COMPLETE DISTRIBUTED CONSEQUENCE FINDING WITH MESSAGE PASSING

Katsumi Inoue, Gauvain Bourgne
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan

Takayuki Okamoto
Graduate School of Science and Technology, Kobe University, 1-1 Rokkodai-cho, Nada-ku, Kobe, Japan

Keywords: Consequence finding, Distributed reasoning, Multi-agent systems.

Abstract: When knowledge is physically distributed, information and knowledge of individual agents may not be collected to one agent because they should not be known to others for security and privacy reasons. We thus assume the situation that individual agents cooperate with each other to find useful information from a distributed system to which they belong, without supposing any master or mediate agent who collects all necessary information from the agents. Then we propose two complete algorithms for distributed consequence finding. The first one extends a technique of theorem proving in partition-based knowledge bases. The second one is a more cooperative method than the first one. We compare these two methods on a sample problem showing that both can improve efficiency over a centrlized approach, and then discuss other related approaches in the literature.

1 INTRODUCTION

There is a growing interest in building large knowledge bases. Dealing with a huge amount of knowledge, two problems can be encountered in real domains. The first case is that knowledge is originally centralized so that one can access the whole knowledge but the knowledge base is too huge to be handled. The second case is that knowledge is *distributed* in several sources so that it is hard or impossible to immediately access the whole or part of knowledge. The former case is studied in the line of research on *parallel* or *partition-based* reasoning. For example, partition-based theorem proving by Amir and McIlraith (Amir et al, 2005) divide a knowledge base into several parts each of which is easier to be handled so that the scalability of a reasoning system is improved.

On the other hand, in the second case we suppose *multi-agent systems* or *peer-to-peer systems* (Adjiman et al, 2006), in which an agent does not want to expose all its information to other agents for security and privacy reasons. Sometimes, it is inherently impossible to tell what other agents want to know and to ask what can be obtained from others. In such a case, each agent must give up gathering all necessary information from other agents, and moreover, no master or mediate agent can be assumed to exist to collect all information from agents. That is, we need to solve the problem with knowledge distributed as it is. In this research, we mainly deal with such distributed knowledge bases, but hope that those algorithms considered for distributed reasoning can be applied to the first case to gain efficiency.

In this work, we consider the problem of *distributed consequence finding*. Consequence finding (Lee, 1967; Inoue, 1992; Marquis, 2000) is a problem to discover an interesting theorem derivable from an axiom set, and is a promising method for problem solving in AI such as query answering (Iwanuma et al, 2002), abduction (Inoue, 1992; Inoue et al, 2009; Nabeshima et al, 2010), induction (Nienhuys-Cheng et al, 1997; Inoue, 2004), diagnosis, planning, recognition and understanding. There are some *complete* procedures for consequence finding in first-order clausal theories (Inoue, 1992; del Val, 1999) and efficient systems have also been developed (Nabeshima et al, 2003; Nabeshima et al, 2010). Our concern here is to design a complete method in the distributed setting, that is, to obtain every consequence that would be derived from the whole knowledge base if it were gathered together. In this paper, we propose two new methods for distributed conse-

Inoue K., Bourgne G. and Okamoto T..
COMPLETE DISTRIBUTED CONSEQUENCE FINDING WITH MESSAGE PASSING.
DOI: 10.5220/0003190001340143
In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (ICAART-2011), pages 134-143
ISBN: 978-989-8425-41-6

quence finding.

The first method here is a generalization of partition-based theorem proving by (Amir et al, 2005) to consequence finding. The whole axiom set is partitioned into multiple sets called *partitions*, each of which can be associated with one agent. In this method, a pair of partitions must be connected with their *communication language*. The connections between partitions constitute a graph, but cycles must be removed so that the graph is transferred to a tree. Consequence finding is firstly performed in the leaves of the connection tree, and its consequences are sent to the parent if they belong to the communication language. This process is repeated until the root. To get a complete procedure in this method, it is important to decide the communication languages between two partitions, so we propose the method to determine them. It should be stressed that, although partition-based theorem proving by (Amir et al, 2005) also uses a consequence finding procedure in each individual reasoning task of an agent, the aim of (Amir et al, 2005) is not consequence finding from the knowledge base but is used for theorem proving tasks.

The second proposed method is a more cooperative one. In this method, we do not presuppose graph structures of agents, but any agent has a chance to communicate with other agents, hence the framework is more dynamic than the first method. Firstly, a new clause is added to an agent $\mathcal{A}_1$, either as a top clause of the given problem or as a newly sent message from other agents, which then triggers consequence finding from that clause with the axioms of $\mathcal{A}_1$. Then, for each such newly derived clause C, if there is a clause D in the axiom set of another agent $\mathcal{A}_2$ such that C and D can be resolved, then C is sent to $\mathcal{A}_2$ and is added there. This process is repeated until no more new clause can be resolved with any clause of any other agent. We will compare these two methods and centralized approaches, and discuss the merits and demerits of both methods. We will also discuss relations with other previously proposed approaches to consequence finding in distributed settings (Inoue et al, 2004; Adjiman et al, 2005; Adjiman et al, 2006).

The rest of this paper is organized as follows. Section 2 reviews the background of consequence finding and SOL resolution. Section 3 proposes partition-based consequence finding. Section 4 proposes a more cooperative algorithm for consequence finding and Section 5 compares the two proposed approaches. Section 6 discusses related work, and Section 7 gives a summary and future work.

2 CONSEQUENCE FINDING

In this section, we review consequence finding from an axiom set and a complete procedure for it. The task of consequence finding is related with many AI reasoning problems, and is indispensable in partition-based theorem proving in Section 3.1 too.

A *clause* is a disjunction of literals. Let C and D be two clauses. C *subsumes* D if there is a substitution θ such that $C\theta \subseteq D$. C *properly subsumes* D if C subsumes D but D does not subsume C. A *clausal theory* is a set of clauses, which is often identified with the *conjunctive normal form* (CNF) formula composed by taking the conjunction of all clauses in it. Let Σ be a clausal theory. $\mu\Sigma$ denotes the set of clauses in Σ not properly subsumed by any clause in Σ. A *consequence* of Σ is a clause entailed by Σ. We denote by $Th(\Sigma)$ the set of all consequences of Σ.

