Peer-to-Peer Query Answering with Inconsistent Knowledge

Arnold Binas and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
{abinas,sheila}@cs.toronto.edu

Abstract

Decentralized reasoning is receiving increasing attention due to the distributed nature of knowledge on the Web. We address the problem of answering queries to distributed propositional reasoners which may be mutually inconsistent. This paper provides a formal characterization of a prioritized peer-to-peer query answering framework that exploits a priority ordering over the peers, as well as a distributed entailment relation as an extension to established work on argumentation frameworks. We develop decentralized algorithms for computing query answers according to distributed entailment and prove their soundness and completeness. To improve the efficiency of query answering, we propose an ordering heuristic that exploits the peers' priority ordering and empirically evaluate its effectiveness.

1 Introduction

With the advent of the Web has come a significant increase in the availability of information from a variety of sources, not all of which are mutually consistent or equally reliable. These information sources are often databases, but many foresee a future in which some of them will be deductive databases, logic programs, or even full-fledged logical reasoners. Motivated by this general problem, this paper addresses the problem of peer-to-peer (P2P) query answering over distributed propositional information sources that may be mutually inconsistent. We assume the existence of a priority ordering over the peers to discriminate between peers with conflicting information. This ordering may reflect an individual's level of trust in an information source or it may be obtained from an objective third party rating. We provide a formal characterization of a prioritized P2P query answering framework and a distributed entailment relation related to argued entailment [5] and argumentation frameworks [13]. To realize the specification of our problem, we develop decentralized algorithms based on [1] for computing answers to arbitrary CNF queries according to distributed entailment and prove their soundness and completeness. To improve the efficiency of reasoning, we propose heuristic and pruning techniques that exploit the priority ordering over peers and their knowledge and empirically illustrate their effectiveness.

There has been significant previous work on distributed logical reasoning. For example, Amir and McIlraith first introduced partition-based logical reasoning for propositional and first-order logic (FOL) to reason with consistent distributed knowledge bases (KBs) or with large KBs that they partitioned. Their message-passing approach to consequence finding was limited to partitioned KBs that were consistent and connected in a tree topology [3]. Adjiman et al. introduced the first consequence-finding algorithm for distributed propositional theories connected by graphs of arbitrary topology, but limited such systems to be globally consistent and not include priorities [1]. Chatalic et al. extended this approach to allow for mutually inconsistent peers that are connected by mapping clauses [12]. However, their approach allows for a formula and its negation to be derived as consequences at the same time. We build on this work, addressing a different general problem. We discuss other related work in the final section.

2 P2P Query Answering Systems

In this section we formalize a P2P query answering framework and provide a distributed entailment relation for such systems. We illustrate concepts via a running example.

2.1 Framework

A P2P query answering system (PQAS) consists of multiple peers, all of which are assigned priorities to reflect user-specific preferences or trust. Priorities could be universal or user specific—distributed to the peers when a user joins the network. In this paper, priorities are drawn from a totally ordered set of comparable elements (such as the set of integers), and a priority is *better* than another priority if its value is *lower*. The peers' priorities define a total or partial preference ordering over the peers. Knowledge from a peer with better priority is preferred over that from a peer with worse priority. Each peer hosts a consistent propositional local KB and consequence relation, which may be classical entailment or any other consequence relation. Multiple peers' KBs may be mutually inconsistent. In the general framework, local consequence relations may vary to allow for systems of different desired behaviors (classical, locally or globally non-monotonic, etc.). In Section 3, we develop a query answering algorithm for the special but interesting

case in which each peer's local consequence relation is classical entailment. Peers are connected by edges, which are labeled by the sets of variables that determine the language in which formulas can be passed between peers. This corresponds to restrictions on the information that KBs will share with each other.

Definition 1 (Peers). *A peer P_i is a triple $(KB_i, Cons_i, I_i)$ comprising a local propositional knowledge base KB_i, its local consequence relation $Cons_i$, and the peer's priority I_i. The notation $\Sigma \models_i \phi$ is used to denote $\phi \in Cons_i(\Sigma)$, where Σ is a set of formulas. Peer P_i is said to have better priority than peer P_j iff $I_i < I_j$. A peer's signature L_i is the set of propositional symbols in its knowledge base.*

Definition 2 (P2P query answering system (PQAS)). *A P2P query answering system $\mathcal{P}$ is a tuple (P, G) where $P = \{P_i\}_{i=1}^n$ is a set of n peers and G is a graph (V, E) describing the communication connections between those peers. $\bigcup_{i=1}^n KB_i$ is called the global theory of $\mathcal{P}$. Each peer's signature L_i is a subset of the global signature $L = \bigcup_{j=1}^n L_j$ of $\mathcal{P}$, which gives rise to the global language $\mathcal{L}$. V is the set $\{1, \ldots, n\}$ of vertices in G, with vertex i corresponding to peer P_i. E is the set of labeled edges (i, j, L_{ij}) in G, where L_{ij} is the edge label (link signature) between peers P_i and P_j and $L_{ij} \subseteq L_i \cap L_j$.*

Figure 1 is a small example of a PQAS consisting of the peers $P = \{Bookstore, Visa, CCOne, CCTwo, Customer\}$ corresponding to peers P_1–P_5, respectively. The priority I_i of peer i is displayed on its upper right corner. The bookstore's KB, KB_1, contains the axioms $\neg pmt\_rcvd \wedge ord \rightarrow \neg del\_ontime$ and $pmt\_rcvd \wedge ord \rightarrow del\_ontime$, stating that the book is delivered on time if payment is received and an order placed and not delivered on time if no payment is received. The Visa peer contains the KB $KB_2 = \{paid \rightarrow pmt\_rcvd\}$, stating that payment is received if money is paid. CCTwo (KB_4) additionally requires confirmation ($conf$). The unreliable CCOne has worse priority than the Visa peer and contains the KB $KB_3 = \{\neg pmt\_rcvd, paid \rightarrow lost\}$, stating that payment will not be received. The edges are labeled by the propositions which pairs of peers may use to communicate. An edge label between two peers only contains propositions which are mentioned in both peers' local KBs, but not necessarily all such propositions (perhaps for reasons of confidentiality). For example, the Visa and CCOne peers both contain the propositions pmt_rcvd and $paid$, but are not connected by an edge labeled with them as they represent confidential financial information.

2.2 Distributed Entailment

The distributed entailment we wish to achieve is best introduced through our example in Figure 1. Consider the case where Customer has asserted ord and $paid$ and poses the query del_ontime. According to Bookstore, the book is delivered on time if an order has been placed and payment received. If the book is ordered and payment not received, the book will not be delivered on time ($\neg del\_ontime$). According to Visa, payment is received ($paid$ and $pmt\_rcvd \vee \neg paid$ imply pmt_rcvd), and del_ontime derived. However, according to credit card peer CCOne, payment will not be

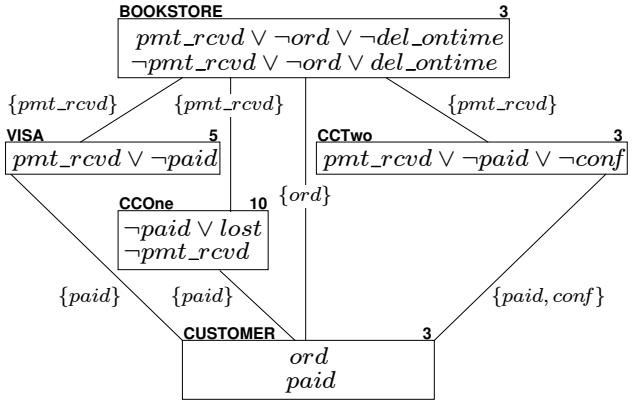

Figure 1: Bookstore example

received by the bookstore ($\neg pmt\_rcvd$) and $\neg del\_ontime$ is derived. Clearly we have a contradiction (del_ontime and $\neg del\_ontime$). But since Visa is preferred over CCOne due to its better priority, we would like to receive the answer it supports. Hence we need del_ontime to be entailed by the system according to *distributed entailment*.

