Query Evaluation in P2P Systems of Taxonomy-based Sources: Algorithms, Complexity, and Optimizations

Carlo Meghini
Consiglio Nazionale delle Ricerche
Istituto della Scienza e delle Tecnologie della Informazione
Pisa, Italy
carlo.meghini@isti.cnr.it

Yannis Tzitzikas
Department of Computer Science
University of Crete, Heraklion, Greece
Institute of Computer Science
Foundation for Research and Technology – Hellas (FORTH-ICS)
tzitzik@ics.forth.gr

Anastasia Analyti
Institute of Computer Science
Foundation for Research and Technology – Hellas (FORTH-ICS)
Crete, Greece
analyti@ics.forth.gr

Abstract

In this study we address the problem of answering queries over a peer-to-peer system of taxonomy-based sources. A taxonomy states subsumption relationships between negation-free DNF formulas on terms and negation-free conjunctions of terms. To the end of laying the foundations of our study, we first consider the centralized case, deriving the complexity of the decision problem and of query evaluation. We conclude by presenting an algorithm that is efficient in data complexity and is based on hypergraphs. More expressive forms of taxonomies are also investigated which however lead to intractability. We then move to the distributed case, and introduce a logical model of a network of taxonomy-based sources. On such network, a distributed version of the centralized algorithm is then presented, based on a message passing paradigm, and its correctness is proved. We finally discuss optimization issues, and relate our work to the literature.

1 Introduction

Consider a tetrad $(T, \preceq, Obj, I)$ where T is a set of terms, $\preceq$ is a subsumption relation over concepts expressed using T (e.g. $(\texttt{Animal} \wedge \texttt{FlyingObject}) \vee \texttt{Penguin} \preceq \texttt{Bird}$), Obj is a set of objects and I is a function from T to $\mathcal{P}(Obj)$, assigning a description (*i.e.*, a set of terms) to each object. Now assume that all these are not stored at a single place but they are distributed over a set $\mathcal{N} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ of independent peers. Moreover assume that each peer S_i can have zero, one or more $\preceq$-relationships between its terms (i.e. T_i) and some concepts over the terminologies of other peers (e.g. $\texttt{Parrot}_\texttt{j} \preceq \texttt{Birds}_\texttt{i}$ and $\texttt{Animal}_\texttt{k} \wedge \texttt{Flying}_\texttt{k} \preceq \texttt{Birds}_\texttt{i}$). In this paper we address the problem of answering Boolean queries over this kind of systems.

Some parts of the work reported in this paper have been already published. Namely, [40] presents a first model of a network of articulated sources, while [39] studies query evaluation on taxonomies including only term-to-term subsumption relationships. Finally, [30] presents a procedure for evaluating queries over centralized sources supporting term-to-query subsumption relationships, as well as hardness results for extensions. In this paper,

- we consider from the start the most complex type of subsumption for which we can propose an efficient query evaluation procedure, allowing subsumption relationships between negation-free DNF combinations of terms and negation-free conjunctions of terms. We then place the hardness results presented in [30] in context, thus showing that any Boolean extension of the expressive power of subsumption leads to intractability of the query answering problem;
- we ground the centralized query evaluation procedure for this kind of sources, presented in [30], on solid theoretical basis, proving its correctness, and linking it to the existing algorithmic and complexity literature;
- we present a distributed query evaluation procedure, based on a functional model of a peer; correctness and complexity of this procedure are given;
- we describe optimization techniques that can be used for improving the efficiency of query evaluation;
- we relate our work to the existing literature on peer-to-peer systems.

The paper is structured as follows: Section 2 gives the background on peer-to-peer systems, while Section 3 introduces sources, presenting the centralized query evaluation procedure. Networks of sources are considered in Section 4, where our algorithm for query evaluation on networks is presented, and Section 5 discusses optimization issues. Section 6 compares our work with related work and Section 7 concludes the paper.

2 Background

A peer-to-peer (P2P) system is a distributed system in which participants (the peers) rely on one another for service, rather than solely relying on dedicated and often centralized servers. The most popular P2P systems have focused on specific application domains like music file sharing [3, 1, 2]) or on providing file-system-like capabilities [8]. In most of the cases, these systems do not provide semantic-based retrieval services as the name of an object (e.g. the title of a music file) is the only means for describing the contents of an object.

Semantic-based retrieval in P2P systems is a great challenge that raises questions about data models, conceptual modeling, query languages, algorithms and data structures for query evaluation, and techniques for dynamic schema mapping. Roughly, the language that can be used for indexing the objects of the domain and for formulating semantic-based queries, can be *free* (e.g natural language) or *controlled*, i.e. object descriptions and queries may have to conform to a specific vocabulary and syntax. The former case, resembles distributed Information Retrieval (IR) systems and this approach is applicable in the case where the objects of the domain have a textual content (e.g. [29, 27, 14, 37]). In the latter case, the objects of a peer are indexed according to a specific conceptual model represented in a particular data model (e.g. relational, object-oriented, logic-based, etc), and content searches are formulated using a specific query language. Of course, a P2P system might impose a single conceptual model on all participants to enforce uniform, global access, but this will be too restrictive. Alternatively, a limited number of conceptual models may be allowed, so that traditional information mediation and integration techniques will likely apply (with the restriction that there is no central authority), e.g. see [32, 31].

The case of fully heterogeneous conceptual models makes uniform global access extremely challenging and this is the focus of this paper. From a data modeling point of view several approaches for P2P systems have been proposed recently, including relational-based approaches [7], XML-based approaches [24] and RDF-based [31].

In this paper we consider the fully heterogeneous conceptual model approach (where each peer can have its own schema), with the only restriction that each conceptual model is represented as a taxonomy. A taxonomy can range from a simple tree-structured hierarchy of terms, to the concept lattice derived by Formal Concept Analysis [22], or to the concept lattice of a Description Logics theory. Specifically, according to our model, each peer consists of a *taxonomy*, an *object base*, i.e. a database that contains descriptions of the objects according to the taxonomy, and a number of (one-way) *articulations* to some of the other peers of the network, where an articulation is actually a mapping between terms of the peer and terms (or queries) of other peers. Articulations aim at bridging the inevitable naming, granularity and contextual heterogeneities that may exist between the taxonomies of the peers (for some examples see [40]). For example, the taxonomy of a peer $\mathcal{S}_1$ could be the following: $\{$ $\texttt{Penguin} \preceq \texttt{Animal}$, $\texttt{Pelican} \preceq \texttt{Animal}$, $\texttt{Ostrich} \preceq \texttt{Animal}$, $(\texttt{Animal} \wedge \texttt{FlyingObject}) \vee \texttt{Penguin} \vee \texttt{Ostrich} \preceq \texttt{Bird}$ $\}$. The object base of $\mathcal{S}_1$ could be the following: $\{$ $\texttt{Ostrich}(1)$, $\texttt{Bird}(2)$, $\texttt{Animal}(3)$, $\texttt{FlyingObject}(3)$ $\}$. $\mathcal{S}_1$ could have an articulation to a peer $\mathcal{S}_2$ like $\{$ $\Pi\iota\nu\gamma\kappa o\upsilon\acute{\iota}\nu o\varsigma_2 \preceq \texttt{Penguin}$, $\Pi\epsilon\lambda\epsilon\kappa\acute{\alpha}\nu o\varsigma_2 \preceq \texttt{Pelican}$ $\}$, an articulation to a peer $\mathcal{S}_3$ like $\{$ $\texttt{Animale}_3 \wedge \texttt{Alato}_3 \preceq \texttt{Birds}$ $\}$, and an articulation to two peers $\mathcal{S}_4, \mathcal{S}_5$ of the form: $\{$ $(\texttt{Fliegentier}_4) \vee (\texttt{Animal}_5 \wedge \texttt{Volant}_5) \preceq (\texttt{Animal} \wedge \texttt{FlyingObject})$ $\}$.

The articulations can be exploited for finding objects in the network using content-based queries, for publishing objects and their descriptions to the network, and for obtaining more rich descriptions of the objects (by aggregating their descriptions according to different conceptual models)[1]. Apart from determining query propagation, these mappings are actually used for translating the query into a vocabulary that the recipient can understand (and thus answer). In certain cases, these inter-taxonomy mappings could be constructed automatically (e.g. using the data-driven method proposed in [38]).

The placement of our work with respect to other logic-founded approaches for query evaluation over P2P systems is given in Section 6.

3 Information sources

This Section defines information sources and derives algorithms and complexity results for querying them. These results will be applied later, upon studying networks of sources. The model is first introduced; the computational and algorithmic foundations of the query evaluation problem are then given; Section 3.3 presents an efficient query evaluation method. Finally, three extensions of information sources are discussed: those having negation in the taxonomy, those having negation only in the query language, and those having disjunction in the taxonomy. For all these, the query evaluation problem is studied, deriving complexity results or correct and efficient algorithms, if any.

3.1 The model

The basic notion of the model is that of *terminology:* a terminology T is a non-empty set of terms. A terminology comes with an associated language for constructing more complex terms, called *queries,* from the given ones.

Definition 1 (Query) The *query language* associated to a terminology T, $\mathcal{L}_T$, is the language defined by the following grammar, where t is a term of T :

$$q \ ::= d \mid q \vee d$$
$$d \ ::= t \mid t \wedge d.$$

An instance of q is called a *query,* while an instance of d is called a *conjunctive query.* Each d component of a query q is called *disjunct* of q. □

Terms and queries can be used for defining taxonomies.

Definition 2 (Taxonomy) A *taxonomy* is a pair $(T, \preceq)$ where T is a terminology and $\preceq$ is a binary relation between queries, $\preceq \subseteq (\mathcal{L}_T \times \mathcal{L}_T)$, which is reflexive and transitive, such that $q \preceq q'$ and $q \neq q'$ imply that q' is a conjunctive query. □

[1] The latter is possible only if the objects have a unique global identity in the entire network (like URI for example).

If $(q, q') \in \preceq$, we say that q is subsumed by q' and we write $q \preceq q'$. The reason for having only conjunctive queries as right-hand sides of non-trivial subsumption relationships is computational, and will be discussed later.

Definition 3 (Interpretation) An *interpretation* for a terminology T is a pair (Obj, I), where Obj is a finite set of objects and I is a total function $I : T \to \mathcal{P}(Obj)$. $\square$

Interpretations can be extended to queries in an intuitive way, thus defining the semantics of the query language:

Definition 4 (Query extension) Given an interpretation I of a terminology T and a query $q \in \mathcal{L}_T$, the *extension of q in I, q^I*, is defined as follows:

1. $(q \vee d)^I = q^I \cup d^I$
2. $(d \wedge t)^I = d^I \cap t^I$
3. $t^I = I(t)$. $\square$

Since the function $\cdot^I$ is an extension of the interpretation function I, we will simplify notation and will write $I(q)$ in place of the formally correct q^I. We can now define a taxonomy-based source, called *information source* or simply *source.*

Definition 5 (Information source) An *information source* S is a 4-tuple $S = (T_S, \preceq_S, Obj_S, I_S)$, where $(T_S, \preceq_S)$ is a taxonomy and (Obj_S, I_S) is an interpretation for T_S. $\square$

When no ambiguity will arise, we will simplify notation by omitting the subscript in the components of sources. In addition, an interpretation will be equated with its interpretation function I. Given a source $S = (T, \preceq, Obj, I)$ and an object $o \in Obj$, the *index of o in S, $ind_S(o)$*, is given by the terms in whose interpretation o belongs, *i.e.*:

$$ind_S(o) = \{t \in T \mid o \in I(t)\}.$$

Some interpretations better reflect the semantics of subsumption.

Definition 6 (Models of a source) Given two interpretations I, I' of the same terminology T,

- I is a *model* of the taxonomy $(T, \preceq)$ if $q \preceq q'$ implies $I(q) \subseteq I(q')$;
- I is smaller than I', $I \leq I'$, if $I(t) \subseteq I'(t)$ for each term $t \in T$;
- I is a *model* of a source $S = (T, \preceq, Obj, I')$ if it is a model of $(T, \preceq)$ and $I' \leq I$. $\square$

The notion of model of a source can be used to obtain a simpler, but equivalent, notion of source, in which (non-trivial) subsumption relationships relate conjunctive queries to terms. The equivalence is based on the observation that the propositional formula:

$$(C_1 \vee \ldots \vee C_n) \to (t_1 \wedge \ldots \wedge t_m)$$

where each C_i in the left hand-side is any propositional formula, is logically equivalent to the formula:

$$(C_1 \to t_1) \wedge (C_1 \to t_2) \wedge \ldots \wedge (C_1 \to t_m) \wedge \ldots \wedge (C_n \to t_1) \wedge (C_n \to t_2) \wedge \ldots \wedge (C_n \to t_m),$$

that is, the two formulae have the same models. Formally, the *simplification* of a taxonomy $(T, \preceq)$, is the taxonomy $(T, \sigma(\preceq))$, where $\sigma(\preceq)$ is the reflexive and transitive closure of the following relation[2]:

$$\{(C, t) \mid (C_1 \vee \ldots \vee C_n, t_1 \wedge \ldots \wedge t_m) \in \preceq^r,\ C \in \{C_1, \ldots, C_n\},\ t \in \{t_1, \ldots, t_m\}\}.$$

[2]The transitive reduction of a binary relation R on a set X, is defined as [17] $R^r = R_1 \setminus R_1^2$, where $R_1 = R \setminus \{(a, a) \mid a \in X\}$ and $R_1^2 = R_1 \circ R_1$. In practice, R^r is R without reflexive and transitive relationships, and its graphical rendering is generally known as the *Hasse diagram* of R.

Correspondingly, the simplification of a source $S = (T, \preceq, Obj, I)$ is the source $\sigma(S) = (T, \sigma(\preceq), Obj, I)$. It is not difficult to see that:

Proposition 1 J is a model of a source S if and only if it is a model of $\sigma(S)$. $\square$

Based on the last Proposition, from now on we will use the terms "taxonomy" and "source" as synonyms of "simplified taxonomy" and "simplified source", respectively. Formally, $(T, \preceq)$ and S will stand for $(T, \sigma(\preceq))$ and $\sigma(S)$, respectively.

A second usage of the notion of model is to define the query-answering function *ans* on sources.

Definition 7 (Answer) Given a source $S = (T, \preceq, Obj, I)$ and a query $q \in \mathcal{L}_T$, the *answer of q in S*, $ans(q, S)$, is given by $ans(q, S) = \{o \in Obj \mid o \in J(q)$ for all models J of $S\}$. $\square$

Indeed, we only need to consider term queries, because non-term queries can be embedded in the taxonomy. Specifically:

Proposition 2 For all sources $S = (T, \preceq, Obj, I)$ and non-term queries $q \in \mathcal{L}_T$, let $t_q \notin T$ and

$$\begin{array}{rcl} T^q & = & T \cup \{t_q\} \\ (\preceq^q)^r & = & \preceq^r \cup \{(t_1 \wedge \ldots \wedge t_m, t_q) \mid t_1 \wedge \ldots \wedge t_m \text{ is a disjunct of } q\} \\ I^q & = & I \cup \{(t_q, \emptyset)\}. \end{array}$$

Then, $ans(q, S) = ans(t_q, S^q)$ where $S^q = (T^q, \preceq^q, Obj, I^q)$. $\square$

In practice, the terminology T^q includes one additional term t_q, which has an empty interpretation and subsumes each query disjunct $t_1 \wedge \ldots \wedge t_m$ of q. The size of S^q is clearly polynomial in the size of S and q.

In light of the last Proposition, the problem of query evaluation amounts to determine $ans(t, S)$ for given term t and source S, while the corresponding decision problem consists in checking whether $o \in ans(t, S)$, for a given object o.

Query evaluation is strictly related to the unique minimal model of a source.

Proposition 3 For all sources $S = (T, \preceq, Obj, I)$ and terms $t \in T$, the unique minimal model of S, $\bar{I}$, is given by

$$\bar{I}(t) = \bigcup \{I(u) \mid u \in T,\ u \preceq t\} \ \cup \ \bigcup \{\bar{I}(q) \mid q \preceq t,\ q = t_1 \wedge \ldots \wedge t_m,\ m > 1\}.$$

Moreover, $ans(t, S) = \bar{I}(t)$. $\square$

3.2 Foundations

In this Section, we consider the computational foundations of query evaluation, starting from those of the more fundamental decision problem.