The *consequence finding* problem was first addressed by Lee (Lee, 1967) in the context of the resolution principle, which has the property that the consequences of Σ that are derived by the resolution principle includes $\mu Th(\Sigma)$. To find "interesting" theorems for a given problem, the notion of consequence finding has been extended to the problem to find *characteristic clauses* (Inoue, 1992). Each characteristic clause is constructed over a sub-vocabulary of the representation language called a "production field". Formally, a *production field* $\mathcal{P}$ is a pair, $\langle \mathbf{L}, Cond \rangle$, where $\mathbf{L}$ is a set of literals closed under instantiation, and $Cond$ is a certain condition to be satisfied, e.g., the maximum length of clauses, the maximum depth of terms, etc. When $Cond$ is not specified, $\mathcal{P} = \langle \mathbf{L}, \emptyset \rangle$ is simply denoted as $\langle \mathbf{L} \rangle$. A production field $\mathcal{P}$ is *stable* if, for any two clauses C and D such that C subsumes D, D belongs to $\mathcal{P}$ only if C belongs to $\mathcal{P}$.

A clause C *belongs to* $\mathcal{P} = \langle \mathbf{L}, Cond \rangle$ if every literal in C belongs to $\mathbf{L}$ and C satisfies $Cond$. For a set Σ of clauses, the set of logical consequence of Σ belonging to $\mathcal{P}$ is denoted as $Th_{\mathcal{P}}(\Sigma)$. Then, the *characteristic clauses* of Σ with respect to $\mathcal{P}$ are defined as: $Carc(\Sigma, \mathcal{P}) = \mu Th_{\mathcal{P}}(\Sigma)$. We here exclude any tautology $\neg L \vee L (\equiv True)$ in $Carc(\Sigma, \mathcal{P})$ even when both L and $\neg L$ belong to $\mathcal{P}$. When $\mathcal{P}$ is a stable production field, it holds that the empty clause $\square$ is the unique clause in $Carc(\Sigma, \mathcal{P})$ if and only if Σ is unsatisfiable. This means that theorem proving is a special case of consequence finding. The use of characteristic clauses enables us to characterize various reasoning problems of interest to AI, such as nonmonotonic reasoning, diagnosis, and knowledge compilation as well as abduction and induction. In the propositional case (Marquis, 2000), each characteristic clause of Σ is a *prime implicate* of Σ.

When a new clause C is added to a clausal the-

ory Σ, further consequences are derived due to this new information. Such a new and "interesting" clause is called a "new" characteristic clause. Formally, the *new characteristic clauses* of C with respect to Σ and $\mathcal{P}$ are: $Newcarc(\Sigma, C, \mathcal{P}) = \mu[Th_{\mathcal{P}}(\Sigma \wedge C) - Th(\Sigma)]$.

When a new formula is not a single clause but a clausal theory or a CNF formula $F = C_1 \wedge \cdots \wedge C_m$, where each C_i is a clause, $Newcarc(\Sigma, F, \mathcal{P})$ can be computed as:

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu[\bigwedge_{i=1}^{m} Newcarc(\Sigma_i, C_i, \mathcal{P})], \quad (1)$$

where $\Sigma_1 = \Sigma$, and $\Sigma_{i+1} = \Sigma_i \wedge C_i$, for $i = 1, \ldots, m-1$. This incremental computation can be applied to get the characteristic clauses of Σ with respect to $\mathcal{P}$ as follows.

$$Carc(\Sigma, \mathcal{P}) = Newcarc(True, \Sigma, \mathcal{P}). \quad (2)$$

Several procedures have been developed to compute (new) characteristic clauses. *SOL resolution* (Inoue, 1992) is an extension of the Model Elimination (ME) calculus to which *Skip* operation is introduced along with *Resolve* and *Ancestry* operations. With Skip operation, SOL resolution focuses on deriving only those consequences belonging to the production field $\mathcal{P}$. *SFK resolution* (del Val, 1999) is based on a variant of ordered resolution, which is enhanced with Skip operation for finding characteristic clauses. SOL resolution is complete for finding $Newcarc(\Sigma, C, \mathcal{P})$ by treating an input clause C as the *top clause* and derives those consequences relevant to C directly. SOLAR (SOL for Advanced Reasoning) (Nabeshima et al, 2003; Nabeshima et al, 2010) is a sophisticated deductive reasoning system based on SOL resolution (Inoue, 1992) and the connection tableaux, which avoids producing non-minimal consequences as well as redundant computation using state-of-the-art pruning techniques. Consequence enumeration is a strong point of SOLAR as an abductive procedure because it enables us to compare many different hypotheses (Inoue et al, 2009).

3 PARTITION-BASED CONSEQUENCE FINDING

This section proposes partition-based consequence finding. We start from a review of the basic terminology and the message passing algorithm between partitioned knowledge bases in (Amir et al, 2005), whose basic idea is from Craig's Interpolation Theorem (Craig, 1957; Slagle, 1970).

3.1 Partitions and Message Passing

We suppose the whole axiom set $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$, in which each axiom set $\mathcal{A}_i$ $(i \leq n)$ is called a *partition*. We denote as $S(\mathcal{A}_i)$ the set of (non-logical) symbols appearing in $\mathcal{A}_i$. A *graph induced from the partitions* $\bigcup_{i \leq n} \mathcal{A}_i$ is a graph $G = (V, E, l)$ such that (i) the set V of nodes are the same as the partitions, that is, $i \in V$ iff the partition $\mathcal{A}_i$ exists; (ii) the set E of edges are constructed as $E = \{(i, j) \mid S(\mathcal{A}_i) \cap S(\mathcal{A}_j) \neq \emptyset\}$, that is, the edge $(i, j) \in E$ iff there is a common symbol between $\mathcal{A}_i$ and $\mathcal{A}_j$; and (iii) the mapping l determines the label $l(i, j)$ of each edge (i, j) called the *communication language* between the partitions $\mathcal{A}_i$ and $\mathcal{A}_j$. In partition-based theorem proving by (Amir et al, 2005), $l(i, j)$ is initially set to the *common language* of $\mathcal{A}_i$ and $\mathcal{A}_j$, which is $C(i, j) = S(\mathcal{A}_i) \cap S(\mathcal{A}_j)$. The communication language $l(i, j)$ is then updated by adding symbols from some other partitions when cycles are broken (Algorithm 3.2). In Section 3.3, $l(i, j)$ is further extended by including the language for consequence finding.