In the following, we formally define this new notion of entailment by appealing to argumentation frameworks [13] and extending them to the distributed and prioritized case. To this end, we employ the notion of *reasons* which was used to define argued entailment for prioritized centralized KBs in [5]. Benferhat et al. defined a formula to be entailed by argued entailment if a better reason exists for it than for its negation. A reason for a formula is a *consistent* subset of the KB that entails the formula (since an inconsistent one could derive any formula and would thus be meaningless). In our distributed setting, we extend this definition to account for several, distributed KBs, possibly different local consequence relations, and restricted sharing of information between KBs. To this end, we define a support relation intended to mention all formulas that can be derived from consistent subsets of the global theory while obeying the link languages between peers. All formulas that can be derived by an individual peer from its local KB are mentioned in the relation. Inductively, all formulas that can be derived at an individual peer using its local KB as well as formulas already mentioned by the support relation which can be communicated to the peer according to the link languages shared with its neighbors are also in the support relation.

Definition 3 (Support of a formula by a reason). *Let $\mathcal{P} = (P, G)$ be a PQAS with $P = \{P_1, \ldots, P_n\}$ and $G = (V, E)$. The support relation $SUPP_{\mathcal{P}} \subseteq \{1, \ldots, n\} \times 2^{KB_1 \cup \cdots \cup KB_n} \times \mathcal{L}$ is the smallest set such that:*

- *If $\Sigma \subseteq KB_i$, $\Sigma \models_i \rho$, and for no $\Sigma^* \subset \Sigma$, $\Sigma^* \models_i \rho$, $(i, \Sigma, \rho) \in SUPP_{\mathcal{P}}$.*
- *If*
  - *there exists a set of formulas $\psi_1, \ldots, \psi_m$ s.t. $\{\psi_1, \ldots, \psi_m\} \models_i \rho$,*
  - *for no $\Psi \subset \{\psi_1, \ldots, \psi_m\}$, $\Psi \models_i \rho$,*
  - *for each ψ_j, there exists a peer P_k and a set of formulas*

Σ_j *s.t.* $(k, \Sigma_j, \psi_j) \in SUPP_{\mathcal{P}}$ *and* $(i, k, L_{ik}) \in E$ *with* $L_{ik} \supseteq sig(\psi_j)$*, and*

- $\bigcup_{j=1}^{m} \Sigma_j \not\models \bot$,

then $(i, \bigcup_{j=1}^{m} \Sigma_j, \rho) \in SUPP_{\mathcal{P}}$.

The formula ϕ *is supported in* $\mathcal{P}$ *by the reason* Σ*, denoted* $\mathcal{P} \models^{\Sigma} \phi$*, if for some* i*,* $(i, \Sigma, \phi) \in SUPP_{\mathcal{P}}$.

Thus a reason for a formula ϕ exists in a PQAS $\mathcal{P}$ if some consistent subset of the peers' local KBs derives ϕ given the peers' local consequence relations and given that whenever two formulas of different peers need to interact, an edge between the peers exists and is labeled by all variables mentioned in the formula. Note a fine subtlety regarding the formula sets in the definition. Formulas coming from a single peer should be minimal, i.e. a reason should contain all and only those formulas required to entail the given formula. Eliminating extraneous formulas in the reason avoids the generation of unnecessary inconsistencies with other formulas that will be added to the reason in later inductive applications of the definition. The minimality constraint does not apply for merged sets of formulas from several peers because of the potentially limited connections between them. In other words, two seemingly redundant formulas from two different peers' KBs may both be required to derive a desired consequence because they cannot travel between the two peers if the peers are not directly connected or the edge connecting them is not labeled by all the symbols occurring in the formula.

Since a reason may exist for both a formula ϕ and its negation $\neg\phi$, we need to weigh the priorities of the formulas in both reasons in order to decide whether to believe ϕ or $\neg\phi$. The priority of a formula is that of its host peer. A reason along with its conclusion will be called an argument to lay the basis for applying Dung's argumentation theory [13]. The rank of an argument is the priority of the worst-priority formula in its reason.

Definition 4 (Arguments (after [13]))**.** *An argument* A *is a pair* $A = (\Sigma, \phi)$ *s.t.* Σ *is a reason for* ϕ*.* ϕ *is called the conclusion of the argument.*

Definition 5 (Rank of an argument)**.** *Let* $A = (\Sigma, \phi)$ *be an argument. The rank of* A*, denoted* $\mathcal{R}(A)$*, is the priority of the worst-priority formula in* A*'s reason* Σ*. I.e.* $\mathcal{R}(A) = \max_{\sigma \in \Sigma}(\sigma.prio)$.

While arguments are, by definition, based on reasons that are consistent in themselves, they may be contradicted, or *attacked*, by other arguments of better rank. The set of arguments along with this attack relation is referred to as the *argumentation framework* induced by a PQAS.

Definition 6 (Attacking arguments (after [13]))**.** *Let* $A = (\Sigma_A, \phi_A)$ *and* $B = (\Sigma_B, \phi_B)$ *be arguments.* A *is said to attack* B *iff* $\phi_A \cup \Sigma_B \models \bot$ *and* $\mathcal{R}(A) \leq \mathcal{R}(B)$.

Definition 7 (Argumentation frameworks (after [13]))**.** *The argumentation framework induced by a PQAS* $\mathcal{P}$ *is* $AF(\mathcal{P}) = (AR, attacks)$*, where* AR *is the set of arguments in* $\mathcal{P}$ *and* $attacks$ *their attack relation.*

The following definitions from [13] then develop the notion of a preferred extension of an argumentation framework.

Definition 8 (Conflict-free argument sets [13])**.** *A set* S *of arguments is conflict-free if no argument in* S *is attacked by another argument in* S.

Definition 9 (Acceptable arguments and admissible argument sets [13])**.** *Let* $AF(\mathcal{P})$ *be a PQAS-induced argumentation framework. An argument* $A \in AR$ *is acceptable wrt. an argument set* $S \subseteq AR$ *iff for each argument* $B \in AR$*, if* B *attacks* A *then* B *is attacked by some argument in* S*. A conflict-free set of arguments* S *is admissible iff each argument in* S *is acceptable wrt.* S.

Definition 10 (Preferred extensions [13])**.** *A preferred extension of an argumentation framework* AF *is a maximal admissible set of* AF.

Given the notion of distributed arguments and the priority-based attack relation between them developed above, distributed entailment is then defined in terms of Dung's preferred extensions on the induced argumentation framework. More specifically, we will believe formulas that are conclusions of arguments that are in some preferred extension of the induced argumentation framework, i.e. those supported and not successfully attacked by the most preferred peers of $\mathcal{P}$.

Definition 11 (Distributed Entailment)**.** *A formula* ϕ *is entailed by the PQAS* $\mathcal{P}$ *by distributed entailment, denoted* $\mathcal{P} \models_D \phi$*, iff* ϕ *is the conclusion of an argument in some preferred extension of* $AF(\mathcal{P})$.

Note that this argumentation-based definition of distributed entailment means that both a formula and its negation can be entailed by distributed entailment at the same time. This may occur in cases where arguments for and against the query have the same rank. To illustrate this, consider the simple example of a PQAS-induced argumentation framework containing only the arguments $A = (\{p\}, p)$ supporting the formula p and $B = (\{\neg p\}, \neg p)$ supporting the formula $\neg p$. If both A and B have the same rank, A attacks B and B attacks A. Thus there are two preferred extensions: one containing only A and one containing only B. Such cases are easily detected and can be reported by a query answering algorithm, as will be done by the algorithm we present in the next section.

