UNIVERSITY
OF TRENTO

DIPARTIMENTO DI INGEGNERIA E SCIENZA DELL'INFORMAZIONE

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

P2P CONCEPT SEARCH: SOME PRELIMINARY RESULTS

Fausto Giunchiglia, Uladzimir Kharkevich and S.R.H Noori

March 2009

Technical Report # DISI-09-018

Also: short version is accepted as a poster at the Semantic Search 2009 workshop (SemSearch2009) co- located with
the 18th Int. World Wide Web Conference WWW2009

P2P Concept Search: Some Preliminary Results

Fausto Giunchiglia
Department of Information Engineering and Computer Science
University of Trento, Italy
fausto@disi.unitn.it

Uladzimir Kharkevich
Department of Information Engineering and Computer Science
University of Trento, Italy
kharkevi@disi.unitn.it

S.R.H Noori
Department of Information Engineering and Computer Science
University of Trento, Italy
noori@disi.unitn.it

ABSTRACT

Concept Search 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 complex concepts. It allows us to deal with ambiguity of natural language. P2P Concept Search extends Concept Search by allowing distributed semantic search over structured P2P network. The key idea is to exploit distributed, rather than centralized, background knowledge and indices.

1. INTRODUCTION

The current web is a huge repository of documents, distributed in a network of autonomous information sources (peers). The number of these documents keeps growing significantly from year to year making it increasingly difficult to locate relevant documents while searching on the web. In addition to the massiveness, the web is also a highly dynamic system. Peers are continually joining and leaving the network, new documents are created on peers, and existing ones are changing their content. The search problem becomes even more complex.

Conventional search engines implement search for documents by using *syntactic search*, i.e., words or multi-word phrases are used as atomic elements in document and query representations. The search procedure, in syntactic search, 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. *Concept Search* [3] (*CSearch* in short) 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 with semantic search, thus improving the system performance. As a special case, when no semantic information is available, *CSearch* reduces to syntactic search, i.e., the results produced by *CSearch* and syntactic search are the same.

Nowadays, the major search engines are based on a centralized architecture. They attempt to create a single index for the whole Web. But the size, dynamics, and distributed nature of the Web make the search problem extremely hard, i.e., a very powerful server farm is required to have complete and up-to-date knowledge about the whole network to index it. The peer-to-peer (P2P) computing paradigm appeared as an alternative to centralized search engines for searching web content. Each peer in the P2P network organizes only a small portion of the documents in the network, while being able to access the information stored in the whole network. Robustness and scalability are major advantages of the P2P architecture over the centralized architecture. Also, as the requirements for computational and storage resources of each peer in a P2P network are much lighter than for a server in a centralized approach, a peer's search engine can employ much more advanced techniques for search, e.g. semantic search.

In this paper, we propose an approach called *P2P Concept Search* which extends *CSearch* allowing semantic search on top of distributed hash table (DHT) [12, 16, 14, 21]. In *P2P Concept Search*, centralized document index is replaced by distributed index build on top of DHT. The reasoning with respect to a single background knowledge is extended to the reasoning with respect to the background knowledge distributed among all the peers in the network.

The remainder of the paper is organized as follows. In Section 2, we describe the semantic continuum, a space of approaches lying between purely syntactic search and fully semantic search. We first discuss the syntactic search approach and then we describe different dimensions where semantics can be enabled in syntactic search. In Section 3, we briefly describe *CSearch* and show how it is positioned within the semantic continuum. In Section 4, we discuss how syntactic search can be implemented on top of DHT. In Section 5, we describe how *P2P CSearch* is implemented using DHT technology. In Section 6, we compare our approach with other related approaches. Section 7 concludes the paper.

2. THE SEMANTIC CONTINUUM

The goal of an information retrieval (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

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.*

D4: *The canine population is growing fast.*

Figure 1: Queries and a document collection

therefore be represented as a mapping function:

$$IR : Q \rightarrow D \quad (1)$$

Conventional search engines implement the mapping function in Equation 1 by exploiting syntactic search. A word or a multi-word phrase is used as an atomic element (*term*) in document and query representations. A syntactic matching of words (*match*) is used for matching document and query terms. Syntactic matching is implemented as search for equivalent words, 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. Let us discuss these problems on the example queries and the document collection shown in Figure 1.

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).

Synonymy. Two different words can express the same meaning in a given context and, therefore, query results, computed by a syntactic search engine, may miss documents where the meaning of a query word is expressed by a different word. For instance, a document $D3$ (in Figure 1) which contains word *mark* is relevant to the query $Q2$ (in Figure 1) which contains word *print*, because both words *mark* and *print* are synonymous when used in the sense of a visible indication made on a surface.

Complex concepts. Syntactic search engines fall short of 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.

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* (see query $Q4$ in Figure 1) 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* (e.g., documents $D1$, $D3$, and $D4$ in Figure 1)).