Given the partitions $\bigcup_{i \leq n} \mathcal{A}_i$ and its induced graph $G = (V, E, l)$, we now consider the query Q to be proved in the partition $\mathcal{A}_k$ $(k \leq n)$. Given a set S of non-logical symbols, the set of formulas constructed from the symbols in S is denoted as $\mathcal{L}(S)$.

Definition 3.1. For two nodes $i, k \in V$, the length of a shortest path between i and k is denoted as $dist(i, k)$. Given k, we define $i \prec_k j$ if $dist(i, k) < dist(j, k)$. When k is clear from the context, we simply denote $i \prec j$ instead of $i \prec_k j$. For a node $i \in V$, a node $j \in V$ such that $(i, j) \in E$ and $j \prec_k i$ is called a *parent* of i (with respect to $\prec_k$). In the ordering $\prec_k$, the node k is called the *root* (with respect to $\prec_k$), and a node i that is not a parent of any node is called a *leaf* (with respect to $\prec_k$).

Algorithm 3.1 (Message Passing)**.** (Amir et al, 2005)

1. Determine $\prec_k$ according to Definition 3.1.
2. Perform consequence finding in each $\mathcal{A}_i$ in parallel. If $\mathcal{A}_k \models Q$, then return YES.
3. For every $i, j \in V$ such that j is the parent of i, if there is a consequence φ of the partition $\mathcal{A}_i$ such that $\varphi \in \mathcal{L}(l(i, j))$, then add φ to the axiom set $\mathcal{A}_j$.
4. Repeat Steps 2 to 3 until no more new consequence is found.

Algorithm 3.1 works well for theorem proving at $\mathcal{A}_k$ when the induced graph is a tree. However, if there is a cycle, we need to break it to transform the graph to a tree.

Algorithm 3.2 (Cycle Cut)**.** (Amir et al, 2005)

1. Find a shortest cycle $v_1, \ldots, v_c (= v_1)$ $(v_i \in V)$ in G. If there is no cycle, return G.
2. Select a such that $a < c$ and $\Sigma_{j<c, j\neq a} \mid l(v_j, v_{j+1}) \cup l(v_a, v_{a+1}) \mid$ is smallest.
3. For every $j < c$, $j \neq a$, let $l(v_j, v_{j+1}) := l(v_j, v_{j+1}) \cup l(v_a, v_{a+1})$.
4. Put $E := E \setminus \{(v_a, v_{a+1})\}$ and $l(v_a, v_{a+1}) := \emptyset$, then go to Step 1.

When there are multiple shortest cycles, common edges should be removed. But if there is no common edge, edges are removed so that the sum of the sizes of communication languages becomes the smallest. It is important to decide the order to remove edges although any ordering results in a translation to a tree. Cycle Cut Algorithm 3.2 is designed to minimize the total size of the communication languages.

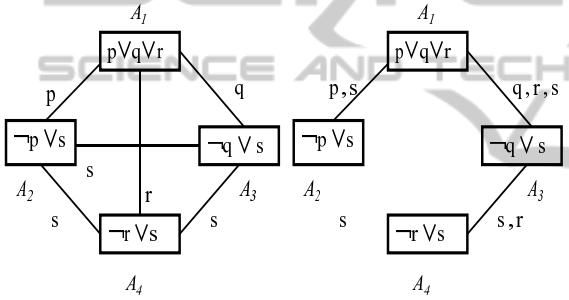


Figure 1: Translation of a cyclic graph to a tree (Amir et al, 2005).

Figure 1 shows an example of cycle cut. The left figure is translated to the right figure. Firstly, the shortest cycle (1,3), (3,4), (4,1) is considered, and then the edge (4,1) is deleted. The communication language of (4,1) is then added to those of (1,3) and (3,4). Next, from the cycle (1,3), (3,2), (2,1), the edge (3,2) is removed, and s is added to $l(1,3)$ and $l(2,1)$. Then, the cycle (1,3), (3,4), (4,2), (2,1) is taken, and the edge (4,2) is removed from it, but s is already in $l(3,4)$ and $l(4,2)$. Now Algorithm 3.1 is applied; $\neg p \vee s$ is sent from $\mathcal{A}_2$ to $\mathcal{A}_1$, deducing $q \vee r \vee s$ (as the resolvent of $\neg p \vee s$ and $p \vee q \vee r$), which is then sent from $\mathcal{A}_1$ to $\mathcal{A}_3$, deducing $r \vee s$ (as the resolvent of $q \vee r \vee s$ and $\neg q \vee s$), which is sent from $\mathcal{A}_3$ to $\mathcal{A}_4$. Finally, the conclusion s is obtained at $\mathcal{A}_4$.

Theorem 3.1. (Amir et al, 2005). *Suppose an axiom set and its partitions $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ and a formula $Q \in \mathcal{L}(\mathcal{A}_k)$ $(k \leq n)$. If the consequence finding procedure in each partition is sound and complete, applying Algorithm 3.2 and then Algorithm 3.1 returns YES iff $\mathcal{A} \models Q$.*

Partition-based theorem proving of (Amir et al, 2005) cannot be directly applied to consequence finding problems for $Q \notin \mathcal{L}(\mathcal{A}_k)$ although (Amir et al, 2005, Section 2.3) briefly mentions how to apply their MP algorithm to such a query constructed from languages in different partitions (a more detailed discussion will be given later in Section 3.3). Hence, we will extend the partition-based reasoning framework to a complete method for distributed consequence finding.

3.2 Example

We now show an example to see that the partition-based theorem proving method cannot be directly applied to consequence finding. The problem is to find means to withdraw money from one's bank account. The intended solution is that one must have either a cash card or a bankbook, which is represented as $card \vee bankbook$. The knowledge base of this problem consists of the following clauses.