3.2.1 The decision problem

Given a source $S = (T, \preceq, Obj, I)$, $o \in Obj$, and $t \in T$, the decision problem $o \in ans(t, S)$ is P-complete in the size of the taxonomy. The hardness part of the proof is based on the following polynomial time reduction from the decision problem $P \models A$ in propositional datalog, known to be P-complete [15]:

- the terminology T is given by the letters occurring in P;

– $\preceq$ is the reflexive and transitive closure of the binary relation, defined as follows:

$$\{(t_1 \wedge \ldots \wedge t_m, t_0) \mid t_0 \leftarrow t_1, \ldots, t_m \in P\}.$$

– $Obj = \{1\}$;

– the interpretation function I is defined as follows: for each term $t_0 \in T$,

$$I(t_0) = \begin{cases} \{1\} & \text{if } t_0 \leftarrow \quad \in P \\ \emptyset & \text{otherwise} \end{cases}$$

It is easy to see that $P \models A$ if and only if $1 \in ans(A, (T, \preceq, Obj, I))$, thus obtaining hardness in the size of the program P.

For the membership part of the proof, we rely on an opposite reduction, which will also be used later. Let $S = (T, \preceq, Obj, I)$ be a source, $o \in Obj$ and $t \in T$. Define P_S to be the following propositional datalog program:

$$P_S = C_S \cup I_S \cup Q_S$$

where

$$\begin{aligned} C_S &= \{t_0 \leftarrow t_1, \ldots, t_m \mid (t_1 \wedge \ldots \wedge t_m, t_0) \in \preceq^r\} \\ I_S &= \{u \leftarrow \quad \mid u \in ind_S(o)\} \\ Q_S &= \{\leftarrow t\} \end{aligned}$$

The size of P_S is polynomial in the size of the taxonomy. It is easy to see that:

Lemma 1 For all sources $S = (T, \preceq, Obj, I)$, $o \in Obj$ and $t \in T$, $o \in ans(t, S)$ iff P_S is unsatisfiable. $\square$

This proves the membership of the decision problem in P, hence its P-completeness. From the P-completeness in the size of the taxonomy of the decision problem, the P-completeness in the size of the information source[3] of the query evaluation problem follows.

From an algorithmic point of view, the decision problem relies on directed B-hypergraphs, which are introduced next. We will mainly use definitions and results from [21].

A *directed hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $\mathcal{E} = \{E_1, E_2, \ldots, E_m\}$ is the set of directed hyperedges, where $E_i = (\tau(E_i), \chi(E_i))$ with $\tau(E_i), \chi(E_i) \subseteq \mathcal{V}$ for $1 \leq i \leq m$. $\tau(E_i)$ is said to be the *tail* of E_i, while $\chi(E_i)$ is said to be the *head* of E_i. A *directed B-hypergraph* (or simply *B-graph*) is a directed hypergraph, where the head of each hyperedge E_i, denoted as $h(E_i)$, is a single vertex.

A taxonomy can naturally be represented as a B-graph whose hyperedges represent one-to-one the subsumption relationships of the transitive reduction of the taxonomy. In particular, the *taxonomy B-graph* of a taxonomy $(T, \preceq)$ is the B-graph $\mathcal{H} = (T, \mathcal{E}_{\preceq})$ where

$$\mathcal{E}_{\preceq} = \{(\{t_1, \ldots, t_m\}, u) \mid (t_1 \wedge \ldots \wedge t_m, u) \in \preceq^r\}$$

Figure 1 left presents a taxonomy, whose B-graph is shown in the same Figure right.

A *path* P_{st} of length q in a B-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a sequence of nodes and hyperedges

$$P_{st} = (s = v_1, E_{i_1}, v_2, E_{i_2}, \ldots, E_{i_q}, v_{q+1} = t)$$

where: $s \in \tau(E_{i_1})$, $h(E_{i_q}) = t$ and $h(E_{i_{j-1}}) = v_j \in \tau(E_{i_j})$ for $2 \leq j \leq q$. If P_{st} exists, t is said to be *connected* to s. If $t \in \tau(E_{i_1})$, P_{st} is said to be a *cycle*; if all hyperedges in P_{st} are distinct, P_{st} is said to be *simple*. A simple path is *elementary* if all its vertices are distinct.

A *B-path* π_{st} in a B-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a minimal (with respect to deletion of vertices and hyperedges) hypergraph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$, such that:

[3] The *size* of an information source comprises the size of its taxonomy and the size of its interpretation, i.e., what is called combined complexity, in the database literature.

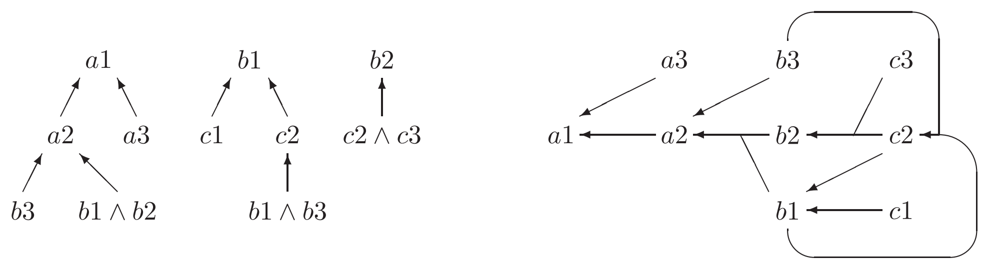

Figure 1: A taxonomy and its B-graph

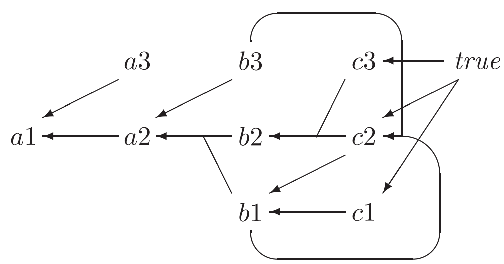

Figure 2: An object graph

1. $\mathcal{E}_\pi \subseteq \mathcal{E}$
2. $\{s, t\} \subseteq \mathcal{V}_\pi$
3. $x \in \mathcal{V}_\pi$ and $x \neq s$ imply that x is connected to s in $\mathcal{H}_\pi$ by means of a cycle-free simple path.

Vertex y is said to be *B-connected* to vertex x if a B-path π_{xy} exists in $\mathcal{H}$.

B-graphs and satisfiability of propositional Horn clauses are strictly related. The B-graph *associated to* a set of Horn clauses has 3 types of directed hyperedges to represent each clause:

– the clause $p \leftarrow q_1 \wedge q_2 \wedge \ldots \wedge q_s$ is represented by the hyperedge $(\{q_1, q_2, \ldots, q_s\}, p)$;

– the clause $\leftarrow q_1 \wedge q_2 \wedge \ldots \wedge q_s$ is represented by the hyperedge $(\{q_1, q_2, \ldots, q_s\}, \mathit{false})$;

– the clause $p \leftarrow$ is represented by the hyperedge $(\{\mathit{true}\}, p)$.

The following result is well-known:

Proposition 4 ([21]) A set of propositional Horn clauses is satisfiable if and only if in the associated B-graph, *false* is not B-connected to *true*. □

We now proceed to show the role played by B-connection in query evaluation. For a source $S = (T, \preceq, \mathit{Obj}, I)$ and an object $o \in \mathit{Obj}$, the *object decision graph* (simply the *object graph*) is the B-graph $\mathcal{H}_o = (T, \mathcal{E}_o)$, where

$$\mathcal{E}_o = \mathcal{E}_\preceq \cup \bigcup \{(\{\mathit{true}\}, u) \mid u \in \mathit{ind}_S(o)\}.$$

Figure 2 presents the object graph for the taxonomy shown in Figure 1 and an object o such that $\mathit{ind}_S(o) = \{c1, c2, c3\}$.

We can now prove:

Proposition 5 For all sources $S = (T, \preceq, \mathit{Obj}, I)$, terms $t \in T$, and objects $o \in \mathit{Obj}$, $o \in \mathit{ans}(t, S)$ iff t is B-connected to *true* in the object graph $\mathcal{H}_o$.

Proof: From Lemma 1, $o \in ans(t, S)$ iff P_S is unsatisfiable iff (by Proposition 4) false *is B-connected to* true *in the associated B-graph. By construction, $\mathcal{H}_o$ is the B-graph associated to P_S, where* t *plays the role of* false. □

3.2.2 Foundation of query evaluation

The basic reason why the decision problem can be efficiently solved, is that it requires traversing any hyperedge of the taxonomy B-graph at most once. In other words, when deciding membership of an object to a query answer, any (non-trivial) subsumption relationship needs to be used no more than once. However, this is not the case for query evaluation, for in this case all objects must be considered at once as potential candidates for the answer, and therefore a hyperedge can be traversed more than once, in different ways. From a more technical point of view, in deciding whether $o \in ans(t, S)$, we consider the cycle-free simple paths from any term in $ind_S(o)$ to t. These paths make up the B-path $\mathcal{H}_o$. Instead, in computing $ans(t, S)$, we need to consider a much larger hypergraph, call it $\mathcal{H}_{Obj}$, in which *true* is connected to *all* terms in T that belong to at least one object index. $\mathcal{H}_{Obj}$ is made up of all cycle-free simple paths from *any* term to t. Now, it is not difficult to see that these paths may be exponentially many in the size of the taxonomy. As an illustration, let us consider the taxonomy whose B-graph contains the following hyperedges:

$$\begin{array}{lllll} h_1 : (\{u_1, v_1\}, u_2) & h_2 : (\{u_2, v_2\}, u_3) & h_3 : (\{u_3, v_3\}, u_4) & h_4 : (\{u_4, v_4\}, u_5) & h_5 : (\{u_5, v_5\}, t) \\ g_1 : (\{u_1, v_1\}, v_2) & g_2 : (\{u_2, v_2\}, v_3) & g_3 : (\{u_3, v_3\}, v_4) & g_4 : (\{u_4, v_4\}, v_5) & \end{array}$$

Let us assume t is the query term. It is easy to verify that there are 2^4 cycle-free simple paths connecting u_1 to t, one for each sequence of the form

$$(u_1\ f_1\ x_2\ f_2\ x_3\ f_3\ x_4\ f_4\ x_5\ h_5\ t)$$

where f_i can be either h_i (in which case x_{i+1} is u_{i+1}) or g_i (in which case x_{i+1} is v_{i+1}) for $1 \leq i \leq 4$. In fact, any object o whose index $ind_S(o)$ contains either both u_j and v_j (for some $1 \leq j \leq 5$) or t, is in the answer of the query, and so there is an exponential number of indices which qualify for the query. In order to avoid examining all these indices, a smart query evaluation algorithm could try to generate only the minimal ones, which in our case are just 6. However, finding all minimal qualifying indices is an NP hard problem.

In proof, let us define an *answer set* A for a term query t to a source $S = (T, \preceq, Obj, I)$, to be a set of terms $A \subseteq T$, such that if the index of an object o, $ind_S(o)$, has all the terms in A, then o is an answer for t in S; formally, $A \subseteq ind_S(o)$ implies $o \in ans(t, S)$. We now present a polynomial time reduction from MINIMAL HITTING SET, a problem known to be NP-complete, to the problem of finding a minimal answer set for t in S. We recall the notion of hitting set: Given a collection $\mathcal{C}$ of subsets of a set C, a *hitting set* for $\mathcal{C}$ is a set $C' \subseteq C$ such that C' contains at least one element from each subset in $\mathcal{C}$. The basic working of the reduction is exemplified in Figure 3, the left part of which shows the collection $\mathcal{C}$, while the right part shows the corresponding taxonomy. The query is t. In general, letting $\mathcal{C} = \{C_1, \ldots, C_k\}$ be a collection of subsets of a set C, the corresponding source $S_\mathcal{C} = (T_\mathcal{C}, \preceq_\mathcal{C}, \emptyset, \emptyset)$ and term query $t_\mathcal{C}$ are defined as follows:

- $T_\mathcal{C} = C \cup \{t, u_1, \ldots, u_k\}$ where $t \notin C$ and $u_i \notin C$ for all $1 \leq i \leq k$.
- $\preceq^r_\mathcal{C} = \bigcup_{1 \leq j \leq k} \{(x, u_j) \mid x \in C_j\} \cup \{(u_1 \wedge \ldots \wedge u_k, t)\}$
- $t_\mathcal{C} = t$.

It can be easily proved that this is a polynomial time reduction and, of course, it holds that $\preceq_\mathcal{C}$ is reflexive and transitive. Moreover, the terms from which each term u_i can be reached in the taxonomy B-graph are those of the i-th collection in $\mathcal{C}$, plus the element u_i. Consequently, each hitting set for $\mathcal{C}$ contains a sub-term of each u_i, therefore it is an answer set for $t_\mathcal{C}$ and $S_\mathcal{C}$; in addition, the minimality of the former implies that of the latter. The converse is not true, because a minimal answer set X for $t_\mathcal{C}$ and $S_\mathcal{C}$ may contain a "foreign" term u_i. However, this is harmless, for u_i can be replaced in X by any of its sub-terms and the result is still a minimal hitting set for $\mathcal{C}$. This proves the NP-hardness.

$$\mathcal{C} = \{\{a, b\},\ \{b, c, d\},\ \{b, c, e, f\}\}$$

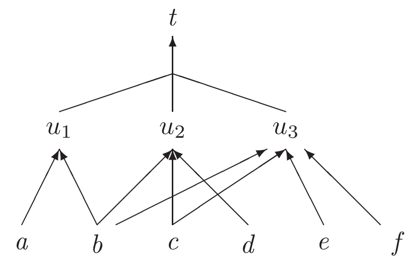

Figure 3: A collection of sets $\mathcal{C}$ and the B-graph of the corresponding taxonomy $(T_{\mathcal{C}}, \preceq_{\mathcal{C}})$

Notice that the reduction uses a much simpler type of information source than the one we consider in the present study, namely one whose taxonomy has only one hyperedge. Also, we have left the domain and the interpretation of $S_{\mathcal{C}}$ empty in order to stress that they play no role in the reduction.

It is not difficult to prove membership of the problem in NP, from which the NP-completeness in the size of the taxonomy of finding one minimal answer set follows. However, query evaluation requires finding *all* minimal answer sets, thus the complexity of this latter problem is much worse, in fact we believe that it is PSPACE-complete.

We now turn to the derivation of an algorithm for query evaluation, whose complexity is polynomial in the size of the information source (which may be exponentially higher than that of the taxonomy, of course).

3.3 Query evaluation

Proposition 3 does not directly lead to a simple method for query evaluation, as it may yield a recursive set of equations. As an illustration, let us consider the query $b1$ in our example source. We have:

$$\begin{aligned} \bar{I}(b1) &= I(b1) \cup I(c1) \cup I(c2) \cup \bar{I}(b1 \wedge b3) \\ \bar{I}(b1 \wedge b3) &= \bar{I}(b1) \cap \bar{I}(b2). \end{aligned}$$

The standard datalog approach to solve this problem is to map the program into a system of equations on relations, which is then solved by applying an iterative method (see Chapter 13 of [5]). Given the simplified form of datalog programs that we are dealing with, we propose a simpler method to perform query evaluation, based on B-graphs. Our method relies on the following result, which is just a re-phrasing of Proposition 5:

Corollary 1 For all sources $S = (T, \preceq, Obj, I)$, $o \in Obj$ and term queries $t \in T$, $o \in ans(t, S)$ if and only if either $o \in I(t)$ or there exists a hyperedge $(\{u_1, \ldots, u_r\}, t) \in \mathcal{E}_{\preceq}$ such that $o \in \bigcap\{ans(u_i, S) \mid 1 \leq i \leq r\}$. $\square$

This corollary simply "breaks down" Proposition 5 based on the distance between t and *true* in the object graph $\mathcal{H}_o$. If $o \in I(t)$, then $t \in ind_S(o)$, hence there is a hyperedge (in fact, a simple arc) from *true* to t in $\mathcal{H}_o$, which are 1 hyperedge distant from each other. If $o \notin I(t)$, then there are at least two hyperedges in between *true* and t. Let us assume that h is the one whose head is t. Since t is B-connected to *true,* each term u_i in the tail of h is B-connected to *true.* But this simply means, again by Proposition 5, that $o \in ans(u_i, S)$ for all the terms u_i, and so we have the Corollary. Notice that, by point 3 in the definition of B-path, t is connected to each u_i by a cycle-free simple path; this fact is used by the procedure QE in order to correctly terminate in presence of loops in the taxonomy B-graph $\mathcal{H}$.

The procedure QE, presented in Figure 4, computes $ans(t, S)$ for a given term t (and an implicitly given source S) by applying in a straightforward way Corollary 1. To this end, QE must be invoked as QE$(t, \{t\})$. The second input parameter of QE is the set of terms on the *path* from t to the currently considered term x. This set is used to guarantee that t is connected to all terms considered in the recursion by a cycle-free simple path. QE accumulates in R the result. The correctness of QE can be established by just observing that, for all objects $o \in Obj$, o is in the set R returned by QE$(t, \{t\})$ if and only if o satisfies the two conditions expressed by Corollary 1.

