Distributed Interactive Learning in Multi-Agent Systems

Jian Huang and **Adrian R. Pearce**
Department of Computer Science and Software Engineering
NICTA Victoria Laboratory
The University of Melbourne
Victoria 3010, Australia
{*jhua,adrian*}@*csse.unimelb.edu.au*

Abstract

Both explanation-based and inductive learning techniques have proven successful in a variety of distributed domains. However, learning in multi-agent systems does not necessarily involve the participation of other agents directly in the inductive process itself. Instead, many systems frequently employ multiple instances of induction separately, or single-agent learning. In this paper we present a new framework, named the Multi-Agent Inductive Learning System (MAILS), that tightly integrates processes of induction between agents. The MAILS framework combines inverse entailment with an epistemic approach to reasoning about knowledge in a multi-agent setting, facilitating a systematic approach to the sharing of knowledge and invention of predicates when required. The benefits of the new approach are demonstrated for inducing declarative program fragments in a multi-agent distributed programming system.

Introduction

This paper concerns agent learning problems, where induction involves the construction of hypotheses that explain observations relative to the context of a particular domain. Learning that utilises logic programming based on domain knowledge has been extensively researched, including explanation-based learning (EBL) (Dejong & Mooney 1986; Mitchell, Keller, & Kedar-Cabelli 1986) and inductive logic programming (ILP) (Muggleton & Raedt 1994).

Success in EBL and ILP has demonstrated that new knowledge can be acquired systematically using domain (or background) knowledge and training examples (or positive and negative examples). When provided with some background knowledge B and examples E of a concept, an ILP system constructs a hypothesis H that explains E in terms of B, i.e. $B \wedge H \models E$, based on the process of inverting deductive inference (Muggleton 1995).

While it has been recognized that domain knowledge performs an important role in constructing these hypotheses in multi-agent systems (Kazakov & Kudenko 2001), a great deal of work on *multi-agent* learning problems has frequently employed multiple instances of induction separately, as opposed to learning that tightly integrates processes of induction *between* agents. For a survey of multi-agent learning, refer to (Stone & Veloso 2000). In those works, agents do not necessarily require the participation of other agents directly in learning.

According to (Kazakov & Kudenko 2001), the problem of true multi-agent learning has far more complexity than simply having each agent perform localized learning in isolation. Weiß and Dillenbourg clearly identify this problem "interaction does not just serve the purpose of data exchange, but typically is in the spirit of a cooperative, negotiated search for a solution of the learning task (Weiß & Dillenbourg 1999)".

The reason that interaction and cooperation in multi-agent learning is important, is because the crucial knowledge necessary in learning a hypothesis is typically distributed over a number of agents. This gives rise not only to the problem that no individual agent can accomplish the learning task alone any more but also the problem of knowing what background knowledge from each agent is required for constructing the global hypothesis, given that sharing complete knowledge is often not feasible in such environment. Due to the above two constraints, neither of the two extremes of the collaboration scheme would work, i.e. learning in isolation or communicating everything. Therefore, the interaction between agents while performing a learning task needs to be elaborated, such that agents draw together knowledge when necessary. Further, they should draw only the necessary knowledge together.

This paper studies interactive multi-agent learning using an integrated approach, which combines ILP with epistemic reasoning. Our work utilizes the implemented ILP system Aleph (Srinivasan 2001), a successor to Progol, for performing hypothesis construction. Instead of viewing induction as a stand alone technique for learning new concepts, our work adopts a perspective in which the induction process is viewed as an extension to deduction for acquiring missing knowledge needed for reasoning. By adopting this view, it allows partially specified programs to be completed and executed by a multi-agent team. We aim at establishing a general model within which execution and symbolic learning can take place interactively among a number of agents autonomously and seamlessly.

In the following sections, we first provide an overview of inductive learning in multi-agent domain and demonstrate the reason interaction is necessary in such an environment. A formalism is then presented for analyzing agents for multi-agent inductive learning. The subsequent section details the interactive strategy the agents are to adopt in order to collaborate in inductive learning. At the end of the paper, we outline the experimental system testing the proposed model and conclude.