- $\neg holiday \vee closed$ (The bank is closed on holidays.)
- $\neg weekday \vee open$ (The bank is open on weekdays.)
- $holiday \vee weekday$ (Any day is either a holiday or a week day.)
- $\neg need\_money \vee \neg open \vee ATM \vee counter$ (If one needs money and the bank is open, then (s)he goes to an ATM or a counter of the bank.)
- $\neg need\_money \vee \neg closed \vee ATM$ (If one needs money and the bank is closed, then (s)he goes to an ATM.)
- $\neg ATM \vee card \vee \neg get\_money$ (One cannot get money if (s)he does not have a cash card at an ATM.)
- $\neg counter \vee bankbook \vee \neg get\_money$ (One cannot get money if (s)he does not have a bankbook at a counter.)
- Input facts: $need\_money$ (One needs money.)
- Input facts: get_money (One gets money.)

Here we assume that the partitions are constructed as in Fig. 2, in which clauses are distributed in a scattered way. Algorithm 3.2 removes the edge (1,3) and then adds ATM to the labels of other edges. However, the clause $card \vee bankbook$ cannot be deduced by Message Passing Algorithm 3.1: since $l(1,2)$ and $l(1,3)$ do not contain $card$, the clause (1) cannot be resolved with any clause in other partitions. In fact, it is necessary to resolve all clauses (1) to (7).

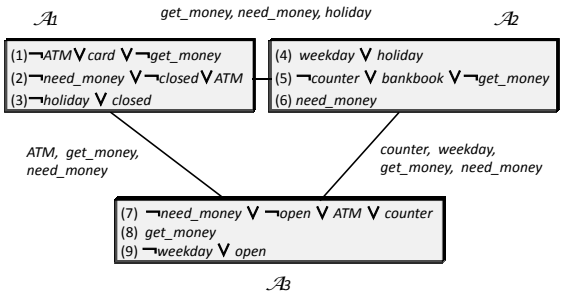


Figure 2: Partitions for "Getting Money".

3.3 Partition-based Consequence Finding

We here propose a new method to construct the communication language so that Message Passing Algorithm can be made complete for consequence finding.

Suppose the whole axiom set and its partitions $\mathcal{A} = \bigcup_{i \le n} \mathcal{A}_i$. Recall that the common language of $\mathcal{A}_i$ and $\mathcal{A}_j$ is $C(i,j) = S(\mathcal{A}_i) \cap S(\mathcal{A}_j)$ $(i, j \le n \quad i \neq j)$. Here, we construct the communication language $l(i,j)$ between $\mathcal{A}_i$ and $\mathcal{A}_j$ for consequence finding by extending $C(i,j)$. Let $\mathcal{P} = \langle \mathbf{L} \rangle$ be the given production field. By adding the literals appearing in $\mathbf{L}$ to the common language, each communication language in the case of trees is defined as

$$l(i,j) = C(i,j) \cup S(\mathbf{L}). \tag{3}$$

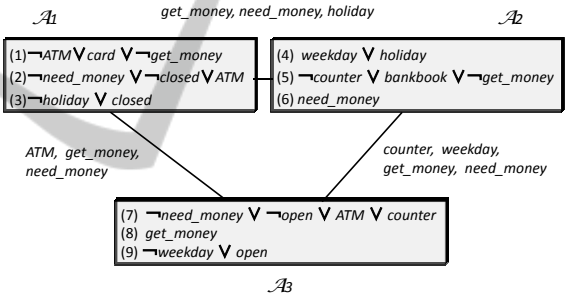
When there are cycles in the graph G, the final communication language is set after all cycles are cut using Algorithm 3.2. For example, suppose that an edge (s,r) is removed from a cycle. Then, the communication language of an edge (i,j) $(\neq (s,r))$ in the cycle is defined in the same way as before:

$$l(i,j) = C(i,j) \cup l(s,r) \cup S(\mathbf{L}). \tag{4}$$

Two remarks are noted here. Firstly, in (3) and (4), the polarity of each literal in $\mathbf{L}$ from the production field $\mathcal{P}$ are lost within the symbols $S(\mathbf{L})$. Although this does not harm soundness and completeness of distributed consequence finding, there is some redundancy in communication. Then, unlike the case of common language $C(i,j)$, we can keep the polarity of each literal in $\mathbf{L}$ in any $l(i,j)$ so that unnecessary clauses possessing literals that do not belong to $\mathcal{P}$ are not communicated between partitions. Second, when $C(i,j) = \emptyset$ the edge (i,j) does not exist in the graph G. In this case, $l(i,j)$ need not be updated as $S(\mathbf{L})$ using (3) and actually the edge is kept unnecessary. In fact, if we could add the literals from the production field to those non-existent edges in G, then the resulting graph G' would become strongly connected. By applying Cycle Cut Algorithm 3.2 to G', the minimal way is to cut those added edges again. However, other edges already have the literals $S(\mathbf{L})$ so their communication languages do not change. Hence, we do not have to reconsider non-existent edges in G.

Algorithm 3.3 (Partition-based Consequence Finding)**.**

1. If there is a cycle in the induced graph G, select some $k \le n$ and apply Cycle Cut Algorithm 3.2 to G and transform it to a tree.
2. Determine the communication language between all pairs of partitions. For each leaf partition $\mathcal{A}_i$, do the following.
3. If $\mathcal{A}_i$ is the root partition, let $\mathcal{P}_i$ be the original production field $\mathcal{P} = \langle \mathbf{L} \rangle$. Otherwise, let j be the parent of i, and define the production field of $\mathcal{A}_i$ as $\mathcal{P}_i = \langle l(i,j)^{\pm} \rangle$, where $l(i,j)^{\pm}$ is the set of literals constructed from $l(i,j)$. Perform consequence finding in $\mathcal{A}_i$ with the production field $\mathcal{P}_i$, and let $Cn_i := Carc(\mathcal{A}_i, \mathcal{P}_i)$. Output each characteristic clause $C \in Cn_i$ if C belongs to the original production field $\mathcal{P} = \langle \mathbf{L} \rangle$.



For the example in Section 3.2, applying Cycle Cut and the decision method of the communication language results in Fig. 3. By this way, the clause (1) consists of the symbols in $l(1,2)$, then can be resolved with other clauses in $\mathcal{A}_2$. Applying Algorithm 3.3, the intended consequence $card \vee bankbook$ can be obtained.

Figure 3: Updating Communication Languages.

