only if it is subsumed by every concept in the conjunction:

$$\models \sqcap A \sqsubseteq \sqcap \sqcup C \text{ iff for all } \sqcup C \text{ in } \sqcap \sqcup C, \models \sqcap A \sqsubseteq \sqcup C \tag{A.7}$$

Recall that if A_1 and A_2 are concepts, then:

$$A_1 \sqsubseteq A_2 \text{ iff } A_1 \sqcap \neg A_2 \sqsubseteq \bot \tag{A.8}$$

$$\neg(A_1 \sqcup A_2) \equiv \neg A_1 \sqcap \neg A_2 \tag{A.9}$$

Given A.8 and A.9, subsumption $\models \sqcap A \sqsubseteq \sqcup C$ in Equation A.7 can be rewritten as follows:

$$\models \sqcap A \sqsubseteq \sqcup C \text{ iff } \models (\sqcap A) \sqcap (\sqcap \neg C) \sqsubseteq \bot \tag{A.10}$$

From A.10, it follows that (a) there exists a pair of atomic concepts A and C which have the same name; or (b) there exists an atomic concept $A \equiv \bot$; or (c) there exists an atomic concept $C \equiv \top$. From above, it follows that there exists a pair of atomic concepts A and C, such that, A is more specific than C.

$$\models \sqcap A \sqsubseteq \sqcup C \text{ iff there exists } A \text{ and there exists } C \text{, s.t., } A \sqsubseteq C \tag{A.11}$$

Recall that if at least one concept A in conjunction $\sqcap A$ is subsumed by concept C, then the whole conjunction $\sqcap A$ is also subsumed by C. Taking it into account and using Equation A.11 we can prove that:

$$\models \sqcap A \sqsubseteq \sqcup C \text{ iff there exists } C \text{ in } \sqcup C, \text{s.t.}, \ \models \sqcap A \sqsubseteq C \tag{A.12}$$

Given A.12, second part of Equation A.7 can be rewritten as follows:

$$\boxed{\text{for all } \sqcup C \text{ there exists } C \text{ in } \sqcup C, \text{s.t.}, \ \models \sqcap A \sqsubseteq C} \tag{A.13}$$

Now, let us assume that the "if" direction of Equation A.3 doesn't hold, i.e., concept $\sqcap A$ is not subsumed by any concept $\sqcap B$:

$$\text{for all } \sqcap B \text{ in } \sqcup \sqcap B, \not\models \sqcap A \sqsubseteq \sqcap B \tag{A.14}$$

Recall that a concept can be subsumed by a conjunction of concepts if and only if it is subsumed by every concept in the conjunction:

$$\models \sqcap A \sqsubseteq \sqcap B \text{ iff for all } B \text{ in } \sqcap B, \models \sqcap A \sqsubseteq B \tag{A.15}$$

Given A.15, Equation A.14 can be rewritten as follows:

$$\text{for all } \sqcap B \text{ in } \sqcup \sqcap B \text{ there exists } B \text{ in } \sqcap B \text{ s.t.}, \ \not\models \sqcap A \sqsubseteq B \tag{A.16}$$

Given Property A.5, Equation A.16 can be rewritten as follows:

$$\boxed{\text{there exists } \sqcup C \text{ s.t.}, \text{for all } C \text{ in } \sqcup C \not\models \sqcap A \sqsubseteq C} \tag{A.17}$$

Equation A.17 is in contradiction with Equation A.13. Therefore, we discard the assumption made in Equation A.14, which means that the "if"direction of Equation A.3 holds.

Only-if The union of concepts is more general or equivalent to every concept in the union:

$$\text{for all } \sqcap B \text{ in } \sqcup \sqcap B, \mathcal{T}_{WN} \models \sqcap B \sqsubseteq \sqcup \sqcap B \tag{A.18}$$

Recall that the subsumption is the transitive relation, i.e.,

$$\text{if } \mathcal{T}_{WN} \models \sqcap A \sqsubseteq \sqcap B \text{ and } \mathcal{T}_{WN} \models \sqcap B \sqsubseteq \sqcup \sqcap B, \text{ then } \mathcal{T}_{WN} \models \sqcap A \sqsubseteq \sqcup \sqcap B \tag{A.19}$$

From A.19, we can see that the "only-if" direction of Equation A.3 holds.

Equation A.3 and consequently Theorem A.1 are proved.