$\textsc{Qe}(x : \textbf{term} \; ; \; A : \textbf{set of terms});$
1. $R \leftarrow I(x)$
2. **for each** hyperedge $\langle\{u_1, ..., u_r\}, x\rangle$ in $\mathcal{H}$ **do**
3. **if** $\{u_1, ..., u_r\} \cap A = \emptyset$ **then** $R \leftarrow R \cup (\textsc{Qe}(u_1, A \cup \{u_1\}) \cap \ldots \cap \textsc{Qe}(u_r, A \cup \{u_r\}))$
4. **return**(R)

Figure 4: The procedure $\textsc{Qe}$

Table 1: Evaluation of $\textsc{Qe}(a2, \{a2\})$

Call	Result
$\textsc{Qe}(a2, \{a2\})$	$I(a2) \cup \textsc{Qe}(b3, \{a2, b3\}) \cup (\textsc{Qe}(b1, \{a2, b1\}) \cap \textsc{Qe}(b2, \{a2, b2\}))$
$\textsc{Qe}(b3, \{a2, b3\})$	$I(b3)$
$\textsc{Qe}(b1, \{a2, b1\})$	$I(b1) \cup \textsc{Qe}(c1, \{a2, b1, c1\}) \cup \textsc{Qe}(c2, \{a2, b1, c2\})$
$\textsc{Qe}(b2, \{a2, b2\})$	$I(b2) \cup (\textsc{Qe}(c2, \{a2, b2, c2\}) \cap \textsc{Qe}(c3, \{a2, b2, c3\}))$
$\textsc{Qe}(c1, \{a2, b1, c1\})$	$I(c1)$
$\textsc{Qe}(c2, \{a2, b1, c2\})$	$I(c2) \star$
$\textsc{Qe}(c2, \{a2, b2, c2\})$	$I(c2) \cup (\textsc{Qe}(b1, \{a2, b2, c2, b1\}) \cap \textsc{Qe}(b3, \{a2, b2, c2, b3\}))$
$\textsc{Qe}(c3, \{a2, b2, c3\}))$	$I(c3)$
$\textsc{Qe}(b1, \{a2, b2, c2, b1\})$	$I(b1) \cup \textsc{Qe}(c1, \{a2, b2, c2, b1, c1\})\star$
$\textsc{Qe}(b3, \{a2, b2, c2, b3\}))$	$I(b3)$
$\textsc{Qe}(c1, \{a2, b2, c2, b1, c1\})$	$I(c1)$

As an example, let us consider the sequence of calls made by the procedure $\textsc{Qe}$ in evaluating the query $a2$ in the example source, as shown in Table 1. The calls marked with a $\star$ are those in which the test in line 3 gives a negative result. Upon evaluating $\textsc{Qe}(c2, \{a2, b1, c2\})$ the procedure realizes that the only incoming hyperedge in $c2$ is $\langle\{b1, b3\}, c2\rangle$, whose tail $\{b1, b3\}$ has a non-empty intersection with the current path $\{a2, b1, c2\}$; so the hyperedge is ignored. In this case, the cycle $(b1, c2, b1)$ is detected and properly handled. Analogously, upon evaluating $\textsc{Qe}(b1, \{a2, b2, c2, b1\})$, the cycle $(c2, b1, c2)$ is detected and properly handled. Also notice the difference between the calls $\textsc{Qe}(c2, \{a2, b1, c2\})$ and $\textsc{Qe}(c2, \{a2, b2, c2\})$. The both concern $c2$, but in the former case, $c2$ is encountered upon descending along the path $(a2, b1, c2)$ whose next hyperedge is $\langle\{b1, b3\}, c2\rangle$; following that hyperedge, would lead the computation back to the node $b1$, which has already been met, thus the result of the call is just $I(c2)$. In the latter case, $c2$ is encountered upon descending along the path $(a2, b2, c2)$, thus the hyperedge leading to $b1$ and $b3$ must be followed, since none of the terms in its tail have been touched upon so far.

From a complexity point of view, $\textsc{Qe}$ visits all terms that lie on a cycle-free simple path ending at the query term t in the taxonomy B-graph $\mathcal{H}$. As shown in Section 3.2.2, the number of such terms can be exponential in the size of the taxonomy. For each term, $\textsc{Qe}$ performs set-theoretic operations on sets of objects, which have polynomial time complexity. Thus, though $\textsc{Qe}$ operates in exponential time in the size of the taxonomy, it has polynomial time complexity in the size of the information source.

From a more practical point of view, there is an obvious alternative to $\textsc{Qe}$ for computing $ans(t, S)$, that is to solve the decision problem for each object $o \in Obj$. However, this method is not practically applicable to peer-to-peer networks, thus we do not take it into consideration any longer.

3.4 Negation

In this section we deal with negation. We first consider the addition of negation to the taxonomy of the source, then the simpler case in which negation is used in queries only.

3.4.1 Adding negation to the taxonomy

If the queries in taxonomy relationships have negation, then the source corresponds to a datalog program with rules that contain negation in their bodies, and it is well known (e.g. see [41]) that such programs may not have a unique minimal model. This is illustrated by the source shown in Figure 5: the left part shows the source taxonomy, while the right part shows the source interpretation, I, and two minimal models I_a and I_b.

$$
\begin{array}{ll}
a2 \wedge \neg a1 \longrightarrow b1 & b2 \\
a2 \qquad a1 \longleftarrow b2 \wedge \neg b1 &
\end{array}
$$

query	I	I_a	I_b
$a1$	$\emptyset$	$\{1\}$	$\emptyset$
$a2$	$\{1\}$	$\{1\}$	$\{1\}$
$b1$	$\emptyset$	$\emptyset$	$\{1\}$
$b2$	$\{1\}$	$\{1\}$	$\{1\}$
$b2 \wedge \neg b1$	$\{1\}$	$\{1\}$	$\emptyset$
$a2 \wedge \neg a1$	$\{1\}$	$\emptyset$	$\{1\}$

Figure 5: A source with no unique minimal model

The lack of a unique minimal model turns out to be a serious drawback. Let $\mathcal{L}_T^\neg$ be the language of conjunctive queries in which negations of terms may occur, *i.e.* $\mathcal{L}_T^\neg$ is given by (as usual, t is a term in T):

$$
\begin{array}{lll}
q & ::= d \mid q \vee d & (q \text{ is a } \textit{query}) \\
d & ::= l \mid l \wedge d & (d \text{ is a } \textit{disjunct}) \\
l & ::= t \mid \neg t & (l \text{ is a } \textit{literal}).
\end{array}
$$

Moreover, let $\mathcal{C}_T^\neg$ be the sub-language of $\mathcal{L}_T^\neg$ consisting of just disjuncts. A *neg-extended taxonomy* is a pair $(T, \preceq^\neg)$, where T is a terminology and $\preceq^\neg \subseteq (\mathcal{C}_T^\neg \times \mathcal{C}_T^\neg)$ is reflexive and transitive, such that if $q_1 \preceq^\neg q_2$ and $q_1 \neq q_2$, then $q_2 = t$ for some term $t \in T$. A *neg-extended source* S is a 4-tuple $(T, \preceq^\neg, Obj, I)$, where $(T, \preceq^\neg)$ is a neg-extended taxonomy and I is an interpretation for it.

It can be proved that:

Proposition 6 Deciding whether an object $o \in Obj$ is in the answer of a query $q \in \mathcal{L}_T^\neg$ in a neg-extended source S, $o \in ans(q, S)$, is a coNP-hard problem.

The proof is based on the following polynomial reduction from SAT. Let α be a CNF formula of propositional logic over an alphabet V, that is:

$$\alpha = \bigwedge_{i=1}^{n} \alpha_i \qquad \alpha_i = \bigvee_{j=1}^{m_i} l_{ij}$$

where l_{ij} is either a positive literal, that is a letter $v \in V$, or a negative literal, that is $\neg u$ where $u \in V$. We map α into a neg-extended source $S_\alpha = (T_\alpha, \preceq_\alpha, Obj_\alpha, I_\alpha)$, and a query q_α as follows:

- $T_\alpha = V$;
- $Obj_\alpha = \{1\}$;
- the query q_α is given by

  $$\bigvee \{v_1 \wedge \ldots \wedge v_k \mid \neg v_1 \vee \ldots \vee \neg v_k \text{ is a conjunct } \alpha_i \text{ in } \alpha \ (v_i \in V)\}.$$

  If there is no conjunct $\neg v_1 \vee \ldots \vee \neg v_k$ in α, then let α_1 be $l_1 \vee \ldots \vee l_k$; we then set $q_\alpha = \overline{l_1} \wedge \ldots \wedge \overline{l_k}$, where $\overline{\neg u} = u$ and $\overline{v} = \neg v$;
- for each remaining conjunct α_i in α,

  1. if α_i is a letter v, then $I_\alpha(v) = \{1\}$; if for no conjunct α_i, $\alpha_i = v$, then $I_\alpha(v) = \emptyset$;

2. if α_i is $l_1 \vee \ldots \vee l_k$ for $k \geq 2$, where at least one literal is positive, say w.l.o.g. that l_1 is the positive literal u, then the subsumption relationship $(\overline{l_2} \wedge \ldots \wedge \overline{l_k}, u)$ is in $\preceq_\alpha^r$.

For instance, the propositional formula

$$\begin{array}{rcl}\alpha & = & a2 \wedge b2 \wedge \\ & & (a1 \vee \neg a2 \vee b1) \wedge (a1 \vee b1 \vee \neg b2) \wedge \\ & & \neg a1 \wedge \neg b1\end{array}$$

is mapped into the source shown in Figure 5 and the query $q_\alpha = a1 \vee b1$. We now show the following

Lemma $1 \in ans(q_\alpha, S_\alpha)$ iff α is unsatisfiable.
In fact, we prove the equivalent form: $1 \notin ans(q_\alpha, S_\alpha)$ iff α is satisfiable.
$(\rightarrow)$ Suppose α is satisfiable, and let f be a truth assignment over V satisfying it. Let J be the interpretation of the taxonomy $(T_\alpha, \preceq_\alpha)$ such that, for each term $t \in V$,

$$J(t) = \begin{cases} \{1\} & \text{if } f(t) = T \\ \emptyset & otherwise \end{cases}$$

We have that $I_\alpha \leq J$, since for each $t \in V$, either $I_\alpha(t)$ is empty, or $I_\alpha(t) = \{1\}$. In the former case, $I_\alpha(t) \subseteq J(t)$ for any $J(t)$. In the latter case, we have that $\alpha_j = t$ for some $1 \leq j \leq n$, which implies $f(t) = T$ (since f satisfies α) which implies $J(t) = \{1\}$ and again $I_\alpha(t) \subseteq J(t)$. Moreover, $(q, u) \in \preceq_\alpha$ implies $J(q) \subseteq J(u)$. In proof, $(q, u) \in \preceq_\alpha$ iff $\alpha_k = \neg q \vee u$ for some $1 \leq k \leq n$, which implies $f(\neg q \vee u) = T$ (since f satisfies α) and therefore: either $f(\neg q) = T$ and by construction $J(q) = \emptyset$, or $f(u) = T$ and by construction $J(u) = \{1\}$; in both cases $J(q) \subseteq J(u)$. Hence J is a model of S_α. However, $1 \notin J(q_\alpha)$. In fact, by construction, for any disjunct d in q_α, there exists $\alpha_j = \neg d$ for some $1 \leq j \leq n$. Since f satisfies α, it follows that f satisfies $\neg d$ so $f(d) = F$. But then $J(d) = \emptyset$ for each disjunct d in q_α, which implies $J(q_\alpha) = \emptyset$. So, $1 \notin J(q)$ for a model J, that is $1 \notin ans(q_\alpha, S_\alpha)$.
$(\leftarrow)$ Suppose $1 \notin ans(q_\alpha, S_\alpha)$, and let J be a model of S_α such that $1 \notin J(q_\alpha)$. Let f be the truth assignment over V defined as follows, for each letter $t \in V$,

$$f(t) = \begin{cases} T & \text{if } 1 \in J(t) \\ F & otherwise \end{cases}$$

By a similar argument to the one developed in the *if* part of the proof, it can be proved that f satisfies α, and this completes the proof of the Lemma.

From the last Lemma and the NP-completeness of SAT, the coNP-hardness of deciding query re-writing in neg-extended sources follows. □

We observe that it is essential for the reduction that the query language allows negation. Otherwise, propositional formulae which do not have a conjunct consisting of all negative literals, such as $\neg v_1 \vee \ldots \vee \neg v_k$, could not be reduced.

3.4.2 Adding negation in queries

In this Section, we consider the evaluation of queries containing negation over a source. To this end, we need first to define the extension of a negative literal in an interpretation I. The obvious way of doing so is as follows: $I(\neg t) = Obj \setminus I(t)$. However, as it is well-known, if we maintain our definition of query answer, as $ans(q, S) = \{o \in Obj \mid o \in J(q)$ for all models J of $S\}$, a negative literal in a query is equivalent to the *false* clause, because there is not enough information in the taxonomy of a source to support a negative fact.

In order to derive an intuitive and, at the same time, logically well-grounded evaluation procedure for extended queries, we need an alternative query semantics (*i.e.* *ans*). In order to define it, let us consider a logical reformulation of the problem in terms of datalog. We map each term t_i into two predicate symbols:

– an extensional one, denoted $\mathtt{C_{t_i}}$, representing the interpretation of t_i, *i.e.* $I(t_{t_i})$; and

– an intensional one, denoted $\mathtt{Y_{t_i}}$, representing t_i in the rules encoding the subsumption relation.

The obvious connection between $\mathtt{C_{t_i}}$ and $\mathtt{Y_{t_i}}$ is that all facts expressed via the former are also true of the latter, and this is captured by stating a rule (named "extensional" below) of the form $\mathtt{C_{t_i}}(x) \rightarrow \mathtt{Y_{t_i}}(x)$ for each term t_i.

Definition 8 (Source program) Given a source $S = (T, \preceq, Obj, I)$, the *source program* of S is the set of clauses P_S given by $P_S = TR_S \cup ER_S \cup F_S$, where:

- $TR_S = \{\mathtt{Y_t(x) :- Y_{t_1}(x), \ldots, Y_{t_m}(x)} \mid t_1 \wedge \ldots \wedge t_m \preceq^r t\}$ are the *terminological rules* of P_S;
- $ER_S = \{\mathtt{Y_{t_i}(x) :- C_{t_i}(x)} \mid t_i \in T\}$ are the *extensional rules* of P_S;
- $F_S = \{\mathtt{C_{t_i}(o)} \mid o \in I(t_i)\}$ are the *facts* of P_S, stated in terms of constants $\mathtt{o}$ which are one-to-one with the elements of Obj (unique name assumption). □

Next, we translate queries in the language $\mathcal{L}_T$.

Definition 9 (Query program) Given a query $q \in \mathcal{L}_T$ to a simple source $S = (T, \preceq, Obj, I)$, the *query program* of q is the set of clauses P_q given by:

$$\{\mathtt{q(x) :- Y_{t_1}(x), \ldots, Y_{t_k}(x)} \mid t_1 \wedge \ldots \wedge t_k \text{ is a disjunct of } q\}.$$

where $\mathtt{q}$ is a new predicate symbol. □

In order to show the equivalence of the original model with its datalog translation, we state the following:

Proposition 7 For each source $S = (T, \preceq, Obj, I)$, and query $q \in \mathcal{L}_T$, $ans(q, S) = \{o \in Obj \mid P_S \cup P_q \models \mathtt{q(o)}\}$. □

Let us consider this mapping in light of the new query language $\mathcal{L}_T^{\neg}$. The source program P_S remains a pure datalog program, while the query program P_q of any query $q \in \mathcal{L}_T^{\neg}$ against S becomes:

$$\{\mathtt{q(x) :- L_{v_1}(x), \ldots, L_{v_k}(x)} \mid v_1 \wedge \ldots \wedge v_k \text{ is a disjunct of } q\}$$

where each $\mathtt{L_{v_i}}$ is either $\mathtt{Y_{v_i}}$, if $v_i = t_i$, or $\neg\mathtt{Y_{v_i}}$, if $v_i = \neg t_i$ $(t_i \in T)$.