While in the general case, a PQAS's peers may be mutually inconsistent, an inconsistency may not always be derived. This could be because all peers are mutually consistent or because the inconsistency is not derivable with the restricted communication imposed by peer connectivity. To distinguish between PQASs that can derive contradictions and those that cannot, we introduce the concept of $\mathcal{P}$-consistency.

Definition 12 ($\mathcal{P}$-consistency)**.** *A PQAS* $\mathcal{P}$ *is said to be* $\mathcal{P}$*-inconsistent if there exists a formula* ϕ *s.t. there exists both a reason for* ϕ *and a reason for* $\neg\phi$ *in* $\mathcal{P}$*. Otherwise* $\mathcal{P}$ *is said to be* $\mathcal{P}$*-consistent.*

Since in a $\mathcal{P}$-consistent system no contradictions (attacks) can be derived, the existence of a reason for a formula ϕ guarantees that the corresponding argument is in the preferred extension formed by the entire argument set and thus that ϕ is entailed by distributed entailment in $\mathcal{P}$.

The following theorem shows that for fully connected systems with classical local entailment, P-consistency is equivalent to the satisfiability of the global theory.

Theorem 1. *Let $\mathcal{P} = (P, G)$ be a PQAS such that if for any two peers P_i and P_j, $L_{ij} = L_i \cap L_j$ is non-empty, $(i, j, L_{ij}) \in E$ and for all peers P_i, $Cons_i$ is the classical entailment relation $\models$. Then $\mathcal{P}$ is $\mathcal{P}$-consistent iff $\bigcup_{i=1}^{n} KB_i$ is satisfiable. Otherwise it is $\mathcal{P}$-inconsistent.*

Proof sketch. Since $\mathcal{P}$ is fully connected, every formula in the system can interact with any other formula in the systems as if they were part of a single theory. Hence, there is an argument for each formula that can be derived from the global theory.

$\Rightarrow$: If $\mathcal{P}$ is $\mathcal{P}$-consistent, there exists no formula ϕ s.t. there is a reason for both ϕ and $\neg\phi$ (Definition 12). Hence since the system is fully connected, for no formula ϕ, both ϕ and $\neg\phi$ can be derived from the global theory (Definition 3). Thus the empty clause cannot be derived from the global theory, which is hence satisfiable.

$\Leftarrow$: If the global theory is satisfiable, there does not exist a formula ϕ s.t. both ϕ and $\neg\phi$ can be derived from the global theory. Thus since the system is fully connected, there exists no formula ϕ s.t. there exists a reason for both ϕ and $\neg\phi$ (Definition 3) and therefore $\mathcal{P}$ is $\mathcal{P}$-consistent (Definition 12). ■

The following theorem further establishes that in a $\mathcal{P}$-consistent system with full peer connectivity and classical local entailment, distributed entailment as defined through preferred extensions in the induced argumentation framework is equivalent to classical entailment from the global theory.

Theorem 2. *Let $\mathcal{P} = (P, G)$ be a $\mathcal{P}$-consistent PQAS such that if for any two peers P_i and P_j, $L_{ij} = L_i \cap L_j$ is non-empty, $(i, j, L_{ij}) \in E$ and for all peers P_i, $Cons_i$ is the classical entailment relation $\models$. Then $\mathcal{P} \models_D \phi$ iff $\bigcup_i KB_i \models \phi$.*

Proof sketch. Since $\mathcal{P}$ is fully connected, every formula in the system can interact with any other formula in the systems as if they were part of a single theory. Hence, there is an argument for each formula that can be derived from the global theory. Since $\mathcal{P}$ is also $\mathcal{P}$-consistent, no argument attacks any other argument and the argument set containing at least one argument for each formula in the classical deductive closure of $\bigcup_i KB_i$ is the lone preferred extension of $AF(\mathcal{P})$; i.e. each formula that is in the deductive closure of $\bigcup_i KB_i$ is also entailed in $\mathcal{P}$ by distributed entailment and vice versa. ■

3 Query Answering in a PQAS

Given the PQAS framework, we want to pose a formula as a query and have the system determine the status of the query according to our definition of distributed entailment by using individual peers' local knowledge and reasoning capabilities. The goal of this section is to develop a message-passing algorithm that solves this problem for arbitrary CNF queries in the special case of a possibly $\mathcal{P}$-inconsistent PQAS in which all peers use classical entailment as their local consequence relation.

A consequence-finding algorithm for systems of mutually consistent peers already exists [1]. Adjiman et al.'s algorithm computes consequences of individual literals for arbitrary peer topologies and shall serve as the basis for our query answering algorithm. Here we modify and extend Adjiman et al.'s algorithm to find the distributed consequences of individual literals as well as their reasons and priorities in a possibly $\mathcal{P}$-inconsistent PQAS $\mathcal{P}$. These consequences and their reasons are then used to compute query answers according to distributed entailment. Our focus on prioritized peers enables a search space pruning technique and search heuristic that drastically improves performance. A query is posed by a query peer that may be one of the peers of $\mathcal{P}$ or an additional peer connected to $\mathcal{P}$.

3.1 Algorithm

To answer an arbitrary CNF query Q to peer P according to distributed entailment, Algorithm 1 finds the ranks of the best-rank arguments from preferred extensions of $AF(\mathcal{P})$ for both Q and $\neg Q$ by passing the negation of each subquery to Find_Bottom_Distr of Algorithm 2 for a refutation proof and returns the answer corresponding to the best-rank argument. $BOTH$ is returned if the best arguments for and against the query have equal rank and UNK (for unknown) if no argument for either exists in any preferred extension.

Find_Bottom_Distr($Q, P, best$) finds the best-rank way to derive the empty clause ($\square$) from Q relying on an argument of rank no worse than $best$ and which is in a preferred extension of $AF(\mathcal{P})$. Arguments with reason Σ that are in no preferred extension due to being attacked are detected by recursive calls to Find_Bottom_Distr($\Sigma, P, best$) and ignored. To find the best-rank way of deriving the empty clause from the CNF query Q, it is split into its individual literals, and each literal's distributed consequences that rely on arguments from preferred extensions (OA sets in the algorithms) are found by sending a *forth* message for each. The returned *back* messages are collected and merged back together to form the consequences of the original clauses in Q. We will outline the workings of the message-passing procedures and give an example in a moment. Given the consequences of Q, their OA sets (reasons) are tested for satisfiability (to comply with Definition 3) and for being in a preferred extension of the argumentation framework induced by the PQAS (to comply with Definition 11). The latter is done by recursive calls to Find_Bottom_Distr, which is only asked to find contradicting arguments of rank worse or equal to the reason attacks on which are to be found. Find_Bottom_Distr then calls the subroutine Find_Empty_Clauses to find contradictions in the deductive closure of all the consequences of the passed-in (negated) query Q, which indicate a refutation proof for the original query. The satisfiability and attack checks are repeated on those proof candidates and the rank of the best-rank argument from a preferred extension returned.

Reasons for individual literal's consequences are computed by the message-passing Algorithms 3, 4, 5, and 6 and then combined to find reasons for the original CNF query. In order to compute reasons for a query literal p's conse-

quences, a *forth* message is sent to a peer whose signature contains p and *back* messages with the consequences collected. A history is used to keep track of which literals and clauses the currently processed literal depended on, enabling the algorithm to detect if a literal has been processed by the same peer before or whether it depends on its own negation in the history and thus derives the empty clause. Each derived clause has an associated OA set (for "original ancestor") which contains all original parent clauses and is used to reconstruct the reasons for consequences and their ranks. The following notation, proposed by [1], is used in the message-passing algorithms, which extend Adjiman et al.'s by priorities, OA sets, and attack checks.