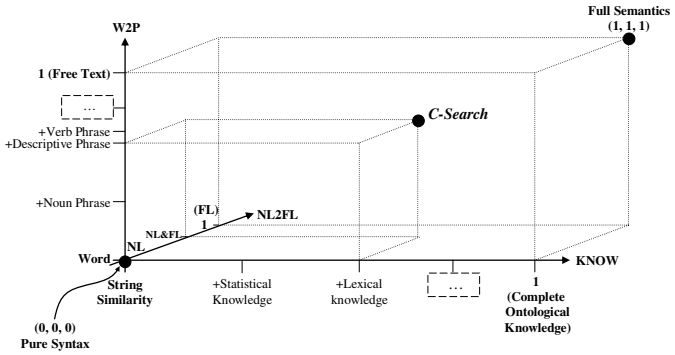

Figure 2: Semantic Continuum

In order to address the problems of syntactic search, we extend syntactic search with semantics. 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. In the following, we briefly discuss different dimensions of the Semantic Continuum (for more details see [3]).

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.

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. Concepts can be expressed as noun phrases, verb phrases, or, in general, a free text.

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 (what we call the *"Background Knowledge" (BK)*). For instance, it can be statistical knowledge about word co-occurrence, lexical knowledge about synonyms and related words, or ontological knowledge about classes, individuals, and their relationships. It is not realistic to assume that a single user can have a complete BK for all the possible domains, therefore, reasoning in the continuum is always performed with respect to an incomplete BK. See [6] for the problems which can appear when a part of the knowledge is missing.

3. CONCEPT SEARCH

C-Search can be positioned anywhere in the semantic continuum with syntactic search being its base case. In fact, *CSearch* reuses retrieval models (*Model*) and data structures

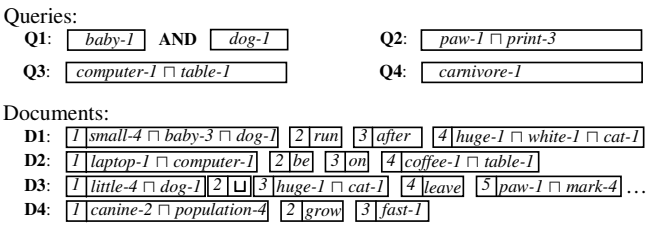

Figure 3: Document and Query Representations

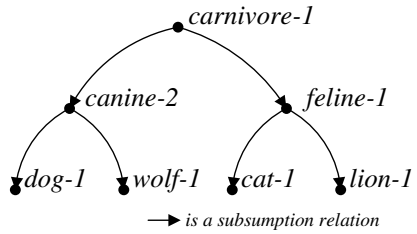

Figure 4: Terminological knowledge base $\mathcal{T}$

($Data\ Structure$) of syntactic search with the only difference in that now words (W) are substituted with complex 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:

$$Syntactic\ Search \xrightarrow{Term(W \to C), Match(WMatch \to SMatch)} CSearch$$

Below we briefly describe how the words in W are converted into the complex concepts in C and also how the semantic matching $SMatch$ is implemented in *CSearch*. We refer the interested reader to [3] for a complete account.

3.1 From Words To Complex Concepts

In *CSearch*, search is implemented by using complex concepts expressed in a propositional Description Logic (DL) language [4] (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 *BK* (e.g., WordNet). For instance, the word *dog* used in the sense of a domestic dog, which is the first sense in the *BK*, is converted into the atomic concept *dog*-1. The conversion of words into atomic concepts is performed as follows. First, we look up and enumerate all meanings of the word in the *BK*. Next, we perform word sense filtering, i.e., we discard word senses which are not relevant in the given context (see [3, 19] for more details).

Noun phrases are translated into the logical conjunction of atomic concepts corresponding to the words. For instance, the noun phrase *A little dog* is translated into the concept *little*-4 $\sqcap$ *dog*-1. Descriptive phrase, defined as a set of noun phrases connected by coordinating conjunction *OR*, 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). In general case, complex concepts (C) can be represented as disjunctions ($\sqcup$) of conjunctions ($\sqcap$) of atomic concepts (A) without negation:

$$C \equiv \sqcup \sqcap A^d \tag{2}$$

In *CSearch*, every document is represented as an enumerated sequence of conjunctive components $\sqcap A^d$ possibly connected by symbol "$\sqcup$". For example, in Figure 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 $\sqcup \sqcap A^d$. For example, the first three positions in the sequence for document $D3$ in Figure 3 represent the concept (*little*-4 $\sqcap$ *dog*-1) $\sqcup$ (*huge*-1 $\sqcap$ *cat*-1).

