Distributed Reasoning in a Peer-to-Peer Setting

P. Adjiman and **P. Chatalic** and **F. Goasdoué** and **M.-C. Rousset** and **L. Simon**[1]

1 Introduction

In a peer-to-peer system, there is no centralized control or hierarchical organization: each peer is equivalent in functionality and cooperates with other peers in order to solve a collective task. Such systems have evolved from simple keyword-based peer-to-peer file sharing systems like Napster and Gnutella to schema-based peer data management systems like Edutella [3] or Piazza [2], which handle semantic data description and support complex queries for data retrieval.

In this paper, we are interested in peer-to-peer inference systems in which each peer can answer queries by reasoning from its local (propositional) theory but can also ask queries to some other peers with which it is semantically related by sharing part of its vocabulary. This framework encompasses several applications like peer-to-peer information integration systems or intelligent agents, in which each peer has its own knowledge (about its data or its expertise domain) and some partial knowledge about some other peers. In this setting, when it is solicited to perform a reasoning task and if it cannot solve completely that task locally, a peer must be able to distribute appropriate reasoning subtasks among its acquainted peers.

The contribution of this paper is the first consequence finding algorithm in a peer-to-peer setting: it is anytime and computes consequences gradually from the solicited peer to peers that are more and more distant. We have exhibited a sufficient condition on the acquaintance graph of the peer-to-peer inference system for guaranteeing the completeness of this algorithm. Our algorithm splits clauses if they involve vocabularies of several peers. Each piece of a splitted clause is transmitted to the corresponding theory to find its consequences. The consequences that are found for each piece of splitted clause must be recomposed to get the consequences of that clause.

2 Peer-to-peer inference: problem definition

A peer-to-peer inference system (P2PIS) is a network of peer theories. Each peer P is a finite set of propositional formulas of a language $\mathcal{L}_P$. We consider the case where $\mathcal{L}_P$ is the language of clauses without duplicated literals that can be built from a finite set of propositional variables $\mathcal{V}_P$, called the *vocabulary* of P. Peers can be semantically related by having common variables in their respective vocabularies, called *shared variables*. In a P2PIS, no peer has the knowledge of the global P2PIS theory. Each peer only knows its own local theory and that it shares some variables with some other peers (its *acquaintances*). It does not necessarily know *all* the peers with which it shares variables. When a new peer joins a P2PIS it simply declares some acquaintances, i.e., the peers it knows to be sharing variables with. A P2PIS can be formalized as an *acquaintance graph*.

Definition 1 (Acquaintance graph) *Let $\mathcal{P} = (P_i)_{i=1..n}$ be a family of clausal theories on their respective vocabularies $\mathcal{V}_{P_i}$, let $\mathcal{V} = \cup_{i=1..n}\mathcal{V}_{P_i}$. An acquaintance graph is a graph $\Gamma = (\mathcal{P}, \text{ACQ})$ where $\mathcal{P}$ is the set of vertices and $\text{ACQ} \subseteq \mathcal{V}\times\mathcal{P}\times\mathcal{P}$ is a set of labelled edges such that for every $(v, P_i, P_j) \in \text{ACQ}$, $i \neq j$ and $v \in \mathcal{V}_{P_i} \cap \mathcal{V}_{P_j}$.*

(v, P_i, P_j) expresses that peers P_i and P_j know each other to be sharing the variable v. For a peer P and a literal l, $\text{ACQ}(l, P)$ denotes the set of peers sharing with P the variable of l.

For each theory P, we consider a subset of *target variables* $\mathcal{TV}_P \subseteq \mathcal{V}_P$, supposed to represent the variables of interest for the application. The goal is, given a clause (called the *query*), to find all the consequences (called *answers*) that belong to some *target language*. The point is that the query only uses the vocabulary of the queried peer, but the expected answers may involve target variables of different peers. The target languages handled by our algorithm are defined in terms of target variables and require that a shared variable has the same target status in all the peers sharing it.

Definition 2 (Target Language) *Let $\Gamma = (\mathcal{P}, \text{ACQ})$ be a P2PIS, and for very peer P, let $\mathcal{TV}_P$ be the set of its target variables such that if $(v, P_i, P_j) \in \text{ACQ}$ then $v \in \mathcal{TV}_{P_i}$ iff $v \in \mathcal{TV}_{P_j}$. For a subset SP of peers of $\mathcal{P}$, we define its* target language *$\mathcal{T}arget(SP)$ as the language of clauses (including the empty clause $\square$) involving only variables of $\bigcup_{P\in SP}\mathcal{TV}_P$.*

Among the possible answers we distinguish *local answers*, involving only target variables of the solicited peer, *navigational answers*, which involve target variables of a single peer, and *integrating answers* which involve target variables of several peers.