Multi-agent Inductive Learning

The study of knowledge in the literature, as it relates to agents, frequently concerns either factual (Fagin *et al.* 1997) or procedural (Georgeff & Lansky 1986) aspects of knowledge. Singh has termed the former "know-that" and the latter "know-how" (Singh 1999). As people often do not distinguish "knowing a fact" from "knowing how to do something" explicitly, it often depends on the context to be able to tell which meaning is referred to, e.g. "Do you know quick sort?". This distinction is, however, important to us because in multi-agent learning domain, knowledge is no longer only about the state of the world but has a lot to do with actual problem solving. This is because participants not only need to know what to do but also how to do it and who can do it.

Throughout the paper, we follow the notation used in the KARO framework (van Linder, van der Hoek, & Meyer 1998), i.e. we use the operator $K_i\varphi$ for agent i knows the fact φ and $A_i\alpha$ for agent i has the ability to perform the action α. Both these two aspects of knowledge are of concern to us because in multi-agent setting, the agent knowing what to do and the agent knowing how to it may not be the same. It often occurs that one agent's knowledge depends on another agent's knowledge of the state of the world or being able to perform some action.

Consider the following example in a logic programming context:

> Agent i requires a definition for the predicate $min(L, M)$ (for finding the min number in a list). It knows that if it can sort a list (in ascending order), then the first element will be the minimum of the list. However, agent i doesn't know how to sort a list (but it does know what sorting a list means!) so its knowledge about min depends on another agent's knowledge on $sort$. Agent j knows how to generate permutations of a list and how to check the ordering of a list. Given that sorting can be performed by generating permutations and checking if the permutation is ordered, agent j is already capable of performing sorting as long as information can be communicated.

In the above example, only agent i knows what $sort$ means at the start, which can be viewed that agent i knows about the +/- examples of sorted lists. Although agent j knows everything it needs to perform $sort$, it doesn't know there exists such a thing called sorting! Nevertheless, it can induce the definition of $sort$, based on its background knowledge and examples given by agent i. One consequence of this separation of know-that and know-how is that there exist such needs to transfer the concept (i.e. +/- examples) to the agent who is capable of implementing the concept. In our case, agent i needs to transfer the knowledge of the +/- examples to agent j first, as a way of communicating to agent j "this is the concept that I need an implementation of".

With the presence of the aforementioned dependency in knowledge, interaction between agents during learning is a prerequisite for successful learning outcome. Without interaction between the agents, hypotheses would not be successfully induced. Interaction in such an environment can potentially be complicated when dependencies span over a large number of agents and when dependencies are circular. In summary, the following situations may arise and need to be handled:

1. The simplest case is that agent j already has an implementation when agent i asks for it. Therefore, it's just a matter of communication.
2. Alternatively, agent j needs to induce a hypothesis based on the +/- examples received from agent i and its own background knowledge to explain the examples.
3. Furthermore, agent j may require agent k to induce some background knowledge for it first before it can induce the definition required by agent i.
4. Finally, even background knowledge required for inducing the definition may be distributed over different agents.

An example of multi-agent inductive learning is provided in Table 1. In the example there are three agents; each contains some background knowledge and some positive/negative examples. This example will be referred to throughout the rest of the paper.

Formalization of Inductive Learning Agents

In this section, we formally specify the agents among which reasoning, learning, induction and execution of logic programs can take place.

Function symbols, predicate symbols, variables, terms and literals are defined in the usual logic programming sense. Every predicate symbol is an atomic action. Each agent consists of the following four components:

Background Set $\mathbb{B} = \{b_1, \cdots, b_n\}$ Agent's background knowledge consists of predicate definitions in logic programming sense, i.e. it is a collection of horn clauses. We may refer to the background set as the *program* the agent contains. Each b_x in the set corresponds to one horn clause in the program.