Definition 13 (History ([1]))**.** *A history $hist$ is a list of tuples (l, P, c) of a literal l, a peer P, and a clause c. c is a consequence of l in P, and l is a literal of the clause of the previous tuple in the $hist$ list.*

Definition 14 (Local consequences ([1]))**.** *$Resolvent(l, P_i)$ is the set $\{c | KB_i \models_i c \vee \neg l\}$ of local consequences in peer P_i of the literal l.*

Definition 15 (Acquainted peers by literal ([1]))**.** *$ACQ(l, P)$ is the set $\{P' \in V | (P, P', L_{ij}) \in E, l \in L_{ij}\}$ of peers sharing an edge labeled by L_{ij} with peer P, where $l \in L_{ij}$.*

Definition 16 (Distributed disjunction ([1]))**.** *The distributed disjunction operator $\otimes$ in $A \otimes B$ forms clauses by disjoining all combinations of clauses in the sets A and B . For an indexed set of clause sets $\{A_i\}_i$, the notation $\otimes_{i \in \{1,2,3\}} A_i$ means $A_1 \otimes A_2 \otimes A_3$.*

Each peer can send and receive four message types during the search for consequences of the query. *Forth* messages (Algorithm 3) request the search for consequences of a literal by neighboring peers. The receiving peer computes all local consequences of the literal of the message (line 15). When two clauses are resolved, their original ancestors are added to the resolvent's OA set. Any consequences with consistent OA sets are returned to the sender peer via a *back* message if a recursive call to Find_Bottom_Distr verifies that their reasons are not attacked without defense (line 23). Then the local consequence clauses are split into their individual literals and each sent to the connected peers (if any) via another *forth* message (line 35). *Back* messages (Algorithm 4) are sent back to the sender peer when a consequence with satisfiable and non-attacked OA set is derived. Upon receiving a *back* message, a peer stores the contained consequence of the literal of the corresponding *forth* message (line 3). If their respective OA sets are mutually satisfiable and not attacked, one *back* message for each combination of per-literal consequences is sent to the last peer in the history $hist$ (line 17). *Final* messages (Algorithm 5) indicate that the exploration of a particular search branch is completed (Algorithm 3, lines 3, 5, 28, and 37). *Prio* messages (Algorithm 6) are sent when a new argument from a preferred extension is founds whose rank is better than that of any such argument for the original query known so far (Algorithm 2, line 29). *Prio* messages serve to update each peer's local value of $best(id)$, which determines the priority at which peers and clauses of the reasoning process id can be safely pruned from the search space. This pruning of the search space deserves highlighting and results in a significant improvement in the performance of the system.

Algorithm 1 Distributed query answering algorithm

```
1: Answer_Query_Distr(Q)
2: Q̄ ← ¬Q in CNF
3: for all P ∈ 𝒫 do
4:     pos_P ← Find_Bottom_Distr(Q̄, P, ∞)
5:     neg_P ← Find_Bottom_Distr(Q, P, ∞)
6: pos ← min_P(pos_P)
7: neg ← min_P(neg_P)
8: if pos < neg then
9:     return YES
10: if neg < pos then
11:     return NO
12: if neg = pos & pos < ∞ then
13:     return BOTH
14: return UNK
```

Algorithm 2 Finding best-rank arguments

```
1: Find_Bottom_Distr(Q, P, best)
2: id ← newID()
3: best(id) ← best
4: for all clauses c ∈ Q do
5:     for all literals l ∈ c do
6:         send m(Self, P, forth, [(l, Self, c)|hist], l, id)
7:         Cons_{c,l} ← ∅
8: best_prio ← ∞
9: while final message not received do
10:     for all back messages received for clause c and literal l with
        ID id, containing consequence cons do
11:         Cons_{c,l} ← Cons_{c,l} ∪ {cons}
12:     for all clauses c ∈ Q do
13:         Cons_c ← ⊗_{l∈c} Cons_{c,l}
14:         c.prio ← max(l.prio)
15:         c.OA ← ⋃_{l∈c} l.OA
16:     Cons ← ⋃_{c∈Q} Cons_c
17:     for all c ∈ Cons do
18:         if c.prio > best then
19:             Cons ← Cons\{c}
20:     for all c ∈ Cons do
21:         if c.OA is UNSAT or
            Find_Bottom_Distr(c.OA, P, c.prio) ≤ c.prio then
22:             Cons ← Cons\{c}
23:     □_Cons ← Find_Empty_Clauses(Cons)
24:     for all □ ∈ □_Cons do
25:         if □.OA is UNSAT or
            Find_Bottom_Distr(□.OA, P, □.prio)≤ □.prio then
26:             □_Cons ← □_Cons\□
27:     best_prio ← min( min_{□∈□_Cons}(□.prio), best_prio )
28:     if best_prio < best(id) then
29:         send m(Self, P, prio, best_prio, id)
30: return best_prio
```

3.2 Example

Returning to the bookstore example of Figure 1, Customer asserts ord and $paid$ and asks at the bookstore whether the book will be delivered on time. This is a single-literal query which Algorithm 1 asks positively and negatively via Algo-

Algorithm 3 *Forth* message algorithm

ReceiveForthMessage(m($Sender, Self, forth, hist, p, id$))
if $(p, Self, \_) \in hist$ **then**
  send m($Self, Sender, final, [(p, Self, true)|hist], id$)
else if $p.prio > best(id)$ or $Self.prio > best(id)$ **then**
  send m($Self, Sender, final, [(p, Self, true)|hist], id$)
else
  if $(\neg p, \_, \_) \in hist$ **then**
    $hist$ is of the form $[(l', \_, c')|hist']$
    if $c'.OA$ is SAT **then**
      let $\Box$ be a new empty clause
      $\Box.OA \leftarrow c'.OA$
      $\Box.prio \leftarrow c'.prio$
      if Find_Bottom_Distr($\Box.OA, Self, \Box.prio$)> $\Box.prio$ **then**
        send m($Self, Sender, back, [(p, Self, \Box)|hist], \Box, id$)
  LOCAL($Self$) $\leftarrow \{p\} \cup Resolvent(p, Self)$
  for all $c \in$ LOCAL($Self$) **do**
    let $\{c_i^*\}_i$ be the set of clauses that went into c
    $c.OA \leftarrow \bigcup_i c_i^*.OA$ (recursively)
    $c.prio \leftarrow \max_i(c_i^*.prio)$ (recursively)
  LOCAL($Self$) $\leftarrow \{c \in$LOCAL($Self$)$| c.prio \leq best(id)\}$
  $temp\_min \leftarrow \infty$
  for all $c \in$ LOCAL($Self$) s.t. $c.OA$ is SAT **do**
    if Find_Bottom_Distr($c.OA, Self, c.prio$)> $c.prio$ **then**
      send m($Self$,$Sender$,$back$,$[(p, Self, c)|hist]$,c,id)
      $temp\_min \leftarrow$ min($temp\_min, c.prio$)
  LOCAL($Self$) $\leftarrow \{c \in$ LOCAL($Self$) | all literals in c are shared$\}$
  if LOCAL($Self$) $= \emptyset$ **then**
    send m($Self$,$Sender$,$final$,$[(p, Self, true)|hist]$,id)
  for all $c \in$ LOCAL($Self$) **do**
    for all $l \in c$ **do**
      CONS($l, [(p, Self, c)|hist]$) $\leftarrow \emptyset$
      $ACQ^* \leftarrow \{P' \in ACQ(l, Self)|P'.prio \leq best(id)\}$
      for all $RP \in ACQ^*$ **do**
        FINAL($l, [(p, Self, c)|hist], RP$) $\leftarrow false$
        send m($Self$,RP,$forth$,$[(p, Self, c)|hist]$,l,id)
  if no *forth* message sent **then**
    send m($Self$,$Sender$,$final$,$[(p, Self, true)|hist]$,id)