Definition 3 (Proper prime implicate wrt a theory) *Let P be a clausal theory and q be a clause. A clause m is said to be:*

- *a* prime implicate *of q wrt P iff $P \cup \{q\} \models m$ and for any other clause m', if $P \cup \{q\} \models m'$ and $m' \models m$ then $m' \equiv m$.*
- *a* proper prime implicate *of q wrt P iff it is a prime implicate of q wrt P but $P \not\models m$.*

Definition 4 (Consequence finding problem) *Let $\mathcal{P} = (P_i)_{i=1..n}$ be a family of clausal theories with respective target variables $(\mathcal{TV}_{P_i})_{i=1..n}$ and let $\Gamma = (\mathcal{P}, \text{ACQ})$ be a $P2PIS$. The* consequence finding problem *is, given a peer P and a clause $q \in \mathcal{L}_P$ to find the set of proper prime implicates of q wrt $\bigcup_{i=1..n} P_i$ which belong to $\mathcal{T}arget(\mathcal{P})$.*

3 Distributed consequence finding algorithm

The distributed algorithm that we have designed is a message passing algorithm implemented locally at each peer. It handles an history which is initialized to the empty sequence. An history $hist$ is a sequence of triples (l, P, c) (where l is a literal, P a peer, and c a clause). An history $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1), (l_0, P_0, c_0)]$ represents a branch of reasoning initiated by the propagation of the literal l_0 within the peer P_0, and the splitting of the clause c_0: for every $i \in [0..n-1]$, c_i is a consequence of l_i and P_i, and l_{i+1} is a literal of c_i, which is propagated in P_{i+1}.

[1] Université Paris-Sud XI – CNRS (LRI) & INRIA (Futurs), Bâtiment 490, Université Paris-Sud XI, 91405 Orsay Cedex, France

The algorithm is composed of three procedures, each one being triggered by the reception of a message.

The procedure RECEIVEQUERYMESSAGE is triggered by the reception of a $query$ message $m(Sender, Receiver, query, hist, l)$ sent by the peer $Sender$ to the peer $Receiver$ which executes the procedure: on the demand of $Sender$, with which it shares the variable of l, it processes the literal l.

The procedure RECEIVEANSWERMESSAGE is triggered by the reception of an $answer$ message $m(Sender, Receiver, answer, hist, r)$ sent by the peer $Sender$ to the peer $Receiver$ which executes the procedure: it processes the answer r (which is a clause involving target variables only) sent back by $Sender$ for the literal l (last added in the history) ; it may have to combine it with other answers for literals being in the same clause as l.

The procedure RECEIVEFINALMESSAGE is triggered by the reception of a $final$ message $m(Sender, Receiver, final, hist, true)$: the peer $Sender$ notifies the peer $Receiver$ that answer computation for the literal l (last added in the history) is completed. Those procedures handle two data structures stored at each peer: ANSWER$(l, hist)$ caches the answers resulting from the propagation of l within the reasoning branch corresponding to $hist$; FINAL$(q, hist)$ is set to true when the propagation of q within the reasoning branch of the history $hist$ is completed. The reasoning is initiated by the user (denoted by a particular peer $User$) sending to a given peer P a message $m(User, P, query, \emptyset, q)$, which triggers the procedure RECEIVEQUERYMESSAGE$(m(\ User, P, query, \emptyset, q))$ that is locally executed by P. In the description of the procedures, since they are locally executed by the peer which receives the message, we will denote by $Self$ the receiver peer.

In the following, we will use the notations:

• for a literal q, $Resolvent(q, P)$ denotes the set of clauses obtained by resolution between q and a clause of P,

• for a literal q, $\bar{q}$ denotes its complementary literal,

• for a clause c of a peer P, $S(c)$ (resp. $L(c)$) denotes the disjonction of literals of c whose variables are shared (resp. not shared),

• $\otimes$ is the distribution operator on sets of clauses: $S_1 \otimes \cdots \otimes S_n = \{c_1 \vee \cdots \vee c_n \mid c_1 \in S_1, \ldots, c_n \in S_n\}$. If $L = \{l_1, \ldots, l_p\}$, we will use the notation $\otimes_{l \in L} S_l$ to denote $S_{l_1} \otimes \cdots \otimes S_{l_p}$.

The following theorems summarize the main properties of our distributed message passing algorithm. Their full proofs are given in [1]. Theorem 2 (completeness) is only guaranteed if the following property holds on the acquaintance graph: *if two local theories have a common variable, there must exist a path between those two theories, all the edges of which are labeled with that variable*.