Termination of Algorithm 3.3 is guaranteed under some finiteness conditions. For this, (1) if there is

a finite number of cycles in the induced graphs, the maximum depth of a tree is finite after applying Algorithm 3.2, and (2) if there are no recursive theories in each partition, consequence finding in the partition produces a finite number of characteristic clauses. The second condition is satisfied if ground consequences are only produced and there are no function symbols in the language.

The correctness of a distributed consequence finding algorithm A is defined as follows. Suppose the whole knowledge base $\mathcal{A}$ and a production field $\mathcal{P}$. A is *sound* if any clause derived by A is a logical consequence of $\mathcal{A}$ and belongs to $\mathcal{P}$. A is *complete* if it holds for any partitioning of A that: for any clause C belonging to $Th_{\mathcal{P}}(\mathcal{A})$, there is a clause D derived by A such that D subsumes C.

Theorem 3.2. (Soundness and Completeness of Partition-based Consequence Finding). *Suppose an axiom set and its partitions $\mathcal{A} = \bigcup_{i \le n} \mathcal{A}_i$, its induced graph $G = (V, E, l)$, and a stable production field $\mathcal{P} = \langle \mathbf{L} \rangle$. We assume that every partition $\mathcal{A}_i$ has a sound and complete algorithm for consequence finding. Then, Algorithm 3.3 is sound and complete for distributed consequence finding.*

Proof. Any clause derived by Algorithm 3.3 refers to a subset of $\mathcal{A}$ and belongs to $\mathcal{P}$. Then, soundness follows from the monotonicity of first-order logic. Completeness can be proved by induction on the length of any clause $C \in Th_{\mathcal{P}}(\mathcal{A})$. When $|C| = 1$, let $\mathcal{A}_k$ be a partition of $\mathcal{A}$ such that $C \in \mathcal{L}(\mathcal{A}_k)$. Then, by Theorem 3.1, a clause D subsuming C can be derived by Algorithm 3.3, which works in the same way as Algorithm 3.1. Suppose that completeness holds for $|C| \le m$, and we prove the case of $|C| = m + 1$. Let $C = C' \vee L$, where $|C'| = m$ and L is a literal. Let $\mathcal{A}'$ be $\mathcal{A} \cup \{\neg L\}$. Since C' belongs to $\mathcal{P}$ and $C \in Th_{\mathcal{P}}(\mathcal{A})$, $C' \in Th_{\mathcal{P}}(\mathcal{A}')$ holds. Then, assume a partition $\mathcal{A}' = \bigcup_{i \le n} \mathcal{A}'_i$ where $\mathcal{A}'_j = \mathcal{A}_j$ for $j \neq k$ and $\mathcal{A}'_k = \mathcal{A}_k \cup \{\neg L\}$ for some $k \le n$. By induction hypothesis, a clause D' subsuming C' can be derived from $\mathcal{A}'$ at $\mathcal{A}'_k$ by Algorithm 3.3. In fact, if D' is derived at some $\mathcal{A}'_j$ $(j \neq k)$, then it can be sent to $\mathcal{A}'_k$ because D' belongs to $\mathcal{P}$ and hence $D' \in \mathcal{L}(l(j,k))$. We now construct a distributed proof of a clause D subsuming C from $\mathcal{A}$ by adding L to C' appearing in the distributed proof of D' from $\mathcal{A}'$. This is possible because $L \in \mathcal{L}(l(i,j))$ for any $i, j \in V$ by the constructions (3) and (4). Hence, D can be derived at $\mathcal{A}_k$ by Algorithm 3.3. □

Algorithm 3.3 can be seen as a simple extension of partition-based theorem proving by Amir and McIlraith (Amir et al, 2005) since the communication languages are extended to include the literals from the production field. However, this small change is essential for consequence finding. For theorem proving, Amir and McIlraith (Amir et al, 2005, Section 2.3) have mentioned how to deal with a query Q that comprises symbols drawn from multiple partitions. For this, a new partition $\mathcal{A}_Q$ is added with the language $S(\mathcal{A}_Q) = S(Q)$ and $\mathcal{A}_Q$ consists of the clausal form of $\neg Q$. Following addition of this new partition, Cycle Cut must be run on the new graph, and then refutation is performed at $\mathcal{A}_Q$. This method, however, cannot be elegantly applied for consequence finding in general since we do not know the exact theorems or even the possible candidates of theorems to be found in consequence finding. Of course, we can consider the production field $\mathcal{P} = \langle \mathbf{L} \rangle$ for restricted consequence finding. But even with a small $\mathcal{P}$, say $\mathcal{L} = \{a, b, c\}$, to find all consequences with theorem proving we need to query for a, b, c, then possibly $a \vee b$, $a \vee c$ and $b \vee c$, and eventually $a \vee b \vee c$ (though querying the last clause $a \vee b \vee c$ and checking all possible proofs would also work but have high complexity too). Alternatively, considering the new partition $\mathcal{A}_{\mathcal{P}}$ with the language $S(\mathbf{L})$ makes the graph more tightly connected and cyclic. Applying Cycle Cut would then modify the communication language of an existing edge to include $S(\mathbf{L})$, which has a similar effect as the equation (3).

Another important change from the MP algorithm by Amir and McIlraith (Amir et al, 2005) is to use the production field $\mathcal{P}_i = \langle l(i,j)^{\pm} \rangle$ in Step 3 of Algorithm 3.3 for consequence finding. This limits the computations that need to be done and thus improves efficiency. The use of production fields also enables us to emulate *default reasoning* by adding each default literal in a production field to be skipped (Inoue et al, 2004; Inoue et al, 2006). Hence, our algorithm can be extended to *partition-based default reasoning*.

4 COOPERATIVE CONSEQUENCE FINDING

Partition-based distributed consequence finding is particularly useful when we have a large knowledge base that should be divided to easily handle each piece of knowledge. However, the algorithm can also be applied to naturally distributed knowledge-based systems in which each theory of an agent grows individually so that multiple agents may have the same knowledge and information simultaneously. Although such possessed knowledge is considered to be redundant in partition-based theories, there is no problem in *decentralized*, *multi-agent* and *peer-to-peer* systems.

In such naturally distributed systems, one problem would be to break cycles in the induced graph because no agent should know an optimal way to minimize the cost of cutting cycles (although we could devise a decentralized version of Cycle Cut algorithm).