Algorithm 4 *Back* message algorithm

ReceiveBackMessage(m($Sender, Self, back, hist, c^*, id$))
$hist$ is of the form $[(l', Sender, c'), (p, Self, c)|hist']$
CONS($l', [(p, Self, c)|hist']$) $\leftarrow$
CONS($l', [(p, Self, c)|hist']$) $\cup\ c^*$
RESULT $\leftarrow (\otimes_{l \in c \setminus \{l'\}}$CONS($l, [(p, Self, c)|hist']$)$) \otimes \{c^*\}$
for all $cl \in$ RESULT s.t. cl contains a distributed consequence for each $l \in c$ **do**
  let $\{c_i^*\}_i$ be the set of distributed consequence clauses that went into cl
  $cl.OA \leftarrow \bigcup_i c_i^*.OA$
  $cl.prio \leftarrow \max_i(c_i^*.prio)$
RESULT $\leftarrow \{cl \in$ RESULT$|cl.prio \leq best(id)$ and $cl.OA$ is SAT$\}$
if $hist' = \emptyset$ **then**
  $U \leftarrow User$
else
  $U \leftarrow$ the first peer P' of $hist'$
$temp\_min \leftarrow \infty$
for all $cl' \in$ RESULT **do**
  if Find_Bottom_Distr($cl'.OA, Self, cl'.prio$)> $cl'.prio$ **then**
    send m($Self, U, back, [(p, Self, c)|hist'], cl', id$)
    $temp\_min \leftarrow$ min($temp\_min, cl'.prio$)

Algorithm 5 *Final* message algorithm

ReceiveFinalMessage(m($Sender, Self, final, hist, id$))
$hist$ is of the form $[(l', Sender, true), (p, Self, c)|hist']$
FINAL($l', [(p, Self, c)|hist'], Sender$) $\leftarrow true$
if $\forall c^* \in$ LOCAL($Self$) and $\forall l \in c^*$, FINAL($l, [(p, Self, c^*)|hist'], \_) = true$ **then**
  if $hist' = \emptyset$ **then**
    $U \leftarrow User$
  else
    $U \leftarrow$ the first peer P' of $hist'$
  send m($Self, U, final, [(p, Self, true)|hist'], id$)

Algorithm 6 *Prio* message algorithm

ReceivePrioMessage(m($Sender, Self, prio, x, id$))
if $x < best(id)s$ **then**
  $best(id)s \leftarrow x$
  for all $RP \in ACQ(\_, Self)$ s.t. $RP \neq Sender$ **do**
    send m($Self, RP, prio, best(id), id$)

rithm 2. Algorithm 2 passes each on as a *forth* message. At the Bookstore peer, $\neg del\_ontime$ generates the local consequence $\neg pmt\_rcvd \vee \neg ord$ which is split into its individual literals. A *forth* message containing $\neg pmt\_rcvd$ is sent to Visa, CCOne, and CCTwo. Visa generates $\neg paid$ which resolves with $paid$ to the empty clause at Customer. The empty clause with its OA set is passed back to Visa and then to Bookstore via *back* messages. $\neg pmt\_rcvd$ generates no consequences in CCOne or CCTwo, and final messages are sent for these branches. The $\neg ord$ of the consequence in Bookstore is sent as a *forth* message to Customer where it resolves with ord to the empty clause and is passed back as a *back* message. The empty clauses for $\neg pmt\_rcvd$ and $\neg ord$ are merged back together at the bookstore and the resulting OA set $\{\neg pmt\_rcvd \vee \neg ord \vee del\_ontime, pmt\_rcvd \vee \neg paid, paid, ord\}$ is determined to be consistent. Thus a reason for del_ontime is found with rank 5 (max over the three peers contributing clauses to the reason). It is verified that the argument this reason is in is in a preferred extension by recursively passing it to Algorithm 2 and observing that no contradiction is derived (and the reason is thus not attacked). Similarly the empty clause can be derived from del_ontime using Bookstore, CCOne, and Customer, resulting in a reason of rank 10 for $\neg del\_ontime$. The argument with this reason, however, is determined to be attacked by a recursive call to Algorithm 2 and thus ignored. Since del_ontime is inferred with better rank than $\neg del\_ontime$, YES is returned as the answer in Algorithm 1.

3.3 Analysis

We proved various results for the algorithms and restate the most important theorems here. The proofs are outlined in as much detail as space permits. Our ultimate goal in this analysis is to show that Algorithm 1 is sound and complete with respect to distributed entailment (Definition 11). To this end, we will first establish that Algorithms 3, 4, and 5 are sound and complete wrt. computing the reasons of distributed consequences of individual literals and then use this result to show that Algorithm 2 computes the rank of the best-rank argument for the query that belongs to a preferred extension of the argumentation framework $AF(\mathcal{P})$ induced by the PQAS $\mathcal{P}$. In order to make the line of argumentation easier to follow, we will first consider the algorithms without priority-based pruning and without recursive calls to Find_Bottom_Distr of Algorithm 2 (i.e. without checking reasons for being attacked). Without attack checks, the arguments that the computed reasons belong to are valid arguments, but do not necessarily belong to a preferred extension of $AF(\mathcal{P})$, which is necessary for distributed entailment. We gradually add both attack checks and priority-based pruning back in to arrive at our final soundness and completeness result.

We proceed by establishing the following in this order, each bullet corresponding to a theorem.

- Computation of reasons for consequences of individual literals by Algorithms 3, 4, and 5 without priority-based pruning and attack checks
- Computation of reasons for arbitrary CNF queries by Algorithm 2 without priority-based pruning and attack checks
- Computation of reasons for arbitrary CNF queries by Algorithm 2 without priority-based pruning but with attack checks
- Termination of Algorithm 2
- Correctness of priority-based pruning
- Soundness and completeness of Algorithm 1 wrt. distributed entailment

First, the following theorem states that, without priority-based pruning and attack checks, the message-passing algorithms find the reasons of distributed consequences of individual literals. Both theorem and proof somewhat resemble those in [1], but additionally take into account link languages and validity of reasons (satisfiability of OA sets).

Theorem 3 (Soundness and completeness wrt. computing reasons for consequences of individual literals)**.** *Let $\mathcal{P}$ be a possibly $\mathcal{P}$-inconsistent PQAS. The following holds for Algorithms 3, 4, and 5 without calls to Find_Bottom_Distr of Algorithm 2 and without priority-based pruning. There exist peers P_i and P_j and a literal p in L_{ij} s.t. P_i receives the message $m(P_j, P_i, back, [(p, P_j, c)], c)$ as a response to the message $m(P_i, P_j, forth, \emptyset, p)$ iff there is a set of formulas Σ s.t. $\mathcal{P} \models^{\Sigma} p \rightarrow c$, where c is a clause.*

Proof sketch. The message passing Algorithms 3, 4, and 5 can be shown to be sound and complete for computing distributed consequences of an individual literal, i.e. those for which a reason exists (by Definition 3). This is proved by induction on the length of the history $hist$ by relating the merging of distributed consequences of individual literals of a clause (line 4 of Algorithm 4) to the inductive step of Definition 3 and checking the satisfiability of OA sets to insure consistent reasons. The full proof is very similar to that in [1], but additionally accounts for satisfiability checks of OA sets and for the constraints imposed by link languages (in the inductive step of Definition 3). ■

Theorem 4 extends the above result to full CNF queries to Algorithm 2. I.e., it is shown that without priority-based pruning and attach checks, reasons are correctly computed, although they may still be attacked.