Theorem 1 (Soundness and termination) *If P receives from the user the message $m(User, P, query, \emptyset, q)$, then: a finite number of answer messages will be produced and each produced answer message $m(P, User, answer, [(q, P, \_)], r)$ is such that r is an implicate of q wrt $S(\mathcal{P})$ which belong to $\mathcal{T}arget(\mathcal{P})$.*

Theorem 2 (Completeness) *If each local theory is saturated by resolution and if P receives from the user the message $m(User, P, query, \emptyset, q)$, then for each proper prime implicates r of q wrt $S(\mathcal{P})$ belonging to $\mathcal{T}arget(\mathcal{P})$, an answer message $m(P, User, answer, [(q, P, \_)], r)$ will be produced.*

Theorem 3 (Notification of termination) *If r is the last result returned through an answer message $m(P, User, answer, [(q, P, \_)], r)$ then the termination will be notified to the user by a message $m(P, User, final, [(q, P, true)], true)$.*

For sake of simplicity, our algorithm applies here to literals. Clausal queries are handled by splitting them into literals and using the $\otimes$ operator to recompose the results obtained for each literal.

An experimental analysis of this algorithm is provided in [1].

Algorithm 1: Message passing procedure for processing queries
RECEIVEQUERYMESSAGE$(m(Sender, Self, query, hist, q))$
(1) **if** $(\bar{q}, \_, \_) \in hist$
(2) **send** $m(Self, Sender, answer, [(q, Self, \Box)|hist], \Box)$
(3) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(4) **else if** $q \in Self$ or $(q, Self, \_) \in hist$
(5) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(6) **else**
(7) LOCAL$(Self) \leftarrow \{q\} \cup Resolvent(q, Self)$
(8) **if** $\Box \in$ LOCAL$(Self)$
(9) **send** $m(Self, Sender, answer, [(q, Self, \Box)|hist], \Box)$
(10) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(11) **else**
(12) LOCAL$(Self) \leftarrow \{c \in$ LOCAL$(Self) \mid L(c) \in \mathcal{T}arget(Self)\}$
(13) **if** for every $c \in$ LOCAL$(Self), S(c) = \Box$
(14) **foreach** $c \in$ LOCAL$(Self)$
(15) **send** $m(Self, Sender, answer, [(q, Self, c)|hist], c)$
(16) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(17) **else**
(18) **foreach** $c \in$ LOCAL$(Self)$
(19) **if** $S(c) = \Box$
(20) **send** $m(Self, Sender, answer, [(q, Self, c)|hist], c)$
(21) **else**
(22) **foreach** literal $l \in S(c)$
(23) **if** $l \in \mathcal{T}arget(Self)$
(24) ANSWER$(l, [(q, Self, c)|hist]) \leftarrow \{l\}$
(25) **else**
(26) ANSWER$(l, [(q, Self, c)|hist]) \leftarrow \emptyset$
(27) FINAL$(l, [(q, Self, c)|hist]) \leftarrow false$
(28) **foreach** $RP \in$ ACQ$(l, Self)$
(29) **send** $m(Self, RP, query, [(q, Self, c)|hist], l)$

Algorithm 2: Message passing procedure for processing answers
RECEIVEANSWERMESSAGE$(m(Sender, Self, answer, hist, r))$
(1) $hist$ is of the form $[(l', Sender, c'), (q, Self, c)|hist']$
(2) ANSWER$(l', hist) \leftarrow$ ANSWER $(l', hist) \cup \{r\}$
(3) RESULT$\leftarrow \otimes_{l \in S(c) \setminus \{l'\}}$ ANSWER$(l, hist) \otimes \{L(c) \vee r\}$
(4) **if** $hist' = \emptyset$, $U \leftarrow User$ **else** $U \leftarrow$ the first peer P' of $hist'$
(5) **foreach** $cs \in$ RESULT
(6) **send** $m(Self, U, answer, [(q, Self, c)|hist'], cs)$

Algorithm 3: Message passing procedure for notifying termination
RECEIVEFINALMESSAGE$(m(Sender, Self, final, hist, true))$
(1) $hist$ is of the form $[(l', Sender, true), (q, Self, c)|hist']$
(2) FINAL$(l', hist) \leftarrow true$
(3) **if** for every $l \in S(c)$, FINAL$(l, hist) = true$
(4) **if** $hist' = \emptyset$ $U \leftarrow User$ **else** $U \leftarrow$ the first peer P' of $hist'$
(5) **send** $m(Self, U, final, [(q, Self, true)|hist'], true)$
(6) **foreach** $l \in S(c)$
(7) ANSWER$(l, [(l, Sender, _), (q, Self, c)|hist']) \leftarrow \emptyset$

References

[1] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon, 'Distributed reasoning in a peer-to-peer setting', Technical Report 1385, Université Paris-Sud XI, (2004). Available at http://www.lri.fr/~goasdoue/biblio/ACGRS-TR-1385.pdf.

[2] A. Halevy, Z. Ives, I. Tatarinov, and Peter Mork, 'Piazza: data management infrastructure for semantic web applications', in *WWW'03*, (2003).

[3] W. Nedjl, 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 *WWW'02*, (2002).