In this section, we thus consider an alternative approach to distributed consequence finding that is suitable for such *autonomous* agent systems. The new method is more *cooperative* than the previous one in the sense that agents are always seeking other agents who can accept new consequences for further inference. In this method, we do not presuppose network structures of agents, but any agent can have a chance to communicate with other agents. As the language and knowledge of each agent evolves through interactions, this framework is more *dynamic* than the first method. Since the method is not partition-based, we do not call each distributed component a partition, but call it an *agent* in this section.

Algorithm 4.1 (Cooperative Consequence Finding)**.**

1. Suppose a set I of *input clauses* is given. This consists of a query or a goal clause in the case of query answering or abduction as well as any clause input to the whole system $\mathcal{A}$. Let $\mathcal{A}_i$ be a newly created agent whose axiom set is I. Let $\mathcal{P}_{\mathcal{A}} = \langle Lit_{\mathcal{A}} \rangle$ be the (stable) production field, where $Lit_{\mathcal{A}}$ is the set of all literals in the language of $\mathcal{A}$. Perform consequence finding in $\mathcal{A}_i$, and let $N := Carc(\mathcal{A}_i, \mathcal{P}_{\mathcal{A}})$.
2. For each clause $C \in N$, decide an agent $\mathcal{A}_j$ to which C is sent from $\mathcal{A}_i$. Put $i := j$.
3. In $\mathcal{A}_i$, consequence finding is performed by SOL resolution with the top clause C. Let $N := Newcarc(\mathcal{A}_i, C, \mathcal{P}_{\mathcal{A}})$. Put $\mathcal{A}_i := \mu(\mathcal{A}_i \cup N)$.
4. Repeat Steps 2 to 3 until no more new characteristic clause is derived.

Algorithm 4.1 repeats the process of (a) and (b): (a) new consequences obtained in an agent are sent to others, and (b) then they trigger consequence finding in those agents. An advantage of this method is that we only need to compute *new characteristic clauses* $Newcarc(\mathcal{A}_i, C, \mathcal{P}_{\mathcal{A}})$ in Step 3. In fact, computation of new characteristic clauses is easier than computation of the whole characteristic clauses by SOL resolution. The whole characteristic clauses are still obtained by accumulating the new ones with subsumption checking by simulating (1) and (2) in Step 3. Note that computation of $Carc(\mathcal{A}_i, \mathcal{P}_{\mathcal{A}})$ in Step 1 is not necessary when I is a single clause or contains no complimentary literals.

In Step 2 of Algorithm 4.1, we assume that any agent can decide to which agent each clause $C \in N$ should be sent. One such implementation is to associate with each agent $\mathcal{A}_i$ the set of predicates with their polarities appearing in the axiom set. This set must be updated each time a new characteristic clause is computed in the agent. Then, it becomes easier to find a literal that is complementary to a literal in C in other agents. One can also use the current communication language $l(i, j)$ between two agents: if $C \cap l(i, j) \neq \emptyset$ holds, then C can be sent from $\mathcal{A}_i$ to $\mathcal{A}_j$. Note here that we do not need to break cycles, but $l(i, j)$ needs to be updated whenever the axioms are updated. In another way, a blackboard architecture like (Ciampolini et al, 2003) can be considered as a place to store new characteristic clauses deduced by agents. An agent should check whether it has a clause which can be resolved with a new characteristic clause.

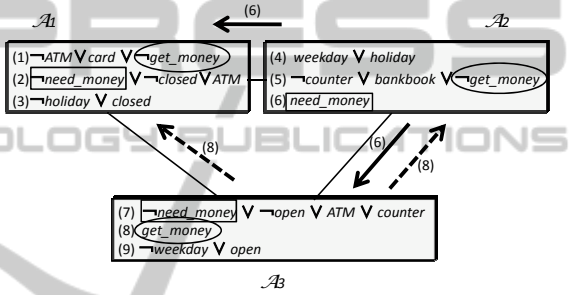


Figure 4: First Message Passing in Cooperative Consequence Finding (arrows indicates message passing, labelled by the number of the sent clause, and the complementary pair of literals causing the sending is also emphasized).

Note that implementation of Steps 2 to 4 can be parallelized provided that synchronization is properly done. The first message passing for unit clauses is illustrated in Fig. 4 for the example of Section 3.2.

Termination of Algorithm 4.1 is similar to the case of Algorithm 3.3. For the correctness of Algorithm 4.1, the following theorem holds.

Theorem 4.1. (Soundness and Completeness of Cooperative Consequence Finding). *Suppose an axiom set and its partitions $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$. We assume that every agent $\mathcal{A}_i$ has a sound and complete algorithm for consequence finding. Then, Algorithm 4.1 is sound and complete for distributed consequence finding of $Newcarc(\mathcal{A}, I, \mathcal{P}_{\mathcal{A}})$.*

Proof. Soundness is proved in the same way as Theorem 3.2. For completeness, $Newcarc(\mathcal{A}, I, \mathcal{A}_{\mathcal{P}})$ can be decomposed into multiple clause-by-clause *Newcarc* operations by (1). Since we use the production field $\mathcal{P}_{\mathcal{A}}$ in which all literals appearing in $\mathcal{A}$ can be skipped, all Skip operations in any SOL deduction from the whole $\mathcal{A}$ are also applied by Algorithm 4.1. On the

other hand, all Resolve operations of SOL deductions can be simulated by sending resolving clauses to other agents. Ancestor resolution in SOL deductions can also be done by sending back to previous agents. Thus, any SOL deduction can be simulated in a distributed setting by 4.1. □

5 COMPARING APPROACHES

We here compare the two proposed methods for distributed consequence finding. We first note that the two methods are not designed to compute the same consequences as long as Theorems 3.2 and 4.1 are concerned. Given an axiom set $\mathcal{A}$, partition-based consequence finding computes $Carc(\mathcal{A},\mathcal{P})$ belonging to a given production field $\mathcal{P}$ in Theorem 3.2, while cooperative consequence finding computes $Newcarc(\mathcal{A},I,\mathcal{P}_{\mathcal{A}})$ for a given set of inputs I in Theorem 4.1. We could extend both methods to deal with any case by considering the same conditions for them. However, the current conditions are natural in both methods. The partition-based approach is based on Interpolation Theorem (Craig, 1957), which refers to the set of consequences of an axiom set of one partition, yet a language restriction can be used effectively. On the other hand, the cooperative approach is more dynamic and reflective so that ramification from the new input is propagated to other agents, but the language restriction is not easily set since every agent could be related to any other. Nevertheless, an obvious merit of the cooperative method is that we do not need to break cycles.