Theorem 4 (Best-rank reasons)**.** *Let $\mathcal{P}$ be a possibly $\mathcal{P}$-inconsistent PQAS. Then for some peer P, Find_Bottom_Distr(Q, P, ∞) of Algorithm 2 without recursive calls to itself and without priority-based pruning returns the rank of the best-rank reason for $\neg Q$.*

Proof sketch. Theorem 3 establishes that the message-passing Algorithms 3, 4, and 5, invoked at some peer P connected to the peer relevant for the query, find all consequences of individual literals along with their supporting reasons. For each clause in the negated query Q in CNF, Algorithm 2 calls the message-passing algorithms to find its per-literal consequences (line 6) and merges them back together to obtain the consequences of the negated query (line 13). All contradictions ultimately caused by the negated query (since all consequences resulting from unsatisfiable OA sets are ignored, lines 21 and 25) are found in the deductive closure of the per-clause consequences (line 23) and the priority of the best one, corresponding to the best-rank reason, is returned. ■

The result obtained in Theorem 4 is extended below to consider only reasons of arguments for the query that are in a preferred extension of $AF(\mathcal{P})$. This is achieved when considering recursive calls to Find_Bottom_Distr of Algorithm 2, which serve as attack checks on the argument with the reason under consideration. Theorem 5 is the core of establishing the soundness and completeness of Algorithm 1 as arguments in preferred extensions are the basis for distributed entailment (Definition 11).

Theorem 5 (Best-rank reasons of preferred extensions)**.** *Let $\mathcal{P}$ be a possibly $\mathcal{P}$-inconsistent PQAS. Then for some peer P, Find_Bottom_Distr(Q, P, ∞) of Algorithm 2 with recursive calls to Find_Bottom_Distr but without priority-based pruning returns the rank of the best-rank reason for $\neg Q$ that belongs to an argument in a preferred extension of $AF(\mathcal{P})$.*

Proof sketch. We already know that, without recursive calls to Find_Bottom_Distr, the algorithms find the best-rank reason for the query (Theorem 4). It remains to show that reasons which are in no preferred extension of $AF(\mathcal{P})$ are filtered out by posing the reasons to recursive calls of Find_Bottom_Distr to find attacks.

This is proved by induction on the number of recursive calls to Find_Bottom_Distr. The intuition underlying the proof is as follows. In the **base case**, Algorithm 2 returns ∞

because no consequence of Q has better or equal priority than $best$ and all of Q's consequences are thus disregarded (line 19). In this case there is no argument attacking Q, so Q is in a preferred extension of $AF(\mathcal{P})$.

We briefly outline the **induction step** as follows. In this step, a recursive call to Find_Bottom_Distr returns a priority better than $best$, indicating that an argument attacking Q exists. Call that argument Q'. Since Q is attacked by Q', it is disregarded by the algorithm (lines 21 and 25). There are two cases.

(1): Q' is not attacked by any other argument of better rank. Then Q' is a valid attack on Q and Q is thus in no preferred extension of $AF(\mathcal{P})$ and hence Q is rightfully disregarded by the algorithm.

(2): Q' is itself attacked by an argument Q'' of better rank. Then either **(a)** $Q \cup Q'' \not\models \bot$ (i.e. both reasons are consistent) or **(b)** $Q \cup Q'' \models \bot$ (i.e. they are not consistent).

(a): Q'' is in the same conflict-free set with Q and thus defends Q against the attack from Q'. In this case the algorithm rightfully disregards Q' as an attacking reason.

(b): Q'' also attacks Q directly (since both are mutually inconsistent). Thus the recursive call to Find_Bottom_Distr will return a better priority than $best$ due to Q'' already and disregarding Q' will do no harm. ■

Theorem 6 below establishes that Algorithm 2 terminates, a necessary condition for soundness and completeness.

Theorem 6 (Termination (partially following [1]))**.** *Algorithm 2 terminates.*

Proof sketch. There are two recursions in the overall algorithm; one grows the history to find consequences of consequences using the message passing algorithms, and the other checks whether reasons are attacked via recursive calls to Find_Bottom_Distr.

For each *forth* message, a *new* element is added to the history. There are finitely many variables and peers, so there are only finitely many possible history elements. Thus a history can only be of a finite length. Therefore, only finitely many *forth* messages can be sent. A *forth* message necessarily results in a *final* message and thus this recursion terminates.

The number of recursions of Find_Bottom_Distr cannot be infinite as for each priority level (parameter $best$ in Algorithm 2), there are finitely many possible attacks (or inconsistencies of better or equal rank, since the global theory is finite) and the priority limit $best$ at which possible attacks are valid is monotonically increasing (i.e. worsening) with each recursive call to Find_Bottom_Distr. ■

Given termination of Algorithm 2, it is easily shown that Algorithm 1 terminates.

In the following, we reintroduce priority-based pruning and show that soundness and completeness are retained. The pruning strategy, while not necessary for the correctness of the algorithm, allows for large performance improvements due to a significantly smaller search space.

Theorem 7 (Pruning by priority limit)**.** *The following hold for a PQAS running Algorithms 2, 3, 4, 5, and 6. (i) Whenever a peer updates its local value of* $best$*, there exists a reason for the original query or its negation with priority* $best$*. (ii) Forth messages and acquainted peers ignored by a peer* P *cannot result in a reason for the original query or its negation with better or equal rank than the current value of* $best$*.*

Proof sketch. **(i)** The algorithms only send *prio* messages if a new argument from a preferred extension for either the query or its negation has been found (line 29 Algorithm 2 and line 5 of Algorithm 6).

(ii) Ignoring a *forth* message or a peer of a certain priority amounts to ignoring a clause of that priority. In a history $[(l_n, P_n, c_n), \ldots, (l_i, P_i, c_i), \ldots, (l_1, P_1, c_1)]$, the priorities $c_i.prio$ are monotonically increasing with increasing i due to the priority of a resolvent being the max-value of the priorities of the input clauses. Thus an empty clause derived from a clause c with priority $c.prio > best$ cannot have a priority better than or equal to $best$. The empty clause corresponds to the reason (via its OA set) and its priority to the reason's rank. ■

Since Algorithm 2 finds the rank of the best-rank argument for the query (Theorem 5), and the algorithm terminates (Theorem 6), asking all peers the query and its negation results in an answer according to distributed entailment (Algorithm 1). The following theorem formalizes this main result of the present paper regarding our set of query answering algorithms.

Theorem 8 (Soundness and completeness wrt. distributed entailment)**.** *Given a PQAS* $\mathcal{P}$*, Answer_Query(*Q*) of Algorithm 1 returns* YES *iff* $\mathcal{P} \models_D Q$ *and* $\mathcal{P} \not\models_D \neg Q$*,* NO *iff* $\mathcal{P} \models_D \neg Q$ *and* $\mathcal{P} \not\models_D Q$*,* $BOTH$ *iff* $\mathcal{P} \models_D Q$ *and* $\mathcal{P} \models_D \neg Q$ *and* UNK *otherwise.*

Proof sketch. Assume that $\mathcal{P} \models_D Q$ and $\mathcal{P} \not\models_D \neg Q$. Then Answer_Query($Q$) of Algorithm 1 collects the ranks of the best-rank arguments for and against the query resulting by asking them at all peers in pos_P and neg_P, respectively (lines 4 and 5). The overall best ranks for and against the query are found (lines 6 and 7) and known to be correct (Theorem 5). Since $\mathcal{P} \models_D Q$ and $\mathcal{P} \not\models_D \neg Q$, we will have $pos < neg$ and YES is returned by the algorithm (line 9). The argument for the algorithm returning NO if $\mathcal{P} \models_D \neg Q$ and $\mathcal{P} \not\models_D Q$ is symmetric.

Assume that $\mathcal{P} \models_D Q$ and $\mathcal{P} \models_D \neg Q$. Then we will have $pos = neg < \infty$ and Algorithm 1 returns $BOTH$ (line 13).