3.2 From Word to Concept Matching

In *CSearch*, we can search for documents describing complex concepts which are semantically related to complex concepts in the user query. We assume that, when a user is searching for a concept, she is also interested in more specific concepts (this assumption can be easily generalized to any "suitable" notion of semantic proximity). Formally a query answer $QA(C^q, \mathcal{T})$, in *CSearch*, is defined as follows:

$$QA(C^q, \mathcal{T}) = \{d \mid \exists C^d \in d, \text{ s.t. } \mathcal{T} \models C^d \sqsubseteq C^q\} \tag{3}$$

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 (i.e., the *BK*) which is used in order to check if C^d is more specific then C^q. A small fragment of $\mathcal{T}$ is represented in Figure 4. $\mathcal{T}$ 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.

Query answer $QA(C^q, \mathcal{T})$, defined in Equation 3, is computed by 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 *CSearch*, we adopt a positional inverted index to index conjunctive components $\sqcap A^d$ by all more general or equivalent atomic concepts from $\mathcal{T}$. For example, in Figure 5 we show a fragment of the positional inverted index created by using the document representations in Figure 3. The inverted index dictionary, in *CSearch*, consists of atomic concepts from $\mathcal{T}$ (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*) = $[\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle; \langle D4, 1, [1]\rangle]$, which means that at first position in documents $D1$,

Dictionary (t)	Posting lists (P(t))
$\sqcup$	$\langle D3, 1, [2]\rangle$
baby-3	$\langle D1, 1, [1]\rangle$
canine-2	$\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle; \langle D4, 1, [1]\rangle$
carnivore-1	$\langle D1, 2, [1, 4]\rangle; \langle D3, 2, [1, 3]\rangle; \langle D4, 1, [1]\rangle$
computer-1	$\langle D2, 1, [1]\rangle$
feline-1	$\langle D1, 1, [4]\rangle; \langle D3, 1, [3]\rangle$
leave	$\langle D3, 1, [4]\rangle$
little-4	$\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle$

Figure 5: Positional Inverted Index

$D3$, and $D4$ there are conjunctive components (i.e., *small*-4$\sqcap$ *baby*-3$\sqcap$*dog*-1, *little*-4$\sqcap$*dog*-1, and *canine*-2$\sqcap$*population*-4) which are more specific than *canine-2*. The posting list $P(\sqcup)$ stores the positions of the symbol "$\sqcup$".

Now the query answer $QA(C^q, \mathcal{T})$ can be computed just by merging posting lists (i.e., by computing intersections and unions of posting lists). For instance, positions of conjunctive components $\sqcap A^d$ which are more specific than complex query concept $\sqcap A^q$, i.e., $\sqcap A^d \sqsubseteq \sqcap A^q$, can be computed by intersecting the posting lists for all the atomic concepts A^q in $\sqcap A^q$. Let us consider, for example, the posting lists $P(A^q)$ for atomic concepts *little*-4 and *carnivore*-1.

$$P(\textit{little}\text{-}4) = [\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$$
$$P(\textit{carnivore}\text{-}1) = [\langle D1, 2, [1, 4]\rangle; \langle D3, 2, [1, 3]\rangle; \langle D4, 1, [1]\rangle]$$

In this case, the posting list computed for complex concept *little*-4 $\sqcap$ *carnivore*-1 is as follows:

$$P(\textit{little}\text{-}4 \sqcap \textit{carnivore}\text{-}1) = [\langle D1, 1, [1]\rangle; \langle D3, 1, [1]\rangle]$$

The complete algorithm for computing a query answer $QA(C^q, \mathcal{T})$ is described in [3].

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

$$\begin{aligned} QA(q_i \text{ AND } q_j, \mathcal{T}) &= QA(q_i, \mathcal{T}) \cap QA(q_j, \mathcal{T}) \\ QA(q_i \text{ NOT } q_j, \mathcal{T}) &= QA(q_i, \mathcal{T}) \ / \ QA(q_j, \mathcal{T}) \end{aligned} \quad (4)$$

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

The main limitation of *CSearch* is that it is a centralized system, i.e., the BK and the inverted index are stored in a single place. As any other centralized system, *CSearch* can not scale without the need for powerful servers.

4. SYNTACTIC SEARCH IN DHT

Distributed Hash Tables (DHTs) have been proposed as a way to enable an efficient discovery of objects in a very large P2P networks [12, 16, 14, 21]. In DHT, every object is associated with a key, which is transformed into a hash using some hash function. The range of the output values of the hash function forms an ID space. Every peer in the network is responsible for storing a certain range of keys. Values, e.g., objects or information about objects, are stored at the precisely specified locations defined by the keys.

The two main operations provided by DHT are:

- *put (key, value)* - stores the *value* on the peer responsible for the given *key*.
- *get (key)* $\rightarrow$ *value* - finds a peer responsible for the *key* and retrieve the *value* for the *key*.