Example Set $\mathbb{E} = E^+ \cup E^-$ The set of positive examples $E^+ = \{e_1^+, \cdots, e_n^+\}$ and negative examples $E^- = \{e_1^-, \cdots, e_n^-\}$ are ground literals, or ground horn clauses with no body. They are *facts* or *training data*, in normal machine learning sense, to which a correct hypothesis must conform. For example, e_1^+ = `sort([2,1,3],[1,2,3])` means this is a positive example of $sort$. Similarly, e_1^- = $\neg$ `sort([3,2,1],[1,3,2])` indicates a negative example. Although in logic programming sense, there is no actual difference between programs and examples, we still differentiate them in our model as they are used for very different purposes.

```
                              Agent i
-------------------------------------------------------------------
range(List,Range) :-      min(List,Min) :-      max(List,Max) :-
    min(List,Min),            sort(List,L),         sort(List,L),
    max(List,Max),            first(L,Min).         last(L,Max).
    Range is Max - Min.

last([L],L).                          first([N|Ns],N).
last([N|Ns],L) :- last(Ns,L).

    +                                     -
sort([],[]). sort([0],[0]).           sort([1],[]). sort([1],[2]).
sort([2,1,3],[1,2,3]).                sort([1,2,3,4,5],[5,4,3,2,1]).
sort([1,5,2,4,3],[1,2,3,4,5]).        sort([9,3,1,5,3],[1,3,3,9,5]).

                              Agent j
-------------------------------------------------------------------
permutate([],[]).                       ordered([]).
permutate(List,[First|Perm]) :-         ordered([_]).
    select(First,List,Rest),            ordered([X,Y|Tail]) :-
    permutate(Rest,Perm).                   X =< Y, ordered([Y|Tail]).

select(Elem,[Elem|Tail],Tail).
select(Elem,[Head|Tail1],[Head|Tail2]) :-
    select(Elem,Tail1,Tail2).

                              Agent k
-------------------------------------------------------------------
is_set(List) :- \+ repeated(List).

    +                                       -
same([]). same([5]).                    same([1,2]). same([1,1,1,2]).
same([0,0,0]).                          same([5,4,3,2,1]).
same([1,1,1,1,1,1]).                    same([2,3,3,3,3,3]).

repeated([1,1]).                        repeated([]).
repeated([1,2,1]).                      repeated([1]).
repeated([3,3,3]).                      repeated([1,2,3]).
repeated([3,5,7,5]).                    repeated([1,3,5,2,4]).
```

Table 1: Example to illustrate multi-agent inductive learning with three agents, each contains some background knowledge, positive and negative examples.

Capability Set $\mathbb{C} = C^+ \cup C^-$ The set of capabilities $C^+ = \{c_1^+, \cdots, c_n^+\}$ and incapabilities $C^- = \{c_1^-, \cdots, c_n^-\}$ contains modal representation corresponding to what atomic actions (with respect to the background set) an agent is able (or unable) to perform while abstracting away the detailed definition. (We tend to use the word 'capability' to mean both capability and incapability unless explicitly differentiated.) Each c_x^+ is in the form of $A_i\alpha$ and each c_x^- is in the form of $\neg A_i\alpha$. $A_i\alpha$ holds iff agent i is able to perform the atomic action α.

Take the example in Table 1, for agent i to be able to perform min, it clearly depends on some agent (including itself) being able to perform $sort$ and $first$. In another words, the agent can perform min if all the atomic actions that min can possibly depend on can be performed. An agent's capability can be systematically reasoned by performing a slightly modified linear resolution technique as specified later in the paper. For this naive example, it is clear that the agent is able to perform the action $first$, but not the action min as it can't perform the action $sort$ that min depends on. In summary, the capability set for agent i, is:

$$\mathbb{C} = \{A_i first, A_i last, \neg A_i sort, \neg A_i min, \neg A_i max, \neg A_i range\}$$

If, however, it is already known by agent i that another agent j is able to perform the action $sort$, the capability set for agent i will instead be:

$$\mathbb{C} = \{A_i first, A_i last, A_j sort, A_i min, A_i max, A_i range\}$$

In another word, agents increase their capabilities by taking other agents' capabilities into consideration, which is what would be expected for a team consisting of multiple agents. Note that the capability set of one agent can contain capabilities of other agents in the team iff the agent is aware of them, and agents do become aware of other agents' capabilities over time, which will be detailed later in the paper.