If neither $pos < \infty$ nor $neg < \infty$, the algorithm returns UNK (line 14).

The reverse direction can be shown similarly. ■

3.4 Pruning and Ordering Heuristic

Time can be saved by the message-passing algorithms when searching for the best-rank reasons for a query and its negation simultaneously by exploiting the priority ordering over the peers and the resulting priority ordering over formulas. Since we are only interested in the best-rank argument from a preferred extension, and since clauses and peers with worse priority than the currently best known such argument are pruned away, it generally pays off for each peer to

process messages with consequences of better priority first. This technique ensures that better-rank reasons are found earlier and the priority limit $best$ is updated more quickly, resulting in a larger part of the search space being pruned. Furthermore, the best-rank reason from a preferred extension will be found earlier (although not necessarily first as the algorithm runs concurrently across distributed peers). In Section 4, we present empirical results illustrating the effectiveness of this priority-ordering heuristic.

3.5 Discussion

Since we are reasoning with peer KBs that may be mutually inconsistent, we want to ensure that any conclusions we draw are derived from a consistent subset of these KBs. Hence, before sending a found consequence in a *back* message, the algorithm checks the satisfiability of the clause's OA set, since this set could be inconsistent. This may be done either by a call to a SAT solver or by comparing the clauses in the OA set against cached *nogoods* as in [12]. Both approaches have their merits—one needs no precomputed nogoods and the other may amortize their computation over multiple queries.

In the special case of a $\mathcal{P}$-consistent PQAS the computation of query answers is less costly. Since no contradictions can be derived in such a system, a reason only exists either for the query, its negation, or neither. Furthermore, the satisfiability of clause sets and therefore also the acceptability of reasons is guaranteed in a $\mathcal{P}$-consistent system and OA sets need not be tracked nor their consistency checked. Priorities need not be tracked when trying to find the answer to a query only. In this case, the first reason for the query found will suffice and the algorithm may terminate. If the best-rank reason for the answers is of interest (for example, if priorities represent reliability), priorities must be tracked, but only the query and not its negation asked in a $\mathcal{P}$-consistent system.

4 Implementation and Experiments

To evaluate our pruning technique and ordering heuristic, we created an implementation that simulates on a single machine a PQAS where reasons are not checked for being attacked (to either leave the freedom to judge acceptability to the querying peer or when it can be assumed that reasons are not attacked in the given PQAS). Although this is a limited case of the full system, this demonstrates the effectiveness of the pruning strategy. There is a message queue for each peer, and peers take turns to process one message each. The default message queue is a first-in first-out (FIFO) queue, ensuring that messages are processed in the order received. The ordering heuristic uses a priority message queue, which is sorted by the priorities of clauses and peers of the messages and returns better-priority messages first. Our pruning strategy prunes consequences and peers of worse priority than the currently best known argument from a preferred extension.

We executed experiments to evaluate the effectiveness of both our priority-ordering heuristic and our pruning strategy. The quality of query answers need not be empirically evaluated as a full implementation of the algorithms would necessarily return the globally correct answer due to their soundness and completeness wrt. distributed entailment (Theorem 8). We ran three variants of the algorithm querying each proposition of each of a set of several hundred randomly generated PQAS instances. For a given number of of peers, instances were generated by giving each peer the same number of (distinct) propositions, sharing a random subset of them with a random selection of other peers through labeled connections, and randomly generating a fixed number of clauses of length 1–3 within each peer from its local vocabulary. Each instance had 4–20 peers with 4–8 clauses per peer, 1–6 shared propositions per peer, and a total of 8–96 distinct propositions. An incomplete attempt to answer a query by a given variant of the algorithm was aborted after ten minutes. Out of all queries given, the naive, pruning, and ordering methods solved 1341, 2292, and 2842 queries, respectively. The average number of messages required to solve a query for the three methods were 2735, 1266, and 456 messages, respectively. Figure 2 shows the results for individual queries in terms of the number of messages passed throughout the system to solve a query. The top plot contrasts the total number of messages sent until termination of the naive algorithm (without pruning) with the number of messages used when pruning. Problem instances are sorted by the number of messages used by the naive approach. The bottom plot compares the number of messages sent by the ordering heuristic version with the pruning-only version and is sorted by the hardness of problems for the latter. We chose the number of messages sent to answer a query as a performance measure as in a real-world distributed system, messages would be sent over a network, creating a bottleneck for the system. In both plots only non-trivial queries that have been solved by at least one of the three methods are shown. A non-trivial query is one that took the naive algorithm at least 300 messages to solve. Such queries took on average 52, 52, and 46 messages to solve for the naive, pruning, and ordering versions of the algorithm, respectively, so ignoring such trivial queries does not hurt the evaluation of the naive approach in the comparison with the other two approaches. Figure 3 shows the number of messages saved by the pruning and priority-ordering techniques compared to not using them for the same set of queries (and in the same order) as in Figure 2.

The graphs show that pruning can lead to a drastic reduction in the number of messages generated over the naive algorithm, and in turn, that the ordering heuristic in conjunction with pruning often beats pruning alone. In a large number of cases, the savings due to both pruning and ordering are very significant. In some cases there are no or almost no savings, and in many cases the savings are somewhere inbetween. In instances where no consequences or peers can be pruned, our pruning strategy may require slightly more messages than the naive approach (*prio* message overhead). In a few degenerate cases, the heuristic version of the algorithm can use more messages than the pruning-only version. This occurs when a medium-priority message generating a reason and $best$ update is at the front of the FIFO queue in the pruning-only algorithm, but the heuristic version processes better-priority messages that lead to no reason first,

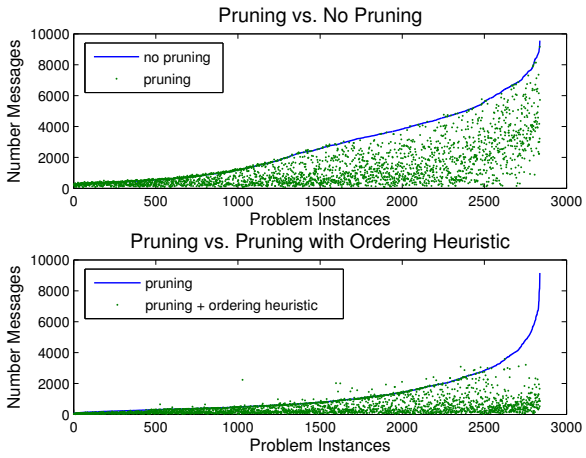

Figure 2: Total number of messages sent

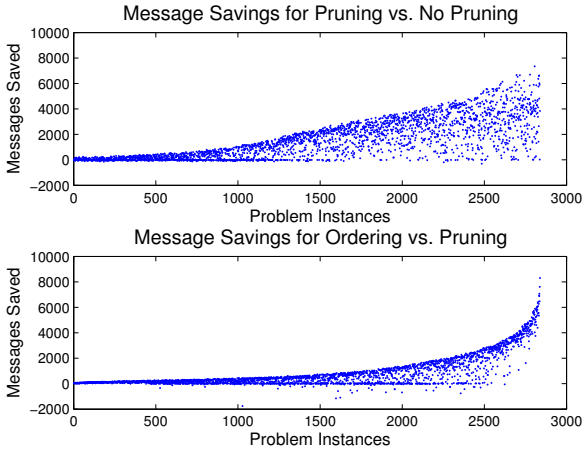

Figure 3: Number of messages saved

potentially generating messages of bad priority before the pruning cutoff variable $best$ is updated in Algorithm 6.