This kind of queries are dealt with by using an approximation of CWA, which can be characterized either procedurally, in terms of program stratification, or declaratively, in terms of perfect model. We will adopt the former characterization. In fact, P_q is a datalog$^{\neg}$ program, and so is the program $P_S \cup P_q$. The latter program is stratified, by the level mapping l defined as follows:

$$l(pred) = \begin{cases} 1 & \text{if } pred \text{ is } \mathtt{q} \\ 0 & \text{otherwise} \end{cases}$$

It follows that $P_S \cup P_q$ has a minimal Herbrand model M_S^q given by ([12]) the least fixpoint of the transformation $T'_{P_q \cup M_{P_S}}$ where M_{P_S} is the least Herbrand model of the datalog program P_S, and T'_P is the extension to datalog$^{\neg}$ of the T_P operator, on which the standard semantics of pure datalog is based. The model M_S^q is found from M_{P_S} in one iteration since only instances of $\mathtt{q}$ are added at each iteration, and $\mathtt{q}$ does not occur in the body of any rule. The following definition establishes an alternative notion of answer for queries including negation.

Definition 10 (Extended answer) Given an extended query q to a source $S = (T, \preceq, Obj, I)$, the *extended answer to q in* S, denoted $\varepsilon(q, S)$, is given by: $\varepsilon(q, S) = \{o \in Obj \mid M_S^q \models \mathtt{q(o)}\}$ □

We conclude by showing how extended answers can be computed.

Proposition 8 For each source $S = (T, \preceq, Obj, I)$, and query $q \in \mathcal{L}_T^\neg$, $\varepsilon(q, S)$ is given by:

1. $\varepsilon(q \vee d, S) = \varepsilon(q, S) \cup \varepsilon(d, S)$,
2. $\varepsilon(l \wedge d, S) = \varepsilon(l, S) \cap \varepsilon(d, S)$,
3. $\varepsilon(t, S) = \bar{I}(t)$,
4. $\varepsilon(\neg t, S) = Obj \setminus \varepsilon(t, S)$. □

From a practical point of view, computing $\varepsilon(\neg t_1 \wedge \ldots \wedge \neg t_k)$ requires computing:

$$Obj \setminus (\bar{I}(t_1) \cup \ldots \cup \bar{I}(t_k))$$

which in turn requires knowing Obj, *i.e.* the whole set of objects of the network. As this knowledge may not be available, or may be too expensive to obtain, one may want to resort to a query language making a restricted usage of negation, for instance by forcing each query disjunct to contain at least one positive term.

3.5 Disjunctive information sources

In this section we consider disjunctive sources, whose taxonomies allow subsumption relationships between queries. Formally, a *disjunctive taxonomy* is a pair $(T, \preceq_d)$ where T is a terminology and $\preceq_d \subseteq (\mathcal{L}_T \times \mathcal{L}_T)$ is reflexive and transitive. A *disjunctive source* S is a 4-tuple $(T, \preceq_d, Obj, I)$ where $(T, \preceq_d)$ is a disjunctive taxonomy and (Obj, I) is an interpretation for it.

Disjunctive sources may not have a unique minimal model. As an example, the source $(T, \preceq_d, Obj, I)$ where:

- $T = \{a1, a2, b1, b2\}$
- $\preceq_d^r = \{(a2, a1 \vee b1), (b2, a1 \vee b1)\}$
- $Obj = \{1\}$ and
- $I = \{(a2, \{1\}), (b2, \{1\})\}$

has two minimal models, $I_1 = I \cup \{(a1, \{1\})\}$ and $I_2 = I \cup \{(b1, \{1\})\}$.

Loosing the uniqueness of the minimal model is enough to make query evaluation for this kind of sources computationally difficult.

Proposition 9 Deciding whether an object $o \in Obj$ is in the answer of a query $q \in \mathcal{L}_T$ in a disjunctive source S, $o \in ans(q, S)$, is a coNP-hard problem.

The proof is similar to that of Proposition 6. For brevity, we just show the reduction from SAT. Let α be as in the proof of Proposition 6. We map α into a disjunctive source $S_\alpha = (T_\alpha, \preceq_\alpha, Obj_\alpha, I_\alpha)$, and a query q_α as follows:

- $T_\alpha = V$;
- $Obj_\alpha = \{1\}$;
- the query q_α is given by

$$\bigvee\{v_1 \wedge \ldots \wedge v_k | \neg v_1 \vee \ldots \vee \neg v_k \text{ is a conjunct in } \alpha \ (v_i \in V)\} \vee$$
$$\bigvee\{\neg u_1 \wedge \ldots \wedge \neg u_k \mid u_1 \vee \ldots \vee u_k \text{ is a conjunct in } \alpha \ (u_i \in V)\}$$

If there are no such conjuncts $\neg v_1 \vee \ldots \vee \neg v_k$ or $\neg u_1 \wedge \ldots \wedge \neg u_k$ in α, then let α_1 be $l_1 \vee \ldots \vee l_k$; we then set $q_\alpha = \overline{l_1} \wedge \ldots \wedge \overline{l_k}$, where $\overline{\neg u} = u$ and $\overline{v} = \neg v$.

– for each remaining conjunct α_i in α,

1. if α_i is a letter v, then $I_\alpha(v) = \{1\}$; if for no conjunct α_i, $\alpha_i = v$, then $I_\alpha(v) = \emptyset$;
2. if α_i is $\neg u_1 \vee \ldots \vee \neg u_j \vee v_1 \vee \ldots \vee v_m$ where $j, m \geq 1$ then the subsumption relationship $(u_1 \wedge \ldots \wedge u_j, v_1 \vee \ldots \vee v_m)$ is in $\preceq^r_\alpha$.

In the present case, the propositional formula

$$\begin{aligned} \alpha \quad = \quad & a2 \wedge b2 \wedge \\ & (a1 \vee \neg a2 \vee b1) \wedge (a1 \vee b1 \vee \neg b2) \wedge \\ & \neg a1 \wedge \neg b1 \end{aligned}$$

is mapped into the source shown in the previous example.

It can be shown that $1 \in ans(q_\alpha, S_\alpha)$ iff α is unsatisfiable.

4 Networks of Information Sources

In this Section we introduce networks of information sources. The model is first outlined, and then query evaluation is considered.

4.1 The model

In order to be a component of a networked information system, a source is endowed with additional subsumption relations, called articulations, which relate the source terminology to the terminologies of other sources of the same kind.

Definition 11 (Articulation) Given two terminologies T and U, an *articulation* from T to U, $\preceq_{TU}$, is a non-empty binary relation from $\mathcal{L}_U$ to T, such that $q \preceq_{TU} t$ implies that q is a conjunctive query. □

An articulation relationship is not syntactically different from a subsumption relationship, except that its head may be a term of a different terminology than the one where the terms making up its tail come from.

Definition 12 (Articulated Source) An *articulated source* $\mathcal{S}$ over $k \geq 0$ disjoint terminologies $T_1, ..., T_k$, is a 5-tuple $\mathcal{S} = (T_\mathcal{S}, \preceq_\mathcal{S}, Obj, I_\mathcal{S}, R_\mathcal{S})$, where:

– $(T_\mathcal{S}, \preceq_\mathcal{S}, Obj, I_\mathcal{S})$ is a source;

– $R_\mathcal{S}$ is a set of articulations $R_\mathcal{S} = \{\preceq_{T_\mathcal{S},T_1}, \ldots, \preceq_{T_\mathcal{S},T_k}\}$. □

Articulations are used to connect an articulated source to other articulated sources, so creating a networked information system. An articulated source $\mathcal{S}$ with an empty stored interpretation, *i.e.* $I_\mathcal{S}(t) = \emptyset$ for all $t \in T_\mathcal{S}$, is called a *mediator* in the literature.

Definition 13 (Network) A *network of articulated sources,* or simply a *network*, $\mathcal{N}$ is a non-empty set of articulated sources $\mathcal{N} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, where each $\mathcal{S}_i$ is articulated over the terminologies of some of the other sources in $\mathcal{N}$ and all terminologies $T_{\mathcal{S}_1}, \ldots, T_{\mathcal{S}_n}$ of the sources in $\mathcal{N}$ are disjoint. □

Notice that the domain of the interpretation of an articulated source is independent from the source, thus the same for any articulated source. This is not necessary for our model to work, just reflects a typical situation of networked resources such as URLs. Relaxing this constrain would have no impact on the results reported in the present study.

Since in a network: (a) there is no source acting at the global level, (b) all sources store data, and (c) as we will see, data are exchanged via direct communication, each source can be seen as, and will in fact be called, a *peer,* and the network as a *peer-to-peer* information system. Articulations of the network peers will also be referred as *P2P mappings.*

An intuitive way of interpreting a network is to view it as a single source which is distributed along the nodes of a network, each node dealing with a specific vocabulary. The global source can be logically constructed by removing the barriers which separate local sources, as if (virtually) collecting all the network information in a single repository. The notion of *network source* captures this interpretation of a network.

Definition 14 (Network source) The *network source* $S_\mathcal{N}$ of a network of articulated sources $\mathcal{N} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, is the source
$S_\mathcal{N} = (T_\mathcal{N}, \sqsubseteq, Obj, I_\mathcal{N})$, where:

- $T_\mathcal{N} = \bigcup_{i=1}^n T_{\mathcal{S}_i}$;
- $I_\mathcal{N} = \bigcup_{i=1}^n I_{\mathcal{S}_i}$
- $\sqsubseteq = (\bigcup_{i=1}^n \sqsubseteq_{\mathcal{S}_i})^*$

where $\sqsubseteq_{\mathcal{S}_i}$ is the *total subsumption* of the source $\mathcal{S}_i$, given by the union of the subsumption relation $\preceq_{\mathcal{S}_i}$ with all articulations of the source, that is:

$$\sqsubseteq_{\mathcal{S}_i} = \preceq_{\mathcal{S}_i} \cup \bigcup R_{\mathcal{S}_i}$$

and A^* denotes the transitive closure of the binary relation A. A *network query* is a query over $T_\mathcal{N}$. □

It is not difficult to see that $\sqsubseteq$ is reflexive and transitive, and every non-trivial subsumption relationship in it relates a conjunctive query in anyone of the terminologies $T_{\mathcal{S}_1}, \ldots, T_{\mathcal{S}_n}$ to a single term. Thus, $S_\mathcal{N}$ is indeed a source. Such source emerges in a bottom-up manner from the articulations of the peers. This distinguishes peer-to-peer systems from federated distributed databases.

A *network query* q is a query in anyone of the query languages supported by the network, that is $q \in \mathcal{L}_{T_{\mathcal{S}_i}}$ for some $i \in [1, n]$. As it will be evident, the method that we will set up only requires minor modifications to be able to evaluate also queries in the language $\mathcal{L}_{T_\mathcal{N}}$, that is queries that mix terms from different terminologies. We do not provide this facility because it does not seem to make much sense in our vision.

The answer to a network query q, or *network answer,* is given by $ans(q, S_\mathcal{N})$.

Figure 6 presents the taxonomy of a network source $S_\mathcal{N}$, where $\mathcal{N}$ consists of 3 peers $\mathcal{N} = \{P_a, P_b, P_c\}$. As it can be verified, this is the same taxonomy as the one shown in Figure 1, except that now some of its subsumption relationships are elements of articulations.

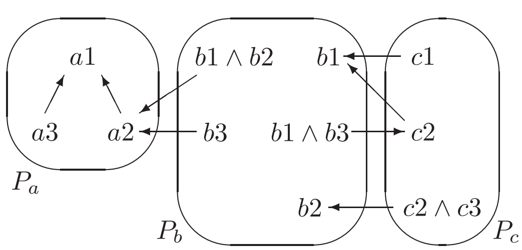

Figure 6: A network taxonomy

4.2 Network query evaluation

This Section presents a network query evaluation procedure based on the method devised in the centralized case. First, a functional model of each peer is introduced, then the algorithms corresponding to the operations on the interface of the peer are given. Correctness and complexity of these algorithms are discussed in Section 4.3, while Section 5 concludes by considering optimization issues.

4.2.1 The functional model of a peer

In order to illustrate our query evaluation procedure, we now define a peer from a functional point of view. In this respect, we see a peer as a software component uniquely identified in the network by a peer ID. The interface of a peer exposes just one method:

- QUERY, which takes as input a network query q and evaluates it, returning the set of objects $ans(q, S_{\mathcal{N}})$.

The user (whether human or application program) is supposed to use this method for the evaluation of network queries. We assume that q is expressed in the query language of the peer. As it will be argued in due course, this assumption can be relaxed without any substantial change to our framework.

In addition to QUERY, a peer has methods for sending to or receiving messages from other peers. We do not enter into the details of these methods: there are several options, which do not make any difference from the point of view of our model. Instead, we detail the types of messages that can be exchanged between peers. These can be of one of the following 2 types:

- ASK: by sending a message of this kind to a peer P, the present peer asks P to evaluate a term query on P's query language. The receiving peer P processes ASK messages according to the QE procedure (Figure 4), as we will see in detail below. An ASK message has the following fields:
  - *PID*: the id of the present peer, which is sending the message;
  - *QID*: the id of the query that *PID* is sending for evaluation;
  - *t:* the query term of *QID*;
  - *A:* the set of already visited terms. These two last parameters are those of the QE procedure.
- TELL: by sending a message of this kind to a peer P, the present peer returns to P the result of the evaluation of a term query which had previously been ASK-ed by P. A TELL message has the following fields:
  - *QID*: the ID of the query whose result is being returned;
  - *RES:* the set of objects resulting from the evaluation of *QID*.

We will denote the sending of a message of one of these two kinds m to the peer P as $P:$*m(field values)*. By decoupling the request of evaluation from the return of the result, we aim at minimizing the number of sessions open at any time between peers, thus removing a serious obstacle towards scalability. QUERY does not follow this paradigm since it involves only a local interaction.

Each peer processes the incoming messages depending on their type and content. In order to carry out this work, the peer keeps a *(query) log,* that is a set of objects, each associated to a query in whose evaluation the peer is currently involved. A log object has the following attributes:

- *PID*: the id of the peer who sent the query (can be the local peer itself);
- *QID*: the id of the query;
- *t:* the query term (we recall that we need to deal only with term queries);
- *n:* the number of open calls in *QID* (see next paragraph);

- QP: the query program representing the current status of evaluation of *QID*. A query program is a set of *sub-programs* $\{SP_1, \ldots, SP_k\}$ where each sub-program SP_j is a set of *calls*. A call is a sub-query of *QID*, and can be:
  - *open,* meaning that the sub-query is being evaluated, in which case the call is the sub-query id; or
  - *closed,* meaning the sub-query has been evaluated, in which case the call is the resulting set of objects.

Since no two log objects can have the same query id, we will represent a log object as a 5-tuple (PID, QID, t, n, QP).

4.2.2 Query

Let us assume that the input query q posed to a peer $\mathcal{S}$, is given by

$$q = \bigvee C_i$$

where each C_i is a conjunctive query. As a first step, Query reduces q to a term query t by generating a new term t not in $T_\mathcal{N}$ and inserting a new hyperedge (C_i, t) into the local taxonomy B-graph (i.e. that corresponding to $(T_\mathcal{S}, \sqsubseteq_\mathcal{S})$), for each conjunctive query C_i in q. This work is carried out by the function Modify-taxonomy, which returns the newly generated term t. A new query id for t is subsequently obtained by Query, and an Ask message is sent to the peer itself for evaluating t. As required by Qe, the set of already visited terms consists just of t itself. At this point Query hangs on the log, until the log object associated to the query t is closed, that is the number of its open call is 0. Notice that this object is created only after the Ask message sent on line 3 is processed, but this creates no problem, as all Query has to do in the meantime is wait. When the log object is finally closed, Query retrieves it and deletes it from the log, by using the function Delete, which returns the object itself. When the object is closed, its query program, that is the value of the last field, equals to $ans(t, S_\mathcal{N})$. This value is assigned to the variable R. On line 6, the subsumption relationships inserted by Modify-taxonomy are removed by Cleanup-taxonomy, and R is finally returned.

Query(q : **query**);
1. $t \leftarrow$ Modify-taxonomy(q)
2. $ID \leftarrow$ New-query-id
3. *self*: Ask(*self*, ID, t, $\{t\}$)
4. **wait until** ID is closed **then**
5. (PID, QID, t, n, R) $\leftarrow$ Delete(ID)
6. Cleanup-taxonomy(t)
7. **return**(R)

Figure 7: The Query procedure

As an example, let us consider the network shown in Figure 6, whose corresponding B-graph is shown in Figure 8, and the query $(a2 \wedge a3)$ on peer P_a. When given as input to Query, this query is passed on to Modify-taxonomy, which adds the hyperedge $(\{a2, a3\}, t)$ to the taxonomy B-graph and returns the newly generated term t. Let us assume that $q1$ is the id of the new query. Query then sends the message Ask(P_a, $q1$, t, $\{t\}$) to itself, and gets into the wait loop until the query is evaluated.