A straightforward way to implement syntactic search is to use the DHT to distribute peers' inverted indices in the P2P network [13]. Peers locally compute posting lists $P(t)$ for every term t and store them in the network by using the DHT 'put' operation. The *key* in this case is a term t while the *value* is a posting list $P(t)$ associated with t. In DHT, each peer is responsible for a few terms and for every term t the peer merges all the posting lists $P(t)$ for t from all the peers in the network. In order to find a set of documents which contain a term t we just need to contact the peer responsible for t and retrieve the corresponding posting list. The DHT 'get' operation does exactly this. In order to search for more than one term, we, first, need to retrieve posting lists for every single term, and then to intersect all these posting lists.

The above approach has several problems (see e.g. [9, 17]). Let us consider some of these problems.

Storage. For a large document collection, the number and the size of posting lists can be also large. Therefore, the storage needed to store the posting lists can potentially be bigger than the storage peers can (or want) to allocate.

Traffic. Posting lists needs to be transferred when peers join or leave the network. Searching with multiple terms requires intersection of posting lists, which also need to be transferred. In the case of huge posting lists, a bandwidth consumption can exceed the maximum allowed. In [9], it is shown that the efficiency of DHT can be even worse than the efficiency of a simple flooding algorithm.

Load balancing. Popularity of terms, i.e., the number of term's occurrences, can vary enormously. It can result in an extremely imbalanced load e.g., some peers will store and transfer much more data than others.

Several approaches were proposed in order to address the described above problems and to improve performance of information retrieval in structured P2P networks. Some of optimization techniques (e.g., Bloom Filters), which can improve the performance of posting lists intersecting, are summarized in [9]. Caching of results for queries with multiple terms is discussed in [2, 15]. In [15], only those queries are cached which are frequent enough and simple flooding is used for rare queries. In [17], only important (or top) terms are used for indexing of each document. Moreover, the term lists are stored on peers responsible for these top terms. Notice that by using only the top terms we can decrease the quality of search results. Automatic query expansion is proposed as a way to address this problem [17]. Some techniques to balance the load across the peers are also presented in [17]. Normally users are interested only in a few (k) high quality answers. An example of the approach

for retrieving *top k* results, which does not require transmitting of entire posting lists, is discussed in [20]. In [10], indexing is performed by terms and term sets appearing in a limited number of documents. Different filtering techniques are used in [10] in order to make vocabulary to grow linearly with respect to the document collection size. In [1], it was proposed to index a peer containing a document and not the document itself. At search time, first, those peers are selected, which are indexed by all the terms in the query, then, the most promising peers are select, and finally, local search is performed on these peers.

Notice that the above approaches are implementing syntactic search. Therefore, the problems of syntactic search, i.e., problems of polysemy, synonymy, complex concepts, and related concepts, can also affect the quality of the results produced by these approaches.

5. P2P CONCEPT SEARCH

In order to provide semantic search in DHT networks, we propose to extend the centralized version of *CSearch* to *P2P CSearch*. First, we extend the reasoning with respect to a single background knowledge $\mathcal{T}$ to the reasoning with respect to the background knowledge $\mathcal{T}_{P2P}$ which is distributed among all the peers in the network. Second, we extend the centralized inverted index (II) to distributed inverted index build on top of DHT. The idea is schematically represented in the equation below.

$$\boxed{CSearch \xrightarrow{Knowledge(\mathcal{T} \rightarrow \mathcal{T}_{P2P}), Index(II \rightarrow DHT)} P2P\ CSearch}$$

In the following, we show how the distributed background knowledge $\mathcal{T}_{P2P}$ can be implemented on top of DHT and also we show how DHT can be used, in *P2P CSearch*, to provide an efficient distributed semantic indexing and retrieval.

5.1 Distributed Background Knowledge

To access the background knowledge $\mathcal{T}$, stored on a single peer, *CSearch* needs at least the following three methods:

getConcepts(W) returns a set of all the possible meanings (atomic concepts A) for word W. For example, *getConcepts(canine)* $\rightarrow$ {*canine-1 ('conical tooth'), canine-2 ('mammal with long muzzles')*}.

getChildren(A) returns a set of all the more specific atomic concepts which are directly connected to the given atomic concept A in $\mathcal{T}$. For example, with respect to $\mathcal{T}$ in Figure 4, *getChildren (carnivore-1)* $\rightarrow$ {*canine-2, feline-1*}.

getParents(A) returns a set of all the more general atomic concepts which are directly connected to the given atomic concept A in $\mathcal{T}$. For example, with respect to $\mathcal{T}$ in Figure 4, *getParents (dog-1)* $\rightarrow$ {*canine-2*}.

In order to provide access to background knowledge $\mathcal{T}_{P2P}$ distributed over all the peers in the P2P network, we create distributed background knowledge *DBK*. In *DBK*, each atomic concept A is identified by a unique concept ID (A_{ID}) which is composed from peer ID (P_{ID}), where peer is a creator of the atomic concept, and local concept ID in the Knowledge Base of the peer. Every atomic concept A is represented as a 3-tuple: $A = \langle A_{ID}, POS, GLOSS \rangle$, where A_{ID} is a concept ID; POS is a part of speech; and $GLOSS$ is a natural language description of A. In the rest of the paper, for the sake of presentation, instead of complete representation $\langle A_{ID}, POS, GLOSS \rangle$ we use just *lemma-sn*.