Then, we consider the situation that Algorithms 3.3 and 4.1 can be fairly compared. In the example in Section 3.2, we have two input clauses from the world as I and the production field can be set to $\mathcal{P}_{\mathcal{A}}$. Here we compare the number of resolution steps as well as the number of sent messages in solving this example. For comparison, we also show the results of the centralized approach. As the consequence finding system in each partition or agent, SOLAR (Nabeshima et al, 2003) is used. The ordering or partitions in the partition-based consequence finding is set to $\mathcal{A}_1 \rightarrow \mathcal{A}_2 \rightarrow \mathcal{A}_3$.

Table 1 (a) shows the number of resolution steps in each method. Comparing two distributed methods with the centralized one, the total number of resolution steps becomes fewer in both methods. This is because (i) the partition-based method restricts clauses sent to its parent to those constructed with the communication language between those partitions and (ii) the cooperative method performs consequence finding in each agent only with top clauses sent from other agents. Comparing the partition-based method with the cooperative one, we see that the latter is more uniformly distributed in its load balance. The former, partition-based one, has the property that the number of inference steps becomes increasing from leaves to the root. In a leaf partition, there are fewer axioms and thus the number of resolution steps is fewer too. Sending messages from descendants to ancestors, more and more clauses are gathered so that more resolution steps become necessary at later stages. Then, the bottom partition mostly collects clauses from the descendants. As a result, the load balance of the partition-based method is not averaged. On the other hand, the cooperative method send newly obtained consequences to all agents that have resolvable clauses, which prevents overloads in particular agents. In the example, agents efficiently cooperate with each other to get the final consequence.

Table 1: Comparison of three methods for "Getting Money"(Okamoto et al, 2005).

Approach	(a) # resolution steps				(b) # sent messages			
	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	total	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	total
Centralized	–	–	–	659	–	–	–	0
Partition-based	19	51	461	531	3	5	0	8
Cooperative	27	62	63	152	5	7	8	20

Table 1 (b) shows a comparison between the partition-based and the cooperative methods on the number of messages sent to other partitions. Each message corresponds to one clause in this comparison. In the partition-based method, all cycles in the graph are broken, and message passing is done from leaves to the root in only one way. Any message is thus sent to the parent only once. On the other hand, the cooperative method sends messages to all agents possessing resolvable clauses, which increases the number of communications between agents. If there are many agents containing clauses that can be resolved with a consequence of some agent, the number of messages to be sent in the cooperative method becomes larger than that of the partition-based method.

6 RELATED WORK

Consequence finding has been investigated in a distributed setting (Inoue et al, 2004; Amir et al, 2005; Adjiman et al, 2005). Inoue and Iwanuma (Inoue et al, 2004) consider a multi-agent framework which performs *speculative computation* under incomplete communication environments. This is a master-slave style multi-agent system, in which a master agent asks queries to slave agents in problem solving and proceeds computation with default answers when answers from slave agents are delayed. Spec-

ulative computation is implemented with SOL resolution with the conditional answer method to update agents' beliefs according to situation changes. On the other hand, distributed consequence finding in this paper does not assume any master agent to control the whole system. Amir and McIlraith (Amir et al, 2005) propose distributed theorem proving to improve the efficiency of theorem proving for structured theories. Their message-passing algorithm reasons over these theories using consequence-finding, and our first (partition-based) approach in this paper also uses it. As already stated in Section 3.3, the main difference between (Amir et al, 2005) and our partition-based approach is that the goal of the former is theorem proving while our goal is consequence finding. Another difference is that (Amir et al, 2005) considers how to partition a problem to minimize the intersection of the languages, while we suppose the situation that such optimal partitioning cannot be applied because of inherent distribution of knowledge and impossibility to collect all information to one place. This last observation directed us to the second, cooperative approach to distributed consequence finding, which is quite different from the first one.

The *peer-to-peer* (P2P) consequence finding system proposed by Adjiman *et al.* (Adjiman et al, 2005; Adjiman et al, 2006) is perhaps closest to our work. Their method is related to our both first (partition-based) and second (cooperative) approaches to consequence finding. (Adjiman et al, 2005) composes an *acquaintance graph* from the peers using information of shared symbols, which is similar to a graph induced from the partitions in our first approach. The difference is that (Adjiman et al, 2005) does not break cycles in a graph while we do. Also, (Adjiman et al, 2005) performs case splitting in goal-oriented reasoning of a peer P_1 by sending to other peer P_2 only those subgoals contained in the shared symbols between P_1 and P_2, then the new consequences of P_2 are returned to P_1, which is then composed in P_1 by replacing the subgoal. Combining the results derived from the subgoals often would result in a huge combination of clauses when the length of the goal is long, yet (Adjiman et al, 2006) analyzes the scalability of large P2P systems. On the other hand, we send the clause itself without splitting and no re-collection is made. Our second approach can be regarded as a dynamic version of the first approach, in which messages are sent whenever new clauses are derived, and there is no presupposed network structures of agents. Such dynamic aspects are not seen in the P2P setting. Another difference is that (Adjiman et al, 2005) can only deal with propositional knowledge bases, while SOL resolution and SOLAR in our paper can be used for consequence finding in first-order clausal theories.

Although not in the context of consequence finding, abduction has also been considered in a distributed setting. Since abduction in clausal theories can be implemented with consequence finding, such work is somehow related to distributed consequence finding. Greco (Greco, 2007) considers how to build *joint explanations* from multiple agents in a P2P setting like (Adjiman et al, 2005), but incorporates preference handling to have an agreement between agents. By extending a blackboard architecture of (Ciampolini et al, 2003), Ma *et al.* (Ma et al, 2008) address distribution of abductive logic programming agents by allowing agents to enter and exit proofs done by other agents. Those works do not use consequence finding, and communication between agents are fully guaranteed. More recently, Bourgne *et al.* (Bourgne et al, 2010) propose the *learner-critique* approach in which the role of each agent dynamically changes between a generator and a tester of hypotheses when each agent never knows which symbols are shared with other agents. In our methods, all agents work uniformly as a reflective inference system that derives consequences upon input of new formulas, although shared symbols are assumed to be known to both agents. Fisher (Fisher, 2000) shows that certain forms of negotiation can be characterized by distributed theorem proving in which agents act as theorem-proving components. Analogously, distributed consequence finding might contribute to extended types of negotiation between agents.