4.2.3 Ask

For readability, we will describe Ask and Tell as if they were methods whose parameters are the message fields. Ask (Figure 9) uses the following variables:

- n : counts how many sub-queries the input query *QID* generates;

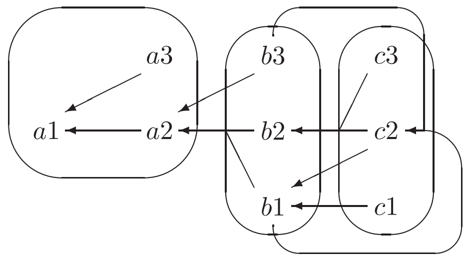

Figure 8: A network taxonomy B-graph

- QP: is the initial query program of QID;
- Q : is a queue holding the information to send the ASK messages required to evaluate QID;
- C : is the query sub-program being currently computed.

After initialization, ASK performs (line 2) the same test as QE, looking for a hyperedge h in the local B-graph whose head is the given term t and whose tail is disjoint form A. If no such hyperedge is found, then n remains 0, the test on line 10 fails, and the result of the evaluation of the given term query t is just $I(t)$ (as QE establishes), which ASK returns by sending a TELL message to the invoking peer PID (line 15). If instead a hyperedge h is found, then the intersection of the evaluation of each term u_i in its tail should be added to the result, according to QE. In order to achieve the same behavior, ASK enters a loop in which it processes each term u_i to the end of constructing in C the query sub-program associated to h. First, a new query id ID is generated (line 5) to denote the sub-query on u_i; the newly generated id is then added to C. On line 7, the number of open calls is increased by one, and on line 8 the required information to evaluate the query u_i is enqueued in Q. This information is:

- the id of the peer P_h holding the terms in the tail of the hyperedge h; we assume this information is stored with the hyperedge just for convenience, the peer can also store it separately;
- the ID of the sub-query;
- the query term u_i and
- the set of the visited terms $A \cup \{u_i\}$, as in QE.

Each sub-program so generated is added to QP, after considering all relevant hyperedges (line 9). At this point, if the number of open calls is positive, ASK uses the function PERSIST in order to create the log object representing the query QID, and to persist it in the log. Once the log object is successfully persisted, ASK must launch the evaluation of the generated sub-queries, which it does in the loop on lines 12-14. Until Q is empty, it dequeues the information for constructing an ASK message for each sub-query, and sends such message to the peer P_h. The value of the first message field is the peer identity (*self*), as the invoking peer.

At this point, it can be easily verified that the assumption that all terms in the tail of a hyperedge are from the same terminology, namely that of peer P_h, *can be relaxed without any impact on the query evaluation procedure.* In logical terms, this is the assumption that the conjunctive queries on the left-hand side of subsumption relationships are from the query language of one peer. We have made this assumption because it fits our vision of a network. But ASK can easily work also with hyperedges whose tails have terms from different terminologies: all that is required is to store the id of the peer holding each term, rather than the id of the peer holding the whole hyperedge.

Let us resume our running example. Upon processing the message (P_a, $q1$, t, $\{t\}$), ASK finds that the hyperedge $h = (\{a2, a3\}, t)$ passes the test on line 2, and enters the loop on the tail of h. For term $a2$, assuming the generated query id is $q2$, the record (P_a, $q2$, $a2$, $\{t, a2\}$) is enqueued in Q, while for term $a3$, (generated id $q3$) it is enqueued the record (P_a, $q3$, $a3$, $\{t, a3\}$). As there are no more hyperedges and $n = 2$, a new log object is created to represent the query t. The attributes of this object are:

ASK(PID,QID: **ID**; t : **term**; A : **set of terms**);
1. $n \leftarrow 0$; QP, $Q \leftarrow \emptyset$
2. **for each** hyperedge $h = \langle \{u_1, ..., u_r\}, t \rangle$ **such that** $\{u_1, ..., u_r\} \cap A = \emptyset$ **do**
3. $\quad C \leftarrow \emptyset$
4. $\quad$ **for each** u_i **do**
5. $\quad\quad ID \leftarrow$ NEW-QUERY-ID
6. $\quad\quad C \leftarrow C \cup \{ID\}$
7. $\quad\quad n \leftarrow n + 1$
8. $\quad\quad$ ENQUEUE(Q, (P_h, ID, u_i, $A \cup \{u_i\}$))
9. $\quad QP \leftarrow QP \cup \{C\}$
10. **if** $n > 0$ **then**
11. $\quad$ PERSIST(PID,QID,t,n,QP)
12. $\quad$ **until** $Q \neq \emptyset$ **do**
13. $\quad\quad$ (P_h,ID,u,B) $\leftarrow$ DEQUEUE(Q)
14. $\quad\quad P_h$: ASK($self$, ID, u, B)
15. **else** PID:TELL(QID,$I(t)$)

Figure 9: The procedure to process ASK messages

- $PID = P_a$
- $QID = q1$
- $t = t$
- $n = 2$
- $QP = \{\{q2, q3\}\}$.

Now two ASK messages are send to P_a :

1. (P_a, $q2$, $a2$, $\{t, a2\}$), and
2. (P_a, $q3$, $a3$, $\{t, a3\}$).

Let us see how the latter message is processed. Since there are no incoming hyperedges into term $a3$, n remains 0, and the processing of the message is concluded by the sending of the message TELL($q3$, $I(a3)$) to P_a.

4.2.4 Tell

When a peer receives a TELL(QID,R) message (see Figure 10), QID is an open call of some log object in the peer's log, in the program of some term (sub)query t with id QID_1. Then, as a first action, the peer retrieves this object by using the DELETE1 function, which takes as input QID, returns the object and *deletes* it form the log. Notice that there is exactly one object having QID as open call, since ASK generates a new id for each sub-query it identifies, as we have already seen. After retrieving the log object, TELL uses CLOSE to modify the query program QP in it, by closing the open call QID: this means to replace QID by R, obtaining a new query program QP_1. On line 3, the number of open calls of the log object is tested: if it is 1, then the just closed call was the last one to be open in query QID_1; in this case, the result of QID_1 is computed in S by COMPUTE-ANSWER. For a given program:

$$QP = \{SP_1, \ldots, SP_m\}$$

where each sub-program SP_j is given by a collection of object sets:

$$SP_j = \{R_1^j, \ldots, R_{m_j}^j\}$$

COMPUTE-ANSWER returns:

$$S = \bigcup \{ \bigcap SP_j \mid 1 \leq j \leq m \}$$

$S \cup I(t)$ is exactly what the QE procedure computes. If t is not in the terminology of the peer ($t \notin T_{self}$) then it follows that QID_1 is the id of the original query q. Thus, $I(t) = \emptyset$ and $S = ans(t, S_{\mathcal{N}})$. Therefore, the object (PID, QID_1, $t, 0, S$) is persisted in the log (line 5), indicating to QUERY(q) (Figure 7) that the evaluation of the query q has finished. Otherwise, the so obtained result $S \cup I(t)$ is TELL-ed to the peer PID which, according to the log object, was the one to ASK the evaluation of QID_1. Notice that this may fire another TELL message, in case QID_1 is the last open call of some other query. If the test on line 3 fails, then there are still open calls in the log object, which is therefore persisted back by PERSIST on line 6, after decreasing the number of open calls in it and replacing the query program QP by the updated one QP_1.

TELL(QID: **ID**; R : **set of objects**);
1. (PID, QID_1, t, n, QP) $\leftarrow$ DELETE1(QID)
2. $QP_1 \leftarrow$ CLOSE(QP, QID, R)
3. **if** $n = 1$ **then**
4. $\quad$ $S \leftarrow$ COMPUTE-ANSWER(QP_1)
5. $\quad$ **if** $t \notin T_{self}$ **then** PERSIST(PID, QID_1, $t, 0, S$)
6. $\quad$ **else** PID:TELL$_c$(QID_1, $t, S \cup I(t)$)
7. **else** PERSIST(PID, QID_1, t, $n - 1$, QP_1)

Figure 10: The procedure to process TELL messages

In our example, the message TELL($q3$, $I(a3)$) is received by peer P_a. The function DELETE1 returns the log object (P_a, $q1$, t, 2, $\{\{q2, q3\}\}$), the only one that has the open call $q3$. CLOSE produces the new query program $\{\{q2, I(a3)\}\}$, and since n is not 1, the following modified log object is persisted:

$$(P_a,\ q1,\ t{,}1{,}\{\{q2, I(a3)\}\}).$$

The example is completed in appendix.

4.3 Correctness and complexity

As it has been argued, the combined action of the procedures processing ASK and TELL messages is equivalent to the behavior of the procedure QE. To see why in more detail, it suffices to consider the following facts:

1. An ASK message is generated for each recursive call performed by QE and vice-versa, that is whenever QE would perform a recursive call, an ASK message is generated. This is guaranteed by the fact that the test on line 2 of ASK is the same as the test on line 3 of QE. Therefore, the number of ASK messages is the same as the number of terms that can be found on a B-path from t.
2. For each ASK message, at most one log object is generated and persisted.
3. For each ASK message, a TELL message results, and no more. This can be observed by considering that, for each processed ASK message, there can be two cases:
   (a) no hyperedge is found that passes the test on line 2 of ASK: in this case, no subsequent ASK message is generated, and a TELL message is generated;
   (b) at least one hyperedge passes the test: in this case a number of sub-queries is generated and registered in the query program of the log object. Each such sub-query is evaluated by issuing an ASK message with a larger set of visited terms. Since the B-graph is finite, eventually each sub-query will lead to a term falling in the previous case (this is how QE terminates). When all sub-queries of a given term query t are closed, the number of open calls of t goes down to 0, and TELL issues another TELL message on t. This will propagate closure up, until all open calls are closed.
4. Finally, the COMPUTE-ANSWER procedure performs the same operation on the result of sub-queries as QE does on the results of its recursive calls.

As a consequence of these facts we have the correctness of the network query evaluation procedure, and also its efficiency. In fact, the total number of messages generated is twice the number of terms visited by QE, and the number of log objects is no larger than that.

5 Optimization issues

So far, we have focused on correctness. In this Section we discuss optimization. There are many techniques that are potentially useful to this end. For instance, when sub-queries return large results, their closing (performed by CLOSE) and the computation of their results (COMPUTE-ANSWER) should be done with care. However, dealing with all the relevant optimization techniques goes beyond the scope of this paper. Instead, we focus on caching (Section 5.1), which is applicable to all situations, and on exploiting data structures employed in structured P2P systems, namely Distributed Hash Tables (like in Chord [36]). This latter issue is tackled in Sections 5.2 and 5.3; besides showing how to further improve the efficiency of the system, the ensuing discussion hints at how to extend the applicability of our model, and highlights the relationship with a large part of the literature on P2P systems. More on related work can be found in Section 6.

5.1 Caching

A strong point of our model is that the adoption of caches could significantly speed up the evaluation of queries, by reducing both the latency time and the network throughput. This is because the set of queries that a peer can send to its articulated peers is bounded in size and can be pre-determined: it comprises all "foreign" queries of the peer, *i.e.* queries that appear as left-hand sides in the peer's articulations. Note that the number of queries that a peer can propagate to its neighbors is unbounded in other models of P2P systems, for example in Gnutella, where each peer propagates whatever query it receives[4]. It follows that the caches of our model will enjoy higher hit ratios compared to other P2P models, for the same cache size. The subsequent subsections present three caching policies, namely:

- caching answers of local terms,
- caching answers of local terms and pushing answers of articulation tails, and
- caching answers of articulation heads.

5.1.1 Caching answers of local terms

According to this caching policy, each peer $\mathcal{S}$ caches pairs of the form $(t, ans(t, S_{\mathcal{N}}))$, where t is a term in the peer's terminology $T_{\mathcal{S}}$. If there are no memory limitations for caches, then after a while each peer will have cached its whole terminology, and query evaluation reduces to locally calculating the extension of the query by union-ing and intersecting the extensions of the peer's terms. In other words, any peer will be able to evaluate network queries over its own taxonomy without sending any message to the network[5]! This is of course the idealistic case. In general, only some terms (possibly none) will be cached in each peer. Under these circumstances, when a peer $\mathcal{S}$ receives an ASK message for a term query t, the ASK procedure checks which of the answers for the term (sub)queries needed for the evaluation of t are in the cache, and issues ASK messages only for evaluating the remaining terms.

The modified query evaluation algorithms for supporting this caching policy are parts of the algorithms for the more general policy that is described in Section 5.1.2.

[4] FreeNet tries to improve the situation by forwarding queries (and new objects too) only to those peers that, according to the contents of the cache, have similar keys. In this way, each cache tends to have entries about similar keys and this tends to improve the quality of routing over time.

[5] Apart those required for re-evaluating queries when updates occur.

ASK_c(*PID*,*QID*: **ID**; t : **term**; A : **set of terms**);

1. **if** t is cached **then** *PID*:$\text{TELL}_c(QID, t, ans(t, S_{\mathcal{N}}))$
2. **else if** $|A| = 2$ **then** add t into TO-BE-CACHED log // t is a term of the original query q
3. $n \leftarrow 0$; QP, Q, $S \leftarrow \emptyset$
4. **for each** hyperedge $h = \langle\{u_1, ..., u_r\}, t\rangle$ **such that** $\{u_1, ..., u_r\} \cap A = \emptyset$ **do**
5. **if** $P_h \neq self$ and $u_1 \wedge \ldots \wedge u_r$ is cached **then** $C \leftarrow \{ans(u_1 \wedge \ldots \wedge u_r, S_{\mathcal{N}})\}$
6. **else** $C \leftarrow \emptyset$
7. **for each** u_i **do**
8. **if** $P_h = self$ and u_i is cached **then**
9. $C \leftarrow C \cup \{ans(u_i, S_{\mathcal{N}})\}$
10. **else**
11. $ID \leftarrow$ NEW-QUERY-ID
12. $C \leftarrow C \cup \{ID\}$
13. $n \leftarrow n + 1$
14. ENQUEUE(Q, (P_h, ID, u_i, $A \cup \{u_i\}$))
15. $QP \leftarrow QP \cup \{C\}$
16. **if** $n > 0$ **then**
17. PERSIST(*PID*,*QID*,t,n,QP)
18. **until** $Q \neq \emptyset$
19. (P_h,ID,u,B) $\leftarrow$ DEQUEUE(Q)
20. P_h:ASK_c(*self*, ID, u, B)
21. **else if** $QP \neq \emptyset$ **then** $S \leftarrow$ COMPUTE-ANSWER(QP)
22. *PID*:$\text{TELL}_c(QID, t, S \cup I(t))$

Figure 11: The procedure to process ASK messages with cache

5.1.2 Caching answers of local terms and pushing answers of articulation tails

A complementary scenario, best suited for a P2P system that offers recommendation services in push-style manner, is to assume that each peer $\mathcal{S}$ knows also the articulations $t_1 \wedge \ldots \wedge t_r \preceq u$ from other peers $\mathcal{S}'$ to $\mathcal{S}$ (called *foreign articulations*). In this case, if all the terms $t_1, \ldots, t_r$ are cached in $\mathcal{S}$, then $\mathcal{S}$ can send to $\mathcal{S}'$ the pair $(t_1 \wedge \ldots \wedge t_r, ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}}))$ to be stored in the cache of $\mathcal{S}'$. This can be done because from Proposition 3 and Definition 4 it follows that

$$ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}}) = \bigcap\{ans(t_i, S_{\mathcal{N}}) \mid 1 \leq i \leq r\}$$

The cache is exploited by the modified ASK procedure (ASK_c), shown in Figure 11. The modified with caching TELL procedure (TELL_c) is shown in Figure 12. The modifications are indicated by bold line numbers and are described in a semi-formal way, in order to abstract from irrelevant details.

The cache of a peer $\mathcal{S}$ consists of two kinds of pairs:

– $(t', ans(t', S_{\mathcal{N}}))$ where t' is a term in the peer's terminology $T_{\mathcal{S}}$. Pairs of this kind are inserted into the cache by the $\text{TELL}_c(QID, t', R)$ procedure[6], when the peer $\mathcal{S}$ is TELL-ed the answer R for a term query t', initiated by an ASK message of type

$$\text{ASK}(PID,\ QID,\ t',\ \{u, t'\})$$

where u is a new term created by QUERY(q) to represent the original (complex) query q, posed to peer $\mathcal{S}$. This means that the term t' appears in q and is not evaluated in the context of the evaluation of a more general term. For example, this is the case of the ASK messages presented at the end of Section 4.2.3:

1. (P_a, $q2$, $a2$, $\{t, a2\}$), and

[6] Note that $\text{TELL}_c(QID, t', R)$ takes an extra argument t', which is the term query corresponding to query id QID.