DBK is created on top of a DHT. Atomic concepts are indexed by words using the DHT 'put' operation, e.g., *put(canine,* {*canine-1, canine-2*}*)*. Moreover, every atomic concept is also indexed by related atomic concepts together with the corresponding relations. We use a modification of the DHT 'put' operation *put(A, B, Rel)*, which stores atomic concept B with relation *Rel* on the peer responsible for (a hash of) atomic concept A, e.g., *put(canine-2, dog-1, '$\sqsubseteq$')*, *put(canine-2, carnivore-1, '$\sqsupseteq$')*.

After *DBK* has been created, *getConcepts(W)* can be implemented by using the DHT 'get' operation, i.e., getConcepts(W) = get(W). Both methods *getChildren(A)* and *getParents(A)* are implemented by using a modified DHT 'get' operation *get(A, Rel)*, i.e., *getChildren(A) = get(W, '$\sqsubseteq$')* and *getParents(A) = get(W, '$\sqsupseteq$')*. The operation *get(A, Rel)* finds a peer responsible for atomic concept A and retrieve only those atomic concepts B which are in relation *Rel* with A.

Let us now see how *DBK* can be bootstrapped. At the beginning we have one single peer in the P2P network and *DBK* is equivalent to the background knowledge $\mathcal{T}$ of this peer. For example, $\mathcal{T}$ can be created from WordNet. A new peer joining the P2P network bootstrap its own background knowledge from *DBK* by doing the following three steps. First, the peer computes a set of words which are used in the local document collection. Second, the peer download from *DBK* a set of all the atomic concepts which are associated with these words by using 'getConcepts' method. Finally, the peer download all the more general atomic concepts by recursively calling 'getParents' method.

After bootstrapping, a user of each peer can extend *DBK* in the domain of her expertise according to her needs. The user can add a new atomic concept A to *DBK* by providing a set of words **W**, a part-of-speech POS, and a gloss $GLOSS$. By using the 'getConcepts' method, the peer retrieves from *DBK* all the atomic concepts A indexed by words in **W**. Then, glosses of retrieved concepts are compared with the $GLOSS$ provided by user. We use gloss-based matchers from [8, 7, 5]. If no similar concepts are found, then A is created in the peer's local background knowledge and indexed in *DBK*. The user can also add a new meaning (i.e., to assign an atomic concept A) to a word W. Similarly to how it was described above, before adding a new information, system checks if this information is not already in the system. Moreover, the user can define a new relation between atomic concepts in *DBK*. Before adding a new relation, system first checks if this relation does not introduce cycles in *DBK*. Cycles should be prevented because we need to have an acyclic *DBK*. Also system checks if the given relation can not be decomposed into a sequence of existing relations. This step is done in order to minimize the amount of stored information.

Notice, that by extending peer's background knowledge $\mathcal{T}$ to *DBK* which stores $\mathcal{T}_{P2P}$, we are likely to have a higher cov-

erage on words, atomic concepts, and relations. Therefore, we can enable semantics to a higher extend in the semantic continuum, e.g., when user types a word which is not present in her $\mathcal{T}$, she can use atomic concepts from background knowledge of other peers stored in *DBK*.

5.2 Indexing and Retrieval

The query answer defined in Equation 3, can be extended to the case of distributed search by taking into account that the document collection D_{P2P} is equivalent to the union of all the documents from all the peers in the network (where each document d is uniquely identified by a document ID) and also that background knowledge $\mathcal{T}_{P2P}$ is distributed among all the peers.

$$QA(C^q,\mathcal{T}_{P2P}) = \{d \in D_{P2P} \mid \exists C^d \in d, \text{s.t. } \mathcal{T}_{P2P} \models C^d \sqsubseteq C^q\} \quad (5)$$

Let us consider a subset $QA(C^q,\mathcal{T}_{P2P}, A)$ of the query answer $QA(C^q,\mathcal{T}_{P2P})$. $QA(C^q,\mathcal{T}_{P2P}, A)$ consists of documents d which contain at least one complex concept C^d which is more specific than the complex query concept C^q and contains atomic concept A.

$$QA(C^q,\mathcal{T}_{P2P},A) = \{d \in D_{P2P} \mid \exists C^d \in d, \text{s.t. } \mathcal{T}_{P2P} \models C^d \sqsubseteq C^q \text{ and } \exists A^d \in C^d, \text{s.t. } A^d = A\} \quad (6)$$

If by $C(A)$ we denote a set of all atomic concepts A^d, which are equivalent to or more specific than concept A, i.e.,

$$C(A) = \{A^d \mid \mathcal{T}_{P2P} \models A^d \sqsubseteq A\} \quad (7)$$

then, it can be shown that, given Equation 6, the query answer $QA(C^q,\mathcal{T}_{P2P})$ can be computed as follows

$$QA(C^q,\mathcal{T}_{P2P}) = \bigcup_{(\sqcap A^q) \in C^q} \bigcup_{A \in C(A^*)} QA(C^q,\mathcal{T}_{P2P},A) \quad (8)$$

where A^* is an arbitrarily chosen atomic concept A^q in conjunctive component $\sqcap A^q$.