Knowledge Base $\mathbb{K} = K^P \cup K^E \cup K^C$ An agent's knowledge base contains everything that the agent knows, including: K^P: the semantics of its background set; K^E: the examples from its examples set, $e \in \mathbb{E}$ iff $K_i e \in K^E$; K^C: its capabilities as well as the capabilities of other agents, $c \in \mathbb{C}$ iff $K_i c \in K^C$.

That is saying, agents know what their background knowledge implies, they know the positive and negative instances of various concepts and they know the capabilities of themselves and of others that they are aware of. They can perform epistemic reasoning based on their knowledge set.

The semantics of the agent's background set capture the meaning of its background knowledge by viewing it as capability dependencies. For example, if agent i has the background knowledge mentioned in Table 1, its semantics are $A_i range \leftarrow A_i min \wedge A_i max$ etc. And by 'agent knows the semantics of its programs', we mean agent i's knowledge contains $K_i(A_i range \leftarrow A_i min \wedge A_i max)$. That is, the agent knows that being able to do min and max leads it to be able to do $range$.

An agent's capability set is populated initially by deriving the action dependency clauses from its program semantics and then performing a resolution technique similar to classical linear input resolution on the action dependency clauses. The resolution process attempts to establish a refutation for each atomic action in the agent's program. If successful, the agent is capable of performing that action, incapable otherwise.

Take the program from Table 1. It is illustrated below how agent i reasons about whether it is capable of performing the action $range$. The following capability dependency is directly obtained from its program:

$$\begin{aligned} A_i range &\leftarrow A_x min \wedge A_y max &(1)\\ A_i min &\leftarrow A_x sort \wedge A_y first &(2)\\ A_i max &\leftarrow A_x sort \wedge A_y last &(3)\\ A_i first && (4)\\ A_i last && (5) \end{aligned}$$

The above clauses are then resolved with the goal clause $\neg A_i range$ as illustrated in Figure 1. The A_i operator is omitted to keep the resolution clear. As can be seen, the resolution without the A_i modality is exactly the SLD resolution for classical propositional logic.

We fail to obtain a refutation for the goal and thus conclude that the agent is unable to perform $range$. What is more, the resolution indicates that the reason for not being able to resolve $\neg sort$ to get the empty clause is due to not having a definition for $sort$. The agent can then proactively seek out the definition for $sort$ in the team.

Apart from reasoning about its abilities and inabilities, an agent can also reason based on its knowledge about other

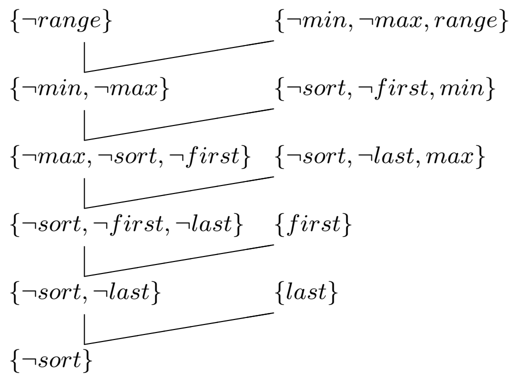
$\{\neg range\}$
$\{\neg min, \neg max, range\}$
$\{\neg min, \neg max\}$
$\{\neg sort, \neg first, min\}$
$\{\neg max, \neg sort, \neg first\}$
$\{\neg sort, \neg last, max\}$
$\{\neg sort, \neg first, \neg last\}$
$\{first\}$
$\{\neg sort, \neg last\}$
$\{last\}$
$\{\neg sort\}$

Figure 1: Resolution for reasoning about capability

agents' knowledge status, e.g. "He knows that I know that he knows ..."; and whether, when, who and what to communicate.

For example, if agent i asks agent k the definition of $sort$, agent k can deduce that agent i doesn't know $sort$, i.e. $K_k(\neg A_i sort)$. Furthermore, if agent k knows that agent j knows about $sort$ it can communicate this information to agent i.