Tell_c(QID: **ID**; t' : **term**; R : **set of objects**);
1. **if** t' in TO-BE-CACHED log **then** // t' is a term of the original query q
2. delete t' from TO-BE-CACHED log
3. CACHE(t', R)
4. **for each** foreign articulation $t_1 \wedge \ldots \wedge t_r \preceq u$ from another peer $\mathcal{S}$ to $self$ **do**
5. **if** $t' \in \{t_1, ..., t_r\}$ and all t_1, ..., t_r are cached **then**
6. forward to $\mathcal{S}$ the pair $(t_1 \wedge \ldots \wedge t_r, ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}})$ for caching
7. (PID, QID_1, t, n, QP) $\leftarrow$ $\text{Delete1}(QID)$
8. $QP_1 \leftarrow$ Close(QP, QID, R)
9. **if** $n = 1$ **then**
10. $S \leftarrow$ $\text{Compute-answer}(QP_1)$
11. **if** $t \notin T_{self}$ **then** Persist(PID, QID_1, $t, 0, S$)
12. **else** PID:Tell_c(QID_1, $t, S \cup I(t)$)
13. **else** Persist(PID, QID_1, t, $n-1$, QP_1)

Figure 12: The procedure to process Tell messages with cache

2. (P_a, $q3$, $a3$, $\{t, a3\}$).

In this way, based on the correctness of the QUERY procedure (Section 4.3), it is guaranteed that $R = ans(t', S_{\mathcal{N}})$, *i.e.* the received answer R is the full answer for t' and not a subset of it, reduced due to cycles in the taxonomy $(T_{\mathcal{N}}, \preceq_{\mathcal{N}})$. Thus, the pair (t', R) can be safely cached.

– $(t_1 \wedge \ldots \wedge t_r, ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}}))$ where $t_1 \wedge \ldots \wedge t_r \preceq u$ is an articulation from $\mathcal{S}$ to $\mathcal{S}'$, *i.e.* $u \in T_{\mathcal{S}}$ and $t_1, \ldots, t_r \in T_{\mathcal{S}'}$. Each such pair is forwarded to $\mathcal{S}$ by the Tell_c procedure executed at the peer $\mathcal{S}'$, upon realizing that all the terms involved in the left-hand side of the articulation are stored in the local (to $\mathcal{S}'$) cache. In particular, this check is made immediately after a pair $(t', ans(t', S_{\mathcal{N}}))$ is added in the cache of $\mathcal{S}'$, where $t' \in \{t_1, ..., t_r\}$ (see lines 3-6 of Tell_c).

Below are the main differences of $\text{Ask}_c(PID, QID, t, A)$ with respect to the cache-less Ask:

– If the answer to the term query t Ask-ed by peer PID is in the cache, then the answer is immediately Tell-ed to peer PID. Otherwise, if $|A| = 2$ then t is added in the TO-BE-CACHED log (t is a term of the original query q). The TO-BE-CACHED log is checked by $\text{Tell}_c(QID, t', R)$. If t' is found in the TO-BE-CACHED log then (t', R) is added to the local cache through the CACHE(t', R) command (line 3 of Tell_c).

– Before processing the tail of a hyperedge h which passes the test on line 4, a test is performed, to ascertain whether the query corresponding to the tail, given by $u_1 \wedge \ldots \wedge u_r$, is in the cache (this test is needed only if $P_h \neq self$, *i.e.* h corresponds to an articulation hyperedge). If yes, the only action taken is the insertion of $ans(u_1 \wedge \ldots \wedge u_r, S_{\mathcal{N}})$ into the query sub-program QP being built (line 15). If the query is not in the cache, then for each u_i, it is checked if its answer is in the cache (this test is needed only if $P_h = self$). If not, then the execution proceeds normally.

– If all sub-queries are cached, then when all relevant hyperedges have been processed (line 16), n is zero but QP is not empty. In this case the test on line 21 is passed, and the result of QID is computed in S as if closing QP in a Tell. S is subsequently returned along with $I(t)$. If QP is empty, then no hyperedge has been found and $S = \emptyset$. So, the result returned to the user is simply $I(t)$.

We would like to note that our algorithms can further be extended such that Tell_c caches the answer $S \cup I(t)$ for term sub-queries t before Tell-ing them to the requesting peer PID (line 12 of Tell_c), as long as it is certain that $S \cup I(t) = ans(t, S_{\mathcal{N}})$. This is the case if (i) for each term u of a peer $\mathcal{S}'$ encountered during the evaluation of t (including t itself), all hyperedges $\langle \{u_1, ..., u_r\}, u \rangle$ of the taxonomy B-graph of $\mathcal{S}'$ pass the test of line 4 of Ask_c, or (ii) u is cached. Thus, (i) no evaluation path of u is eliminated due to cycles in the taxonomy $(T_{\mathcal{N}}, \preceq_{\mathcal{N}})$ or (ii) $ans(u, S_{\mathcal{N}})$ is immediately retrieved from the cache.

ASK_c^{ext}(*PID*,*QID*: **ID**; t : **term**; A : **set of terms**);
1. **if** t is cached **then** *PID*:TELL_c^{ext}(*QID*, t, $ans(t, S_{\mathcal{N}})$, `full`)
2. **else if** $|A| = 2$ **then** add t into TO-BE-CACHED log // t is a term of the original query q
3. $n \leftarrow 0$; *QP*, *Q*, $S \leftarrow \emptyset$; $flag$ =`full`
4. **for each** hyperedge $h = \langle\{u_1, ..., u_r\}, t\rangle$ **do**
5. **if** $\{u_1, ..., u_r\} \cap A = \emptyset$ **then**
6. **if** $P_h \neq self$ and $u_1 \wedge \ldots \wedge u_r$ is cached **then** $C \leftarrow \{ans(u_1 \wedge \ldots \wedge u_r, S_{\mathcal{N}})\}$
7. **else** $C \leftarrow \emptyset$
8. **for each** u_i **do**
9. **if** $P_h = self$ and u_i is cached **then**
10. $C \leftarrow C \cup \{ans(u_i, S_{\mathcal{N}})\}$
11. **else**
12. $ID \leftarrow$ NEW-QUERY-ID
13. $C \leftarrow C \cup \{ID\}$
14. $n \leftarrow n + 1$
15. ENQUEUE(Q, (P_h, *ID*, u_i, $A \cup \{u_i\}$))
16. $QP \leftarrow QP \cup \{C\}$
17. **else** $flag$ =`partial`
18. **if** $n > 0$ **then**
19. PERSIST(*PID*, *QID*, t, n, *QP*, $flag$)
20. **until** $Q \neq \emptyset$
21. (P_h,*ID*,u,B) $\leftarrow$ DEQUEUE(Q)
22. P_h:ASK_c^{ext}(*self*, *ID*, u, B)
23. **else if** $QP \neq \emptyset$ **then** $S \leftarrow$ COMPUTE-ANSWER(*QP*)
24. *PID*:TELL_c^{ext}(*QID*, t, $S \cup I(t)$, $flag$)

Figure 13: The extended procedure to process ASK messages with cache

For this reason PERSIST(*PID*,*QID*,t,n,*QP*) and TELL_c(*QID*,t',R) should be extended with an extra field $flag$ that takes the values `full` or `partial`. A (query) log object (*PID*,*QID*,t,n, *QP*, $flag$), where $flag$ =`full`, of a peer $\mathcal{S}$ indicates that (i) for all *closed* term sub-queries of *QP*, full answers have been received and (ii) all hyperedges $\langle\{u_1, ..., u_r\}, t\rangle$ of the taxonomy B-graph of $\mathcal{S}$ have passed the test of line 4 of ASK_c. If this is not the case, $flag$ =`partial`. A message $\text{TELL}_c(QID, t', R, flag)$, where $flag$ =`full`, indicates that $R = ans(t', S_{\mathcal{N}})$, whereas a message $\text{TELL}_c(QID, t', R, flag)$, where $flag$ =`partial`, indicates that $R \subseteq ans(t', S_{\mathcal{N}})$. Thus, based on the $flag$ information, the TELL_c procedure executed at a peer will always be able to know if the computed answer $S \cup I(t)$ for a term sub-query t requested by peer PID is a full or partial answer. In the case of a full answer and if t is the head of an articulation hyperedge then $(t, S \cup I(t))$ is cached. We want to note that the latter condition is not a strong condition and is needed only in order to reduce the cache size, while taking the most advantage of caching.

The extended ASK_c procedure (ASK_c^{ext}) and the extended TELL_c procedure (TELL_c^{ext}) are given in Figures 13 and 14, respectively. The modifications are indicated by bold line numbers. Note that TELL_c^{ext} calls the procedure CACHE&FORWARD (Figure 15), when a pair $(t, ans(t, S_{\mathcal{N}}))$ is going to be stored in the cache. Additionally, TELL_c^{ext} uses the function $min(flag, flag')$ (lines 8, 11), which returns the minimum of the flag values $flag$, $flag'$, based on the ordering `partial` $\leq$ `full`. This guarantees that the flag value of the TELL_c^{ext} message in line 8 and the log object in line 11 is correct.

5.1.3 Caching answers of articulation heads

The previous algorithms will cache the most frequently used terms, taking full advantage of caching with no extra cost for computing cached answers. However, caches may get filled very quickly. Below we investigate the case that we cache only the heads of articulation hyperedges, as the cached answer of these terms is the most beneficial for speeding-up query answering. For instance, in the example of Figure 8, we want to cache only a_2 on Peer P_a, b_1 and b_2 on Peer P_b, and c_2 on Peer P_c.

For this alternative caching case, a top algorithm can be easily designed such that whenever a peer

TELL_c^{ext}(QID: **ID**; t' : **term**; R : **set of objects**; $flag'$: {`full`, `partial`});
if t' in TO-BE-CACHED log **then** // t' is a term of the original query q
CACHE&FORWARD(t', R)
(PID, QID_1, t, n, QP, $flag$) $\leftarrow$ DELETE1(QID)
$QP_1 \leftarrow$ CLOSE(QP, QID, R)
if $n = 1$ **then**
$S \leftarrow$ COMPUTE-ANSWER(QP_1)
if $t \notin T_{self}$ **then** PERSIST(PID, QID_1, t, 0, S, `full`)
else PID:TELL_c^{ext}(QID_1, t, $S \cup I(t)$, $min(flag, flag')$)
if $min(flag, flag')$=`full` and t is the head of an articulation hyperedge **then**
CACHE&FORWARD($t, S \cup I(t)$)
else PERSIST(PID, QID_1, t, $n-1$, QP_1, $min(flag, flag')$)

Figure 14: The extended procedure to process TELL messages with cache

CACHE&FORWARD(t : **term**; R : **set of objects**);
// It stores the pair (t, R) in the local cache and checks if related (foreign articulation) query-answer pairs can be forwarded to other peers for caching
CACHE(t, R)
if t in TO-BE-CACHED log **then** delete t from TO-BE-CACHED
for each foreign articulation $t_1 \wedge \ldots \wedge t_r \preceq u$ from another peer $\mathcal{S}$ to $self$ **do**
if $t \in \{t_1, ..., t_r\}$ and all t_1, ..., t_r are cached **then**
forward to $\mathcal{S}$ the pair $(t_1 \wedge \ldots \wedge t_r, ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}})$ for caching

Figure 15: The procedure CACHE&FORWARD

ASK_c^{alt}(PID,QID: **ID**; t : **term**; A : **set of terms**);
if t is cached **then** PID:TELL($QID, t, ans(t, S_{\mathcal{N}})$)
else $n \leftarrow 0$; QP, Q, $S \leftarrow \emptyset$
for each hyperedge $h = \langle\{u_1, ..., u_r\}, t\rangle$ **such that** $\{u_1, ..., u_r\} \cap A = \emptyset$ **do**
$C \leftarrow \emptyset$
for each u_i **do**
if u_i is cached **then**
$C \leftarrow C \cup \{ans(u_i, S_{\mathcal{N}})\}$
else
$ID \leftarrow$ NEW-QUERY-ID
$C \leftarrow C \cup \{ID\}$
$n \leftarrow n+1$
ENQUEUE(Q, (P_h, ID, u_i, $A \cup \{u_i\}$))
$QP \leftarrow QP \cup \{C\}$
if $n > 0$ **then**
PERSIST(PID,QID,t,n,QP)
until $Q \neq \emptyset$
(P_h,ID,u,B) $\leftarrow$ DEQUEUE(Q)
P_h:ASK_c^{alt}($self, ID, u, B$)
else if $QP \neq \emptyset$ **then** $S \leftarrow$ COMPUTE-ANSWER(QP)
PID:TELL($QID, S \cup I(t)$)

Figure 16: An alternative procedure to process ASK messages with cache

receives an external query q, it finds the local terms that are heads of articulation hyperedges and are needed for the evaluation of the query. Then, for each such term t, if t is not cached, it calls the $\textsc{Query}(t)$ procedure (Figure 7) and it caches t along with the received answer R, as it is certain that $R = ans(t, S_{\mathcal{N}})$. This will fill the needed caches. The answer of the original query is then computed locally (*e.g.*by a version of the $\textsc{Qe}$ procedure, modified with caching). Note that $\textsc{Query}(t)$, in this case, should call $\textsc{Ask}_c^{alt}$ (Figure 16) which is a simplified version of $\textsc{Ask}_c$ that issues $\textsc{Ask}_c^{alt}$ and $\textsc{Tell}$ messages. Though this approach has the extra cost of requiring full answers for terms that do not belong to the original query q, it is the most beneficial with respect to the trade-off cache size versus speed.

Of course, another alternative is if the above mentioned top algorithm asks for the answers of foreign terms t (through $\textsc{Query}(t)$) that appear in the body of articulation hyperedges, instead of asking for the answers of (local) terms t that are heads of articulation hyperedges.

5.1.4 Synopsis

Above we described three caching policies. Overall, four query evaluation modes can be supported by our model. The three caching policies result in faster query evaluation, but possibly not very updated results, since taxonomies, interpretations and articulations change. The mode without cache results in fresher results but with a slower query evaluation.

In case there are memory limitations for caches, various update policies could be employed, *e.g.* keep in cache only the answers of the most frequently used terms, or keep in cache only some parts of the answers, for instance "popular" objects according to some external information collected for this purpose (object-ranking techniques similar to page-ranking techniques for the Web could be employed to this end).

5.2 Querying for object descriptions

The query language of our model is term-centered, in the sense that users can extract information from a source only by asking (Boolean combinations of) terms. But sometimes it would be useful for the user to better understand the contents of an object, or the meaning or usage of terms. In these cases, a user would like to be able to ask "what are the terms that are used for describing this object?" This question can be modulated in different ways, depending whether or not only local terms are desired, and whether or not only most specific terms are desired. Correspondingly, an enhanced query language would offer 4 types of queries, for a given object o :

- the most specific, local terms describing o; assuming the local source is $S = (T, \preceq, Obj, I)$, the semantics of this query would be $ind_S(o)$;
- the local terms describing o, that is $\{t \in T \mid o \in ans(t, S)\}$;
- the most specific terms describing o in the network; assuming $\mathcal{N}$ is the network, this query would return $\bigcup\{ind_{\mathcal{S}_i}(o) \mid \mathcal{S}_i \in \mathcal{N}\}$;
- the terms describing o in the network, that is $\bigcup\{t \in T_{\mathcal{N}} \mid o \in ans(t, S_{\mathcal{N}})\}$.

The last two queries clearly make sense only if the objects are shared amongst the peers, otherwise their results would be the same as that of the previous two, respectively.

Assuming the peers are willing to share their interpretation, an efficient way of answering queries of these kinds would be to "invert the network", that is to assign each object o to one peer[7]. The designated peer can store all terms that have been assigned to o by any peer of the network, *i.e.* $ind_{S_{\mathcal{N}}}(o)$. Interestingly, much work on P2P systems has focused on the design of data structures for solving this kind of problems (see Section 6). The existence of a Distributed Hash Table (DHT) as an additional data structure (considering Obj as the set of keys) would allow checking whether $t \in ind_{S_{\mathcal{N}}}(o)$ for any t and o very efficiently, by exchanging $O(logK)$ messages where $K \simeq n$.

[7] as opposed to assign each term t to one peer.