In this work, we have focused on distributed reasoning systems in which a clause set is partitioned and the common symbols between partitions are associated with links. In contrast, there is another formalization of distribution in which variables or symbols are partitioned and clauses containing symbols from different partitions are associated with links between those partitions. The former formalization is called *clause-set partitioned distribution*, while the latter is called *variable-set partitioned distribution*. Most works on *distributed constraint satisfaction problems* (DCSP) are based on the latter formalization (Yokoo et al, 1998; Hirayama et al, 2005). These two formalizations can be converted into each other in the propositional case (cf., (Dechter et al, 1989)), yet the effect of the latter is unknown for consequence finding while the former often occurs in real cases.

7 CONCLUSIONS

In this paper, we have proposed the two complete approaches for distributed consequence finding. The

first one extends the method of partition-based theorem proving in a suitable way, and the second one is a more cooperative method for inherently distributed systems. This paper rather focuses on completeness of inference systems, and both approaches have merits and demerits. Partition-based approaches can utilize communication languages to realize restricted consequence finding between the partitions, while the cooperative approach does not need Cycle Cut algorithm. On the negative side, it is important to determine an appropriate ordering in the partition-based method, while the number of messages sent between agents tends to become larger in the cooperative approach. We could consider a third approach by inheriting the merits of both approaches, such that each agent is autonomous and cooperates each other like the cooperative approach, yet each consequence finder incorporates production fields and communication languages between agents to enhance efficiency. Consideration of such a new approach is left as an important future work. Another future task includes more experiments with large distributed knowledge bases by refining details of two algorithms and by changing topological properties of agent links. More comparison with P2P consequence finding (Adjiman et al, 2006) is also necessary.

REFERENCES

Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.-C. and Simon, L. (2005). Scalability study of peer-to-peer consequence finding. In *Proc. IJCAI-05*, pp.351–356.

Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.-C. and Simon, L. (2006). Distributed reasoning in a peer-to-peer setting: Application to the semantic web. In *J. Artif. Intell. Res.*, 25:269–314.

Amir, A. and McIlraith, S. (2005). Partition-based logical reasoning for first-order and propositional theories. In *Artif. Intell.*, 162:49–88.

Bourgne, G., Maudet, N., and Inoue, K. (2010). Abduction of distributed theories through local interactions. In *Proc. ECAI'10*, 901–906.

Ciampolini, A., Lamma, E., Mello, P., Toni, F., and Torroni, P. (2003). Cooperation and competition in ALIAS: A logic framework for agents that negotiate. *Ann. Math. Artif. Intell.*, 37(1-2):65–91.

Craig, W. (1957). Linear reasoning: A new form of the Herbrand-Gentzen theorem. *J. Symbolic Logic*, 22:250–268.

Dechter, R. and Pearl, J. (1989). Tree clustering for constraint networks. *Artif. Intell.*, 38:353–366.

del Val, A. (1999). A new method for consequence finding and compilation in restricted languages. In *Proc. AAAI-99*, pp.259–264.

Fisher, M. (2000) Characterizing simple negotiation as distributed agent-based theorem-proving—a preliminary report. in: *Proc. 4th ICMAS*, pp. 127–134.

Greco, G. (2007). Solving abduction by computing joint explanations. *Ann. Math. Art. Intel.*, 50(1–2):143–194.

Hirayama, K. and Yokoo, M. (2005). The distributed breakout algorithms. *Artif. Intell.*, 161:89–115.

Inoue, K. (1992). Linear resolution for consequence finding. *Artif. Intell*, 56:301–353.

Inoue, K. (2004). Induction as consequence finding. *Machine Learning*, 55:109–135.

Inoue, K. and Iwanuma, K. (2004). Speculative computation through consequence-finding in multi-agent environments, *Ann. Math. Artif. Intell.*, 42(1–3):255–291.

Inoue, K., Iwanuma, K. and Nabeshima, H. (2006). Consequence finding and computing answers with defaults. *J. Intell. Inform. Systems*, 26:41–58.

Inoue, K., Sato, T., Ishihata, M., Kameya, Y. and Nabeshima, H. (2009). Evaluating abductive hypotheses using an EM algorithm on BDDs. *IJCAI*, 810–815.

Iwanuma, K. and Inoue, K. (2002). Minimal answer computation and SOL. *JELIA '02*, LNAI 2424, 245–257, Springer.

Lee, C.T. (1967). A completeness theorem and computer program for finding theorems derivable from given axioms. Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA.

Ma, J., Russo, A., Broda, K., and Clark, K. (2008). DARE: A system for distributed abductive reasoning. *AAMAS'08*, 16(3):271–297.

Marquis, P. (2000). Consequence finding algorithms. in: *Handbook for Defeasible Reasoning and Uncertain Management Systems, Vol. 5*, pp.41–145, Kluwer.

Nabeshima, H., Iwanuma, K. and Inoue, K. (2003). SOLAR: A consequence finding system for advanced reasoning. *TABLEAUX*, LNAI 2796, 257–263, Springer.

Nabeshima, H., Iwanuma, K., Inoue, K. and Ray, O. (2010). SOLAR: An automated deduction system for consequence finding. *AI Communic.*, 23(2–3):183–203.

Nienhuys-Cheng, S.-H. and de Wolf, R. (1997). *Foundations of Inductive Logic Programming*. LNAI 1228, Springer.

Okamoto, T., Inoue, K. (2005). Distributed consequence finding with message communication. *IPSJ-SIG Tech. Rep.*, 24:25–30, (in Japanese).

Slagle, J.R. (1970). Interpolation theorems for resolution in lower predicate calculus. *J. ACM*, 17(3):535–542.

Yokoo, M., Durfee, E.H., Ishida, T., and Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Know. & Data Eng.*, 10(5):673–685.