5 Discussion and Related Work

In this paper we provided a formal characterization of a P2P query answering system and a distributed entailment relation which extends established work on argumentation frameworks to the distributed and prioritized case. Our system includes a priority ordering over the peers (which may reflect trust or reputation) and allows for inconsistent information among them. The priority ordering is employed both to find the best-rank argument (for or against a query) from a preferred extension, and to improve the efficiency of this computation. While a priority specification as simple as the one presented in this paper is sufficient for many applications, our framework can be easily extended to richer specifications of priorities and aggregation schemes.

We provided an extension to Adjiman et al.'s message-passing algorithm that computes query answers in the presence of inconsistent knowledge and, optionally, a preference ordering over the peers' knowledge. In this paper we used this preference ordering solely to determine whether or not a query was entailed by the system. However, the algorithm can be trivially extended to report the best priority with which a formula is entailed. We proved our algorithm sound and complete with respect to our specification. The complexity of the problem requires the algorithms to be doubly recursive, and thus doubly exponential in runtime. We provided a safe pruning technique and ordering heuristic to alleviate some of this cost. Empirical analysis demonstrated a dramatic improvement in the algorithm's performance using these techniques. This work provides a foundation for many interesting extensions, including one to rule languages for the Semantic Web such as SWRL and RuleML, information integration by ontology translation between peers, and richer priority specification and aggregation schemes. One especially promising direction is our current work on global closed-world reasoning from local closed-world reasoners.

The work presented in this paper touches upon issues from a variety of other areas of research including but not limited to distributed reasoning in AI and databases, reasoning with inconsistent knowledge, and reasoning with priorities and about trust.

In Section 1 we discussed the most significant related work within the area of distributed logical reasoning (i.e., [1, 3, 12]). Also related is the work on distributed logical reasoning with distributed first-order logics [9], and on distributed description logics [19].

Much of the related work on reasoning with inconsistent knowledge concerns inconsistency within a lone KB. Argumentation frameworks (used in this paper to define distributed entailment) provide one of the most plausible semantics for inconsistent knowledge [13]. Preference-based argumentation frameworks only consider preference orderings over arguments, not individual formulas, and compute the best preference ordering over arguments as a function of an ordering over conclusions, and not the other way around [2, 16]. Benferhat et al. compare several entailment relations for prioritized inconsistent KBs [5]. (We exploited their notion of prioritized argued entailment to define distributed reasons in this paper.) Previously, Baral et al. presented an approach to merge multiple inconsistent KBs with associated integrity constraints [4]. In related work, Nebel explored syntax-based approaches to belief revision [17]. Grosof provided a framework for prioritized, inconsistent logic programs with unique consistent answer sets [15]. Priorities have also been used in [8] to achieve extended default reasoning in centralized theories.

There is also significant work on distributed inconsistent databases. For example, Bertossi and Bravo study query answering in P2P data exchange systems that must adhere to data exchange constraints and trust relationships between peers. They develop a semantics of consistency within this setting [6, 7]. Calvanese et al. develop global semantics in first-order and epistemic logics for P2P systems of databases that may be globally inconsistent, and present a corresponding query answering algorithm [11, 10]. Other related work is on P2P information management systems, in which the

peers are again regular databases (e.g., [20, 14]). Again, in all of these approaches, information sources are databases, i.e. they contain facts but no rules. This means that, while the work in this paper is restricted to propositional logic, in many cases their knowledge is expressed in first-order languages. However, distributed database systems are restricted to exchanging facts, whereas we enable a diversity of propositional reasoning in our PQAS.

Lastly, there has been a diversity of work on trust representation and aggregation for distributed systems. Much of the work that is most closely related to this paper concerns the role of trust in peer data exchange systems, as discussed above [7]. In this work, trust relationships exist between peers expressed in binary relations of the form "peer A trusts its own data less/equally much as that of peer B". This differs significantly from the priorities used in our work as such binary relations do not necessarily give rise to a partial or total ordering of the peers by priority or trust. For a more extensive review of trust models see [18].

Acknowledgments

We are very grateful to Gerhard Brewka for feedback on this work in its various stages of development. We also thank Christian Fritz for helpful discussions on technical details of the paper. Michael Gruninger provided his insights on an earlier version of this work. We furthermore thank Philippe Adjiman for providing an early version of his P2PIS code which we did not end up using. Support from the Natural Sciences and Engineering Research Council of Canada (NSERC) is also gratefully acknowledged.

References

[1] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Distributed Reasoning in a Peer-to-Peer Setting: Application to the Semantic Web. *Journal of Artificial Intelligence Research*, 25:269–314, 2006.

[2] L. Amgoud and C. Cayrol. A Reasoning Model Based on the Production of Acceptable Arguments. *AMAI Journal*, 34:197–216, 2002.

[3] E. Amir and S. McIlraith. Partition-Based Logical Reasoning for First-Order and Propositional Theories. *Artificial Intelligence*, 162(1-2):49–88, 2005.

[4] C. Baral, S. Kraus, and J. Minker. Combining Multiple Knowledge Bases. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):208–220, 1991.

[5] S. Benferhat, D. Dubois, and H. Prade. Some Syntactic Approaches to the Handling of Inconsistent Knowledge Bases: a Comparative Study Part 2: the Prioritized Case. In E. Orłowska, editor, *Logic at Work: Essays Dedicated to the Memory of Helen Rasiowa*, volume 24, pages 437–511. Physica-Verlag, 1999.

[6] Leopoldo E. Bertossi and Loreto Bravo. Query Answering in Peer-to-Peer Data Exchange Systems. In *Proceedings of the EDBT workshops, Heraklion, Greece*, pages 476–485, 2004.

[7] Leopoldo E. Bertossi and Loreto Bravo. The Semantics of Consistency and Trust in Peer Data Exchange Systems. In *Proceedings of LPAR 2007*, pages 107–122, 2007.

[8] G. Brewka and T. Eiter. Prioritizing Default Logic. In *Intellectics and Computational Logic*, pages 27–45. Kluwer Academic Publishers, 2000.

[9] Ghidini C. and Serafini L. Distributed First Order Logics. In *Frontiers of Combining Systems 2*, pages 121–139. Research Studies Press Ltd. Baldock, 2000.

[10] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Inconsistency Tolerance in P2P Data Integration: An Epistemic Logic Approach. *Information Systems*, 33(4-5):360–384, 2008.

[11] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical Foundations of Peer-to-peer Data Integration. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 241–251, New York, NY, USA, 2004. ACM.

[12] P. Chatalic, G.-H. Nguyen, and M.-C. Rousset. Reasoning with Inconsistencies in Propositional Peer-to-Peer Inference Systems. In *ECAI 2006*, pages 352–357, 2006.

[13] P. M. Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence*, 77(2):321–358, 1995.

[14] Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. A Distributed Algorithm for Robust Data Sharing and Updates in P2P Database Networks. In *Proceedings of the EDBT P2P&DB workshop, Heraklion, Greece*, pages 446–455, 2004.

[15] B. N. Grosof. Courteous Logic Programs: Prioritized Conflict Handling for Rules. Technical Report RC 20836, IBM, 1997.

[16] S. Kaci and L. van der Torre. Preference Reasoning for Argumentation: Non-monotonicity and Algorithms. In *Proceedings of the International Workshop on Non-Monotonic Reasoning (NMR'06)*, pages 237–243, 2006.

[17] B. Nebel. Syntax-based Approaches to Belief Revision. In P. Gärdenfors, editor, *Belief Revision*, volume 29, pages 52–88. Cambridge University Press, 1992.

[18] Jordi Sabater and Carles Sierra. Review on Computational Trust and Reputation Models. *Artificial Intelligence Review*, 24(1):33–60, 2005.

[19] L. Serafini and A. Tamilin. DRAGO: Distributed Reasoning Architecture for the Semantic Web. Technical Report T04-12-05, ITC-irst, 2004.

[20] I. Zaihrayeu. *Towards Peer-to-Peer Information Management Systems*. PhD thesis, University of Trento, 2006.