5.3 Supporting tacit name-based articulations

In a complementary way to the network inversion discussed in the previous subsection, suppose that each element of $T_{\mathcal{N}}$ has a unique global identity and meaning, *i.e.* if the taxonomies of two peers $\mathcal{S}_1$ and $\mathcal{S}_2$ contain two terms having the same name, say $\texttt{train}_1$ and $\texttt{train}_2$, then these two correspond to the same "concept" `train`. Making the above assumption means that $T_{\mathcal{N}}$ exists before the formation of the network and that $T_{\mathcal{N}}$ comprises elements that have the same meaning for all sources that will form the network[8], *e.g.* $T_{\mathcal{N}}$ could be the set of all Greek words, or all terms of the CACM taxonomy. Note that structured P2P systems (like Chord and CAN) are based on this assumption (*i.e.* that there is a globally accepted set of keys). In contrast, our model considers that if the same term (e.g. word) appears in the taxonomies of two different peers, then these occurrences do not denote the same concept; for example $\texttt{train}_1$ could mean "wagon train", while $\texttt{train}_2$ could mean "instruct". So in our model all agreements should be represented explicitly in articulations.

However, we could extend our model so that to be able to also capture a preexisting globally accepted terminology $T_{\mathcal{N}}$, as follows: If a term t appears in two peers $\mathcal{S}_1$ and $\mathcal{S}_2$, then we could assume that $\mathcal{S}_1$ has in its articulation the relationship $t_2 \preceq t_1$, and that $\mathcal{S}_2$ has in its articulation the relationship $t_1 \preceq t_2$. Note that this would result in symmetric articulations, *i.e.* it is like assuming that we have one two-way articulation $t_1 \sim t_2$ (that is known by both $\mathcal{S}_1$ and $\mathcal{S}_2$). Although we could capture in this way the existence of a globally accepted terminology $T_{\mathcal{N}}$, in practice the definition of articulations would be problematic: how could a peer discover that another peer uses the same term?

This problem could be solved by employing a DHT that stores the terms and the addresses of the peers that use these terms. Specifically, for each term t in $T_{\mathcal{N}}$ there will be one peer that stores the addresses of all peers that have t in their taxonomies. It follows, that a peer can exploit the DHT in order to get efficiently the implicit (term-to-term) articulations of its terms (without having to discover by itself the online peers that happen to use terms that it uses too).

Specifically, if t is a term of a peer P and t is involved in the query evaluation procedure (that takes place in P), then P should ask the DHT in order to get that addresses of the peers that also use t. It follows that the calls to the DHT should be issued in the context of the Ask procedure, so as the resulting terms to be taken into account as articulation hyperedges. For example, if $\{1, 3, 5\}$ is the set of addresses returned by the DHT, then the peer behaves as if its articulation contained the relationships $t_1 \preceq t$, $t_3 \preceq t$, $t_5 \preceq t$.

Also note that a special prefix could be used for discriminating global terms from non global terms, e.g. `global` : `train`. This could be extended to support several name spaces (e.g. `transportation` : `train`, `education` : `train`).

Overall, we can exploit a DHT of this kind in order to support efficient query evaluation in cases where both implicitly defined articulations (e.g. name-based) and explicitly defined articulations (like those discussed in this paper) are desired.

6 Related work

In this paper we studied the problem of evaluating content-based retrieval queries in an entirely pure P2P architecture (without any form of structuring), where each peer can have its own conceptual model expressed as a taxonomy.

To evaluate a query q posed to a peer $\mathcal{S}$, peer $\mathcal{S}$ propagates the incoming query (which is always expressed over its own taxonomy) only to those peers to which $\mathcal{S}$ has an articulation and who can contribute to the answer of the query (the latter is determined by the taxonomy and the articulations of $\mathcal{S}$). Specifically, $\mathcal{S}$ does not propagate the original query q, but a set of queries each one expressed in the query language (here vocabulary) of the recipient peer. Note that there is not any form of centralized index (like in Napster [3]), nor any flooding of queries (like in Gnutella [1]), nor any form of partitioned global index (like in Chord [36]

[8] In other words, it is assumed that there is already a set of agreements between all peers on a common vocabulary. These agreements are not represented explicitly within the network (they are external).

and CAN [33]). Instead we have a query propagation mechanism that is query and articulation dependent (note that Semantic Overlay Networks [13] is a very simplistic approach to this). In case the objects of the domain happen to have a unique global identity (like URI), then automatic techniques can be applied for the construction of articulations (e.g. see [38]), and we can also obtain more rich object descriptions by aggregating the descriptions that have been associated to each object.

Moreover note that the peers of our model are quite autonomous in the sense that they do not have to share or publish their stored objects, taxonomies or mappings with the rest of the peers (neither to one central server, nor to the on-line peers). To participate in the network, a peer just has to answer the incoming queries by using its local base, and to propagate queries to those peers that according to its "knowledge" (i.e. taxonomy + articulations) may contribute to the evaluation of the query. However both of the above tasks are optional and at the "will" of the peer.

The literature about information integration distinguishes two main approaches: the *local-as-view* (LAV) and the *global-as-view* (GAV) approach (see [10, 28] for a comparison). In the LAV approach the contents of the sources are defined as views over the mediator's schema, while in the GAV approach the mediator's virtual contents are defined as views of the contents of the sources. The former approach offers flexibility in representing the contents of the sources, but query answering is "hard" because this requires answering queries using views ([16, 26, 42]). On the other hand, the GAV approach offers easy query answering (expansion of queries until getting to source relations), but the addition/deletion of a source implies updating the mediator view, i.e. the definition of the mediator relations. In our case, and if the articulations contain relationships between single terms, then we have the benefits of both GAV and LAV approaches, *i.e.* (a) the query processing simplicity of the GAV approach, as query processing basically reduces to unfolding the query using the definitions specified in the mapping, so as to translate the query in terms of accesses (*i.e.* queries) to the sources, and (b) the modeling scalability of the LAV approach, *i.e.* the addition of a new underlying source does not require changing the previous mappings. On the other hand, term-to-query articulations resemble the GAV approach. In a P2P setting, the cycles create more complex emergent relationships. For example suppose a peer A having an articulation $b_1 \wedge b_2 \leq a_1$ to a peer B (this is a GAV definition for a_1 of A) and a peer B having an articulation $a_1 \wedge a_2 \leq b_3$ to the peer A (this is a GAV definition for b_3 of B). However by taking into account the entire network, we result in the "mixed" relationship $b_1 \wedge b_2 \wedge a_2 \leq b_3$.

Recently, there have been several works on P2P systems endowed with logic-based models of the peers' information bases and of the mappings relating them (called *P2P mappings*). These works can be classified in 2 broad categories: (1) those assuming propositional or Horn clauses as representation language or as a computational framework, and (2) those based on more powerful formalisms. With respect to the former category (*e.g.*, see [6]), our work makes an important contribution, by providing a much simpler algorithm for performing query answering than those based on resolution. Indeed, we do rely on the theory of propositional Horn clauses, but only for proving the correctness of our algorithm. For implementing query evaluation, we devise an algorithm that avoids the (unnecessary) algorithmic complications that plague the methods based on resolution. As an example, after appropriate transformations our framework can be seen as a special case of that in [6]. Then, query evaluation can be performed by first computing the prime implicates of the negation of each term in the query, using the resolution-based algorithms presented in [6]. As the complexity of this problem is exponential w.r.t the size of the taxonomy and polynomial w.r.t. the size of Obj, there is no computational gain in using this approach. Instead, there is an algorithmic loss, since the method is much more complicated than ours.

As for the second category above, works in this area have focused on providing highly expressive knowledge representation languages in order to capture the widest range of applications. Notably, [11] proposes a model allowing, among other things, for existential quantification both in the bodies and in the heads of the mapping rules. Inevitably, such languages pose computational problems: deciding membership of a tuple in the answer of a query is undecidable in the framework proposed by [11], while disjunction in the rules' heads makes the same problem coNP-hard already for datalog with unary predicate (*i.e.* terms), as we have proved in Section 3.5. These problems are circumvented in both approaches by changing the semantics of a P2P network, in particular by adopting an epistemic reading of mappings.

Below, we review in more detail several works dealing with the problem of answering (union of) conjunctive queries posed to a peer in logic-based P2P frameworks.

In [9], a query answering algorithm for simple P2P systems is presented where each peer $\mathcal{S}$ is associated with a local database, an (exported) peer schema, and a set of local mapping rules from the schema of the local database to the peer schema. P2P mapping rules are of the form $cq_1 \leadsto cq_2$, where cq_1, cq_2 are conjunctive queries of the same arity $n \geq 1$ (possibly involving existential variables), expressed over the union of the schemas of the peers, and over the schema of a single peer, respectively[9]. Note that this representation framework partially subsumes our network source framework, since in our case cq_1, cq_2 are of arity 1, cq_1 is a conjunctive query of the form $u_1(x) \wedge ... \wedge u_r(x)$ over the terminology of a single peer[10] and q_2 is a single atom query $t(x)$ over the terminology of the peer that the mapping (articulation) belongs to. However, simple P2P systems cannot express the local to a peer $\mathcal{S}$ taxonomy $\preceq_{\mathcal{S}}$ of our framework. Query answering in simple P2P systems according to the first-order logic (FOL) semantics is in general undecidable. Therefore, the authors adopt a new semantics based on epistemic logic in order to get decidability for query answering. Notably, the FOL semantics and epistemic logic semantics for our framework coincide. In particular, in [9], a centralized bottom-up algorithm is presented which essentially constructs a finite database RDB which constitutes a "representative" of all the epistemic models of the P2P system. The answers to a conjunctive query q are the answers of q w.r.t. RDB. However, though this algorithm has polynomial time complexity, it is centralized and it suffers from the drawbacks of bottom-up computation that does not take into account the structure of the query.

The work in [9] is extended in [11], where a more general framework for P2P systems is considered, which fully subsumes our framework and whose semantics is based on epistemic logic. In particular, in [11], a peer is also associated with a set of (function-free) FOL formulas over the schema of the peer. A top-down distributed query answering algorithm is presented which is based on synchronous messaging. Essentially, the algorithm returns to the peer where the original query is posed, a datalog program by transferring the full extensions of the relevant to the query, peer source predicates along the paths of peers involved in query processing. The returned datalog program is used for providing the answers to the query. Obviously, our algorithm has computational advantages w.r.t. the algorithm in [11], since during query evaluation only the full or partial answer to a term (sub)query is transfered to the peer that posed the (sub)query, and not the full extensions of all terms involved in its evaluation.

The framework in [34], extends our framework by considering (i) n-ary (instead of unary) predicates (*i.e.* P2P mappings are general datalog rules) and (ii) a set of domain relations (also suggested in [35]), mapping the objects of one peer to the objects of another peer. A distributed query answering algorithm is presented based on synchronous messaging. However, the algorithm will perform poorly in our restricted framework[11], since when a peer receives a (sub)query, it iterates through the relevant P2P mappings and for each one of them, sends a (sub)query to the appropriate peer (waiting for its answer), until fixpoint is reached. In our case, when a peer receives a (sub)query, each relevant P2P mapping is considered just once and no iteration until fixpoint is required.

A P2P framework similar to [9] is presented in [25], where query answering according to FOL semantics is investigated. Since in general, query answering is undecidable, the authors present a centralized algorithm (employed in the Piazza system [23]), which however is complete (the algorithm is always sound), only for the case that polynomial time complexity in query answering can be achieved. This includes the condition that inclusion P2P mappings are acyclic. However, such a condition severely restricts the modularity of the system. Note that our algorithm is sound and complete even in the case that there are cycles in the term dependency path and it always terminates. Thus, our framework allows placing articulations between peers without further checks. This is quite important, because the actual interconnections are not under the control of any actor in the system.

In [20, 19], the authors consider a framework where each peer is associated with a relational database, and P2P mapping rules contain conjunctive queries in both the head and the body of the rule (possibly with existential variables), each expressed over the alphabet of a single peer. Again the semantics of the system is defined based on epistemic logic [18]. In these papers, a peer database update algorithm is provided allowing for subsequent peer queries to be answered locally without fetching data from other nodes at query

[9]Note that P2P mapping rules of this kind can accommodate both GAV and LAV-style mappings, and are referred in the literature as GLAV mappings.

[10]Recall that this restriction can be easily relaxed.

[11]In our framework, domain relations correspond to the identity relation.

time. The algorithm (which is based on asynchronous messaging) starts at the peer which sends queries to all neighbour peers according to the involved mapping rules. When a peer receives a query, the query is processed locally by the peer itself using its own data. This first answer is immediately replied back to the node which issued the query and sub-queries are propagated similarly to all neighbour peers. When a peer receives an answer, (i) it stores the answer locally, (ii) it materializes the view represented in the head on the involved mapping rule, and (ii) it propagates the result to the peer that issued the (sub)query. Answer propagation stops when no new answer tuples are coming to the peer through any dependency path, that is until fixpoint is reached. In our case, the database update problem for a peer $\mathcal{S}$ amounts to invoking $\mathcal{S}'$: $\textsc{Query}(q)$ for each articulation $q \preceq t$ from $\mathcal{S}$ to another peer $\mathcal{S}'$ and storing the answer locally to $\mathcal{S}$. Note that our query answering algorithm is also based on asynchronous messaging. However, since it considers a limited framework, it is much simpler and no computation until fixpoint is required. In particular, for each term (sub)query issued to a peer through Ask, only one answer is returned through Tell.

7 Conclusions

This study presents a model of a P2P network consisting of sources based on taxonomies. A taxonomy states subsumption relationships between negation-free DNF formulas on terms and negation-free conjunctions of terms. The language for querying such sources offers Boolean combinations of terms, in which negation can be efficiently handled by adopting a closed-world reading of the information. An efficient, hypergraph-based query evaluation method is presented for such sources, resting on results coming from the theory of propositional clauses. It is also shown that extending the expressive power of the taxonomy language by adding negation or full disjunction, leads to the intractability of the decision problem.

A model of a P2P network, having sources as nodes, is subsequently presented. The essential feature of the model is the possibility of relating the assumed disjoint peer terminologies by means of subsumption relationships of the same type as those in the taxonomies of the sources. The resulting system subscribes to the universally accepted notion of P2P information system, recently postulated also in the context of the so-called emergent semantics [4]. It is also shown that the results presented in the paper do apply also if the subsumption relationships are formed by arbitrarily mixing terms from different terminologies.

An efficient query evaluation procedure for queries stated against such a network is presented, and proved correct. The procedure is a distributed version of the centralized procedure, based on an asynchronous, message-based interaction amongst the peers aimed at favoring scalability. Some optimization techniques are also discussed, namely one based on caching, for which the algorithms for message processing are given.

Finally, the work is related to the most relevant papers in the area of P2P systems. It remains to be seen, whether the same efficiency can be obtained by allowing full datalog as a representation language for information sources and for articulations. Yet, it is evident that the B-graph based algorithm presented in this paper does not extend immediately to the general datalog case, due to the presence of multiple variables in the rules and unification.

Acknowledgments

We thank Nicolas Spyratos for inspiring this work. We also thank ERCIM for offering Yannis Tzitzikas a fellowship in the course of which the work was started, and the DELOS Network of Excellence Exchange Programme, for supporting Carlo Meghini's visit to the Institute of Computer Science of FORTH, in the course of which the work was completed.

References

[1] Gnutella(http://gnutella.wego.com).

[2] Kazaa (http://www.kazaa.com/).

[3] Napster (www.naptster.com), 2001.

[4] K. Aberer, T. Catarci, P. Cudré-Mauroux, T. S. Dillon, S. Grimm, M. Hacid, A. Illarramendi, M. Jarrar, V. Kashyap, M. Mecella, E. Mena, E. J. Neuhold, A. M. Ouksel, T. Risse, M. Scannapieco, F. Saltor, L. De Santis, S. Spaccapietra, S. Staab, R. Studer, and O. De Troyer. "Emergent Semantics Systems". In *Procs. of the 1st Intern. IFIP Conference on Semantics of a Networked World (ICSNW 2004)*, pages 14–43, 2004.

[5] S. Abitebul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN: 0-201-53771-0.

[6] Philippe Adjiman, Philippe Chatalic, Francois Goasdoué, Marie-Christine Rousset, and Laurent Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *Journal of Artificial Intelligence Research*, 25:269–314, 2006.

[7] Philip A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. "Data Management for Peer-to-Peer Computing: A Vision". In *Proceedings of WebDB02*, Madison, Wisconsin, June 2002.

[8] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. "Feasibility of a Serveless Distributed File System Deployed on an Existing Set of Desktop PCs". In *Proceedings of Measurement and Modeling of Computer Systems*, June 2000.