Given Equation 8, the query answer can be computed by using a recursive algorithm described below. The algorithm takes as input complex query concept C^q and computes as output a query answer QA in five macro steps:

Step 1 Initialize a query answer: $QA = \emptyset$.

Step 2 Select one atomic concept A from every conjunctive component $\sqcap A^q$ in complex query concept C^q. For every selected A, repeat steps 3, 4, and 5.

Step 3 Compute $QA(C^q,\mathcal{T}_{P2P},A)$ and add the results to QA.

Step 4 Compute a set $\mathbf{C}_{ms}$ of all more specific atomic concepts B which are directly connected to the given atomic concept A in $\mathcal{T}_{P2P}$.

Step 5 If $\mathbf{C}_{ms} \neq \emptyset$, then for every atomic concept in $\mathbf{C}_{ms}$, repeat steps 3, 4, and 5.

Note, that on step 2, atomic concepts A can be selected arbitrarily. In order to minimize the number of iterations, we chose A with the smallest number of more specific atomic concepts. The smaller the number, the fewer times we need to compute $QA(C^q,\mathcal{T}_{P2P},A)$ on step 3.

Word senses	
canine	*canine-1*, *canine-2*
More specific concepts	
canine-2	*dog-1*, *wolf-1*
More general concepts	
canine-2	*carnivore-1*

CSearch index	
canine-2	$\langle D4, 1, [1]\rangle$
carnivore-1	$\langle D4, 1, [1]\rangle$
population-4	$\langle D4, 1, [1]\rangle$

Figure 6: Peer's information

In the following, we, first, show how documents are indexed in *P2P CSearch*, and then we show how the described above algorithm can be implemented.

In *P2P CSearch*, complex concepts are computed in the same way as in *CSearch* (see Section 3.1). The only difference is that now if an atomic concept is not found in the local background knowledge $\mathcal{T}$, then $\mathcal{T}_{P2P}$ is queried instead. *P2P CSearch* also uses the same document representation as *CSearch* (see Figure 3).

After document representations are computed, the indexing of documents is performed as follows. Every peer computes a set of atomic concepts A which appear in the representations of peer's documents. For every atomic concept A, the peer computes a set of documents d which contain A. For every pair $\langle A, d\rangle$, the peer computes a set $S(d, A)$ of all the complex document concepts C^d in d, which contain A.

$$S(d, A) = \{C^d \in d \mid A \in C^d\} \quad (9)$$

For example, if d is document $D1$ in Figure 3 and A is equivalent to *dog-1*, then $S(d, A) = \{$*small-4*$\sqcap$*baby-3*$\sqcap$*dog-1*$\}$. For every A, the peer sends document summaries corresponding to A, i.e., pairs $\langle d, S(d, A)\rangle$, to a peer p_A responsible for A in *DBK*. The peer p_A indexes these summaries using the local *CSearch*. In total, every peer in the network is responsible for some words and for some atomic concepts. Peers maintain the following information for their words and concepts:

1. For every word, the peer stores a set of atomic concepts (word senses) for this word.
2. For every atomic concept, the peers stores a set of direct more specific and more general atomic concepts.
3. Document summaries $\langle d, S(d, A)\rangle$ for all the atomic concepts A (for which the peer is responsible) are stored on the peer and indexed in the local *CSearch*, i.e., the summaries are indexed in the positional inverted index (like the one in Figure 5).

An example of the information, which can be stored on the peer responsible for a single word *canine* and for a single atomic concept *canine-2*, is shown in Figure 6.

Now, let us see how different steps of the algorithm for computing the query answer are implemented in *P2P CSearch*:

Step 1 A peer p_I initiates the query process for complex query concept C^q and initialize the query answer QA.

Step 2 For every $\sqcap A^q$ in C^q, p_I selects A in $\sqcap A^q$ with the smallest number of more specific atomic concepts. For every selected A, C^q is propagated to the peer p_A responsible for A.

Step 3 p_A receives the query concept C^q and locally (by using *CSearch*) computes the set $QA(C^q,\mathcal{T}_{P2P},A)$. The results are sent directly to p_I. On receiving new results $QA(C^q,\mathcal{T}_{P2P},A)$, p_I merges them with QA. An (intermediate) result is shown to the user.

Step 4 p_A computes the set $\mathbf{C}_{ms}$ by querying locally stored (direct) more specific concepts (e.g., see 'More specific concepts' in Figure 6).