[9] Diego Calvanese, Elio Damaggio, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. "Semantic Data Integration in P2P Systems". In *Procs. of the First International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2003)*, pages 79–90, 2003.

[10] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. "A Framework for Ontology Integration". In *Proc. of the 2001 Int. Semantic Web Working Symposium (SWWS 2001)*, pages 303–316, Stanford University, California, USA, July 30 - August 1 2001.

[11] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. "Logical foundations of peer-to-peer data integration". In *Procs. of the 23rd ACM symposium on Principles of database systems, PODS'2004*, pages 241–251, New York, NY, USA, 2004. ACM Press.

[12] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer Verlag, 1990.

[13] Arturo Crespo and Hector Garcia-Molina. "Semantic Overlay Networks for P2P Systems". Technical report, Computer Science Department, Stanford University, October 2002.

[14] Francisco Matias Cuenca-Acuna, Christopher Peery, Richard P. Martin, and Thu D. Nguyen. "PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities". In *Procs. of 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, pages 236–249. IEEE Press, June 2003.

[15] E. Dantsin, T. H. Eiter, G. Gottlob, and A. Voronkov. "Complexity and Expressive Power of Logic Programming". *ACM Computing Survey*, ACM Computing Survey 33(3):374–425, September 2001.

[16] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Procs. of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97)*, pages 109–116, Tucson, Arizona, 12-14 May 1997.

[17] P.A. Fejer and D.A. Simovici. *Mathematical Foundations of Computer Science. Volume 1: Sets, Relations, and Induction*. Springer-Verlag, 1991.

[18] E. Franconi, G. M. Kuper, A. Lopatenko, and L. Serafini. "A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems". In *Procs. of the First International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P 2003)*, pages 64–76, 2003.

[19] E. Franconi, G. M. Kuper, A. Lopatenko, and I. Zaihrayeu. "A Distributed Algorithm for Robust Data Sharing and Updates in P2P Database Networks". In *Procs. of the EDBT'04 Intern. Workshop on Peer-to-Peer Computing and Databases (P2P&DB 2004)*, pages 446–455, 2004.

[20] E. Franconi, G. M. Kuper, A. Lopatenko, and I. Zaihrayeu. "Queries and Updates in the coDB Peer to Peer Database System". In *Procs. of the 30th International Conference on Very Large Data Bases (VLDB 2004)*, pages 1277–1280, 2004.

[21] Giorgio Gallo, Giustino Longo, and Stefano Pallottino. "Directed Hypergraphs and Applications". *Discrete Applied Mathematics*, 42(2):177–201, 1993.

[22] Bernhard Ganter and Rudolf Wille. *"Formal Concept Analysis: Mathematical Foundations"*. Springer-Verlag, Heidelberg, 1999.

[23] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. "The Piazza Peer Data Management System". *IEEE Transactions on Knowledge and Data Engineering*, 16(7):787–798, 2004.

[24] Alon Halevy, Zachary Ives, Peter Mork, and Igor Tatarinov. "Piazza: Data Management Infrastructure for Semantic Web Applications". In *Procs. of the 12th International Conference on World Wide Web (WWW 2003)*, pages 556 – 567, May 2003.

[25] Alon Halevy, Zachary Ives, Dan Suciu, and Igor Tatarinov. "Schema Mediation in Peer Data Management Systems". In *Procs. of the 19th International Conference on Data Engineering (ICDE'03)*, pages 505–518, March 2003.

[26] Alon Y. Halevy. "Answering Queries Using Views: A Survey". *VLDB Journal*, 10(4):270–294, 2001.

[27] M. Koubarakis and C. Tryfonopoulos. "Peer-to-Peer Agent Systems for Textual Information Dissemination: Algorithms and Complexity". In *Proceedings of the UK Workshop on Multiagent Systems,UKMAS'02*, Liverpool, UK, 2002.

[28] Maurizio Lenzerini. "Data Integration: A Theoretical Perspective". In *Proc. of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2002)*, pages 233–246, Madison, Wisconsin, USA, June 2002.

[29] Bo Ling, Zhiguo Lu, Wee Siong Ng, BengChin Ooi, Kian-Lee Tan, and Aoying Zhou. "A Content-Based Resource Location Mechanism in PeerIS". In *Proceedings of the 3rd International Conference on Web Information Systems Engineering, WISE 2002*, Singapore, December 2002.

[30] Carlo Meghini and Yannis Tzitzikas. "Querying Articulated Sources". In *Procs. of the 3rd Intern. Conference on Ontologies, Databases and Applications of Semantics for Large Scale Information Systems, ODBASE'2004*, pages 945–962, Larnaca, Cyprus, October 2004.

[31] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. "EDUTELLA: A P2P networking infrastructure based on RDF". In *Procs. of the 11th International Conference on World Wide Web (WWW'02)*, pages 604 – 615, 2002.

[32] W. Nejdl, B. Wolf, S. Staab, and J. Tane. "EDUTELLA: Searching and Annotating Resources within an RDF-based P2P Network". In *Procs of the WWW2002 International Workshop on the Semantic Web*, Honolulu, Havaii, May 2002.

[33] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. "A Scalable Content Addressable Network". In *Procs. of 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'2001)*, pages 161 – 172, 2001.

[34] L. Serafini and C. Ghidini. "Using Wrapper Agents to Answer Queries in Distributed Information Systems". In *Procs. of the First International Conference on Advances in Information Systems (ADVIS '00)*, pages 331–340. Springer-Verlag, 2000.

[35] L. Serafini, F. Giunchiglia, J. Mylopoulos, and P. A. Bernstein. "Local Relational Model: A Logical Formalization of Database Coordination". In *Procs. of the 4th International and Interdisciplinary Conference on Modeling and Using Context(CONTEXT 2003)*, pages 286–299, 2003.

[36] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001.

[37] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam. "pSearch: Information Retrieval in Structured Overlays". In *Procs. of ACM HotNets-I*, October 2002.

[38] Yannis Tzitzikas and Carlo Meghini. "Ostensive Automatic Schema Mapping for Taxonomy-based Peer-to-Peer Systems". In *Seventh International Workshop on Cooperative Information Agents, CIA-2003*, pages 78–92, Helsinki, Finland, August 2003. (Best Paper Award).

[39] Yannis Tzitzikas and Carlo Meghini. "Query Evaluation in Peer-to-Peer Networks of Taxonomy-based Sources". In *Procs. of 19th Int. Conf. on Cooperative Information Systems, CoopIS'2003*, pages 263–281, Catania, Sicily, Italy, November 2003.

[40] Yannis Tzitzikas, Carlo Meghini, and Nicolas Spyratos. "Taxonomy-based Conceptual Modeling for Peer-to-Peer Networks". In *Procs. of 22th Int. Conf. on Conceptual Modeling, ER'2003*, pages 446–460, Chicago, Illinois, October 2003.

[41] Jeffrey D. Ullman. *"Principles of Database and Knowledge-Base Systems, Vol. I"*. Computer Science Press, 1988.

[42] Jeffrey D. Ullman. "Information integration using logical views". In *Procs. of the 6th Int. Conf. on Database Theory (ICDT-97)*, pages 19–40, Delphi, Greece, 8-10 January 1997.

A Completion of the example

We resume the example from the processing of the message P_a:$\textsc{Tell}(q3, I(a3))$.

- P_a:$\textsc{Tell}(q3, I(a3))$
  $\textsc{Tell}$ finds the object in the log and updates it. The old log on P_a was:

P_a log
$(P_a, q1, t, 2, \{\{q2, q3\}\})$

  The new log is:

P_a log
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$

- P_a:$\textsc{Ask}(P_a, q2, a2, \{t, a2\})$
  Since there are two incoming hyperedges in $a2$, both in P_b, $\textsc{Ask}$ enqueues 3 $\textsc{Ask}$ messages to P_b, one for each involved term:
  - P_b:$\textsc{Ask}(P_a, q4, b3, \{t, a2, b3\})$
  - P_b:$\textsc{Ask}(P_a, q5, b1, \{t, a2, b1\})$
  - P_b:$\textsc{Ask}(P_a, q6, b2, \{t, a2, b2\})$

  It then persists the corresponding log object. The new log is:

P_a log
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$
$(P_a, q2, a2, 3, \{\{q4\}, \{q5, q6\}\})$

  and issues the 3 enqueued messages.
- P_b:$\textsc{Ask}(P_a, q4, b3, \{t, a2, b3\})$
  Since there are no incoming hyperedges in $b3$, the message P_a:$\textsc{Tell}(q4, I(b3))$ is produced.
- P_a:$\textsc{Tell}(q4, I(b3))$
  $\textsc{Tell}$ finds the object in the log and updates it. The updated log is:

P_a
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$
$(P_a, q2, a2, 2, \{\{I(b3)\}, \{q5, q6\}\})$

- P_b:$\textsc{Ask}(P_a, q5, b1, \{t, a2, b1\})$
  Since there are two incoming hyperedges in $b1$, $\textsc{Ask}$ enqueues 2 $\textsc{Ask}$ messages to P_c, one for each involved term:
  - P_c:$\textsc{Ask}(P_b, q7, c1, \{t, a2, b1, c1\})$
  - P_c:$\textsc{Ask}(P_b, q8, c2, \{t, a2, b1, c2\})$

  It then persists the corresponding log object. The log is now:

P_b log
$(P_a, q5, b1, 2, \{\{q7\}, \{q8\}\})$

- P_b:$\textsc{Ask}(P_a, q6, b2, \{t, a2, b2\})$
  Since there is one incoming hyperedge in $b2$, $\textsc{Ask}$ enqueues 2 $\textsc{Ask}$ messages to P_c, one for each involved term:
  - P_c:$\textsc{Ask}(P_b, q9, c2, \{t, a2, b2, c2\})$
  - P_c:$\textsc{Ask}(P_b, q10, c3, \{t, a2, b2, c3\})$

  It then persists the corresponding log object. The log is now:

P_b log
$(P_a, q5, b1, 2, \{\{q7\}, \{q8\}\})$
$(P_a, q6, b2, 2, \{\{q9, q10\}\})$

- P_c:$\textsc{Ask}(P_b, q7, c1, \{t, a2, b1, c1\})$
  Since there are no incoming hyperedges in $c1$, $\textsc{Ask}$ generates P_b:$\textsc{Tell}(q7, I(c1))$.
- P_b:$\textsc{Tell}(q7, I(c1))$
  $\textsc{Tell}$ finds the object in the log and updates it. The new log is:

P_b log
$(P_a, q5, b1, 1, \{\{I(c1)\}, \{q8\}\})$
$(P_a, q6, b2, 2, \{\{q9, q10\}\})$

- P_c:Ask$(P_b, q8, c2, \{t, a2, b1, c2\})$
  Since there is one incoming hyperedge in $c2$ but its tail has a non-empty intersection with the set of visited terms, just a Tell message results: P_b:Tell$(q8, I(c2))$.
- P_b:Tell$(q8, I(c2))$
  Tell finds the object in the log and updates it. The new log is:

P_b log
$(P_a, q5, b1, 0, \{\{I(c1)\}, \{I(c2)\}\})$
$(P_a, q6, b2, 2, \{\{q9, q10\}\})$

  There are no more open calls in the updated log object, therefore the answer to the query $q5$ can be computed as $I(c1) \cup I(c2)$. Then the object is deleted permanently from the log and the message P_a:Tell$(q5, I(b1) \cup I(c1) \cup I(c2))$ is issued.
- P_a:Tell$(q5, I(b1) \cup I(c1) \cup I(c2))$
  Tell finds the object in the log and updates it. The new log is:

P_a log
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$
$(P_a, q2, a2, 1, \{\{I(b3)\}, \{I(b1) \cup I(c1) \cup I(c2), q6\}\})$

- P_c:Ask$(P_b, q9, c2, \{t, a2, b2, c2\})$
  Since there is one incoming hyperedge in $c2$, Ask enqueues 2 Ask messages to P_b, one for each involved term:
  - P_b:Ask$(P_c, q11, b1, \{t, a2, b2, c2, b1\})$
  - P_b:Ask$(P_c, q12, b3, \{t, a2, b2, c2, b3\})$

  It then persists the corresponding log object. The updated log is:

P_c log
$(P_b, q9, c2, 2, \{\{q11, q12\}\})$

- P_c:Ask$(P_b, q10, c3, \{t, a2, b2, c3\})$
  Since there are no incoming hyperedges in $c3$ a Tell message results: P_b:Tell$(q10, I(c3))$.
- P_b:Tell$(q10, I(c3))$
  Tell finds the object in the log and updates it. The updated log is:

P_b log
$(P_a, q6, b2, 1, \{\{q9, I(c3)\}\})$

- P_b:Ask$(P_c, q11, b1, \{t, a2, b2, c2, b1\})$
  There are two incoming hyperedges in $b1$, but the one having $c2$ in the tail generates no Ask messages. The only Ask enqueued is therefore:
  - P_c:Ask$(P_b, q13, c1, \{t, a2, b2, c2, b1, c1\})$

  It then persists the corresponding log object. The updated log is:

P_b log
$(P_a, q6, b2, 1, \{\{q9, I(c3)\}\})$
$(P_c, q11, b1, 1, \{\{q13\}\})$

- P_b:Ask$(P_c, q12, b3, \{t, a2, b2, c2, b3\})$
  Since there are no incoming hyperedges in $b3$, it results: P_c:Tell$(q12, I(b3))$.
- P_c:Tell$(q12, I(b3))$
  Tell finds the object in the log and updates it. The new log is:

P_c log
$(P_b, q9, c2, 1, \{\{q11, I(b3)\}\})$

- P_c:Ask$(P_b, q13, c1, \{t, a2, b2, c2, b1, c1\})$
  Since there are no incoming hyperedges in $c1$, Ask issues P_b:Tell$(q13, I(c1))$.
- P_b:Tell$(q13, I(c1))$
  Tell finds the object in the log and updates it. The new log is:

P_b log
$(P_a, q6, b2, 1, \{\{q9, I(c3)\}\})$
$(P_c, q11, b1, 0, \{\{I(c1)\}\})$

  There are no more open calls in the updated log object, therefore the answer to the query $q11$ can be computed as $I(c1)$. Then the object is permanently deleted from the log and the message P_c:Tell$(q11, I(b1) \cup I(c1))$ is issued.

– P_c:TELL$(q11, I(b1) \cup I(c1))$

TELL finds the object in the log and updates it. The updated log is:

P_c log
$(P_b, q9, c2, 0, \{\{I(b1) \cup I(c1), I(b3)\}\})$

There are no more open calls in the updated log object, therefore the answer to the query $q9$ can be computed. Then the object is permanently deleted from the log and the message P_b:TELL$(q9, [(I(b1) \cup I(c1)) \cap I(b3)] \cup I(c2))$ is issued.

– P_b:TELL$(q9, [(I(b1) \cup I(c1)) \cap I(b3)] \cup I(c2))$

TELL finds the object in the log and updates it. The updated log is:

P_b log
$(P_a, q6, b2, 0, \{\{[(I(b1) \cup I(c1)) \cap I(b3)] \cup I(c2), I(c3)\}\})$

There are no more open calls in the updated log object, therefore the answer to the query $q6$ can be computed. Then the object is permanently deleted from the log and the message P_a:TELL$(q6, [X \cap I(c3)] \cup I(b2))$ is issued, where

$$X = [(I(b1) \cup I(c1)) \cap I(b3)] \cup I(c2)$$

– P_a:TELL$(q6, [X \cap I(c3)] \cup I(b2))$

TELL finds the object in the log and updates it. The updated log is:

P_a log
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$
$(P_a, q2, a2, 0, \{\{I(b3)\}, \{I(b1) \cup I(c1) \cup I(c2), [X \cap I(c3)] \cup I(b2)\}\})$

There are no more open calls in the updated log object, therefore the answer to the query $q2$ can be computed. Then the object is permanently deleted from the log and the message P_a:TELL$(q2, I(a2) \cup I(b3) \cup (Y \cap Z)$ is issued, where

$$\begin{array}{rcl} Y & = & I(b1) \cup I(c1) \cup I(c2) \\ Z & = & [X \cap I(c3)] \cup I(b2) \end{array}$$

– P_a:TELL$(q2, I(a2) \cup I(b3) \cup (Y \cap Z))$

TELL finds the object in the log and updates it. The new log is:

P_a log
$(P_a, q1, t, 0, \{\{I(a3) \cap (I(a2) \cup I(b3) \cup (Y \cap Z))\}\})$

There are no more open calls in the updated object and $q1 \notin T_{P_a}$. Therefore, $q1$ must be a user (external) query.

The QUERY procedure will realize that $q1$ is complete, and return the answer to the user, thus concluding query evaluation.