Step 5 p_A propagates C^q to all the peers p_B responsible for atomic concepts B in $\mathbf{C}_{ms}$, i.e., Step 2 is repeated on every p_B.

Note, that, in order to optimize query propagation, peer p_A can pre-compute addresses of peers p_B which are responsible for more specific atomic concepts, and use DHT to locate such peers only when pre-computed information is outdated.

An example of how the query answer $QA(C^q,\mathcal{T}_{P2P},A)$ is computed is given in Figure 7. Peers, represented as small circles, are organized in a DHT overlay, represented as a ring. A query consisting of a single query concept C^q = *little*-4 $\sqcap$ *canine*-2 is submitted to peer P_I. Let us assume that atomic concept *canine*-2 has smaller number of more specific atomic concepts then concept *little*-4. In this case, C^q is propagated to a peer $P_{canine\text{-}2}$, i.e., the peer responsible for atomic concept *canine*-2. The query propagation is shown as a firm line in Figure 7. $P_{canine\text{-}2}$ searches in a local *CSearch* index with C^q. No results are found. $P_{canine\text{-}2}$ collects all the atomic concepts which are more specific then *canine*-2, i.e., atomic concepts *dog*-1 and $wolf$-1. Query concept C^q is propagated to peers $P_{dog\text{-}1}$ and $P_{wolf\text{-}1}$. $P_{dog\text{-}1}$ finds no results while $P_{dog\text{-}1}$ finds document D_1. D_1 is an answer because it contains concept *small*-4 $\sqcap$ *baby*-3 $\sqcap$ *dog*-1 which is more specific than *little*-4 $\sqcap$ *canine*-2. D_1 is sent to P_I, which presents it to the user. The results propagation is shown as a dash line in Figure 7. Both peers $P_{dog\text{-}1}$ and $P_{wolf\text{-}1}$ have no more specific concepts than *dog*-1 and $wolf$-1, therefore they do not propagate C^q to any other peers.

Note that the further we go in propagating query, the less precise is the answer. For instance, the user searching for *canine*-2 might be more interested in documents about concept *canine*-2 than in documents about concept *dog*-1, and she can be not interested at all in documents about very specific types of dogs (e.g., $affenpinscher$-1). In *P2P CSearch*, we allow user to specify the max allowed distance in numbers of links between atomic concepts in $\mathcal{T}_{P2P}$. Notice that this distance is similar to a standard time-to-live (TTL) [13].

In order to compute the query answer for a more complex query, e.g., query *baby*-1 AND *dog*-1 (in Figure 3), the intersection of posting lists needs to be computed (see Equation 4). Since our approach is not replacing syntactic search

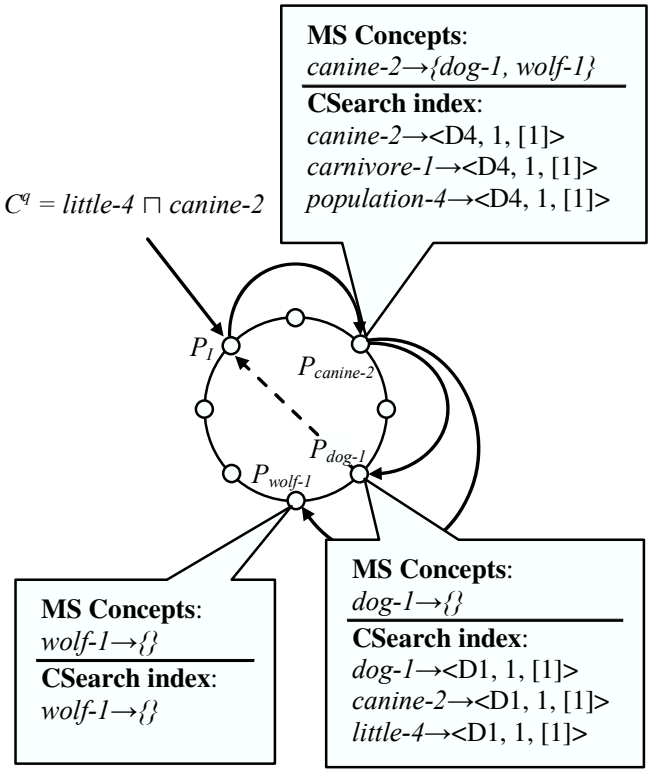

Figure 7: Query Answering

but extending it with semantics, for an efficient implementation of the intersection, we can reuse the optimization techniques developed in P2P syntactic search (see e.g. Section 4).

Other syntactic techniques, e.g., for ranking and merging of query results, can also be reused in *P2P CSearch*. For this, words, in these techniques, need to be replaced by concepts and syntactic matching needs to be replaced by semantic matching. Notice that in some P2P search approaches, instead of a single document, a group of documents, a peer, or a group of peers are indexed and searched. Our approach can be adopted to these problems: a group of documents, peers, or a group of peers should be annotated by complex concepts and then they can be indexed in the same way as a single document.

6. RELATED WORK

A number of P2P search approaches have been proposed in the literature (for an overview see [13]). Examples of how a full text retrieval can be efficiently implemented on top of structured P2P networks are described in [9, 2, 17, 20, 15, 1, 10]. All of these approaches are based on syntactic matching of words and, therefore, the quality of results produced by these approaches can be negatively affected by the problems related to the ambiguity of natural language. *P2P CSearch* is based on semantic matching of concepts which allows it to deal with ambiguity of natural language. Note that, since our approach extends syntactic search and does not replace it, the optimization techniques which are used in P2P syntactic search can be easily adapted to *P2P CSearch*.

Some P2P search approaches use matching techniques which are based on the knowledge about term relatedness (and not only syntactic similarity of terms). For instance, statistical knowledge about term co-occurrence is used in [18]. Knowledge about synonyms and related terms is used in [11]. Differently from these approaches, *P2P CSearch* is based on semantic matching of complex concepts and knowledge about concept relatedness, in *P2P CSearch*, is distributed among all the peers in the network.

7. CONCLUSIONS

In this paper, we have presented an approach, called *P2P CSearch*, which allows for a semantic search on top of distributed hash table (DHT). There are two main aspects in which *P2P CSearch* extends *CSearch*: (i) centralized document index is replaced by distributed index build on top of DHT; (ii) reasoning with respect to a single background knowledge is extended to the reasoning with respect to the background knowledge distributed among all the peers in the network. *P2P CSerarch* addresses the scalability problem of *CSerarch* and the ambiguity problem of natural language in P2P syntactic search. Future work includes: (i) the development of techniques which can control the quality of a user input and in general to control the quality of *DBK*; (ii) the development of document relevance metrics based on both syntactic and semantic similarity of query and document descriptions; (iii) evaluating the efficiency of the proposed solution.

8. REFERENCES

[1] Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. P2P content search: Give the web back to the people. In *5th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.

[2] Bobby Bhattacharjee, Sudarshan Chawathe, Vijay Gopalakrishnan, Pete Keleher, and Bujor Silaghi. Efficient peer-to-peer searches using result-caching. In *Proc. of the 2nd Int. Workshop on Peer-to-Peer Systems*, 2003.

[3] Fausto Giunchiglia, Uladzimir Kharkevich, and Ilya Zaihrayeu. Concept search. In *Proc. of ESWC'09*, Lecture Notes in Computer Science. Springer, 2009.

[4] Fausto Giunchiglia, Maurizio Marchese, and Ilya Zaihrayeu. Encoding classifications into lightweight ontologies. In *Journal on Data Semantics (JoDS) VIII*, Winter 2006.

[5] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Semantic schema matching. In *In Proceedings of CoopIS*, pages 347–365, 2005.

[6] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Discovering missing background knowledge in ontology matching. In *Proc. of ECAI*, 2006.

[7] Fausto Giunchiglia, Mikalai Yatskevich, and Enrico Giunchiglia. Efficient semantic matching. In *Proc. of ESWC*, Lecture Notes in Computer Science. Springer, 2005.

[8] Fausto Giunchiglia, Mikalai Yatskevich, and Pavel Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics (JoDS)*, 9:1–38, 2007.

[9] Jinyang Li, Boon Thau, Loo Joseph, M. Hellerstein, and M. Frans Kaashoek. On the feasibility of peer-to-peer web indexing and search. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, 2003.

[10] Toan Luu, Gleb Skobeltsyn, Fabius Klemm, Maroje Puh, Ivana Podnar Žarko, Martin Rajman, and Karl Aberer. AlvisP2P: scalable peer-to-peer text retrieval in a structured p2p network. In *Proc. VLDB Endow.*, 2008.

[11] Wenhui Ma, Wenbin Fang, Gang Wang, and Jing Liu. Concept index for document retrieval with peer-to-peer network. In *Proc. SNPD '07*, 2007.

[12] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001.

[13] John Risson and Tim Moors. Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks*, 50:3485–3521, 2006.

[14] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *IFIP/ACM Middleware*, 2001.

[15] Gleb Skobeltsyn and Karl Aberer. Distributed cache table: efficient query-driven processing of multi-term queries in p2p networks. In *P2PIR '06: Proceedings of the international workshop on Information retrieval in peer-to-peer networks*, 2006.

[16] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[17] Chunqiang Tang and Sandhya Dwarkadas. Hybrid global-local indexing for effcient peer-to-peer information retrieval. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, 2004.

[18] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. *SIGCOMM '03: Proceedings of the 2003 conference on Applications,technologies, architectures, and protocols for computer communications*, 2003.

[19] 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.

[20] Jiangong Zhang and Torsten Suel. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, 2005.

[21] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, January 2001.