Over time, an agent's capability set expands as new predicates are induced by itself and other agents. Here we examine what updates to the sets need to be carried out whenever a new predicate is induced. For example, agent i has a program $\alpha \leftarrow \beta, \gamma$ and requires a definition for β, which has been induced by agent j. The following updates need to be done to agent i's sets:

1. Program Set $\mathbb{P}$: no change.
2. Example Set $\mathbb{E}$: no change.
3. Capability Set $\mathbb{C}$: $\mathbb{C} = \mathbb{C} \cup \{A_j\beta\}$, and the resolution described previously will be run to re-assess its capabilities since new atomic actions which it wasn't previously capable of might now be enabled.
4. Knowledge Set $\mathbb{K}$: $\mathbb{K} = \mathbb{K} \cup \{C_{ij}A_j\beta\} \cup \{K_i c_1\} \cdots \cup \{K_i c_n\}$, where c_i are the new capability after updating the capability set. $\{C_{ij}A_j\beta\}$ indicates it is now common knowledge between agent i and agent j that $A_j\beta$.

Similar updates need to be performed for agent j:

1. Program Set $\mathbb{P}$: $\mathbb{P} = \mathbb{P} \cup \{p'\}$, where p' is the definition induced for β.
2. Example Set $\mathbb{E}$: $\mathbb{E} = \mathbb{E} \cup E'$, where E' is the examples for β that was obtained from agent i.
3. Capability Set $\mathbb{C}$: $\mathbb{C} = \mathbb{C} \cup \{A_j\beta\}$. No resolution needs to be performed, since there can't possibly be any incapabilities previously existing that depends on the action β.
4. Knowledge Set $\mathbb{K}$: $\mathbb{K} = \mathbb{K} \cup \{C_{ij}A_j\beta\}$, as before.

Inductive Learning through Interaction

Based on the formalism detailed in the previous section, we illustrate how a team of agents interact while collaborating

```
DEDUCE($Goal$)
 0: initialize goal list with $Goal$
 1: while goal list is not empty do
 1:     pick the first goal $g$
 2:     if $g$ is defined then
 3:         if $g$ is resolvable then
 3:             replace $g$ with its body
 4:         else
 5:             return FAIL
 6:         end if
 7:     else
 7:         INDUCE($g, example(g)$)
 8:         if $g$ is induced then
 8:             continue
 9:         else
10:             return FAIL
11:         end if
12:     end if
13: end while
14: return SUCCEED
```

Table 2: Algorithm for deduction

in inductive learning. As mentioned in the introduction, in order to utilize the full power of inductive learning in a multi-agent system, we view induction as part of deduction for acquiring missing knowledge to overcome the problem of knowledge being distributed. Here we present a generic algorithm to be followed by the agents that facilitates systematic acquisition of missing knowledge and allows execution based on incomplete knowledge.

Table 2 outlines the algorithm for the deduction process. When asked to prove a goal, an agent keeps replacing the first goal in the goal list by its body, just like the Prolog interpreter, unless it encounters a goal for which it fails to find a corresponding definition. Then it invokes the induction process and either returns back with the induced definition or fails if the induction fails.

Table 3 outlines the algorithm for induction. During induction, an agent first induces any undefined predicates before using them as background knowledge. It then calls an ILP system to induce the definition of the predicate based on its own background knowledge. Upon failing, it asks the rest of the team, one by one to, induce the definition and the induction process will be called recursively until either the definition is obtained or all agents have tried and failed.

If all agents have tried and failed, this indicates that some background knowledge is missing and the hypothesis does not exist. This is because the induction algorithm constructs the hypothesis in a bottom up fashion, i.e. it tries to resolve all undefined predicates that the hypothesis may possible rely on first, and uses them as background knowledge to induce more background knowledge recursively. This thus guarantees that the hypothesis will be found so long as all necessary low level ingredients exist in the team.

Given that the algorithm doesn't require the communication of background knowledge (only examples), one exceptional situation is when background knowledge needed to induce a definition is distributed over multiple agents. We

```
INDUCE($Pred, Example$)
 1: for all background predicate $b$ do
 2:     if $b$ does not depend on $Pred$ then
 3:         if $b$ is not defined then
 3:             INDUCE($b, example(b)$)
 4:         end if
 4:         $Background \leftarrow Background \cup b$
 5:     end if
 6: end for
 6: ILP($Pred, Background, Example$)
 7: if $Pred$ is induced then
 8:     return SUCCEED
 9: else
10:     for each agent $i$ in the team do
10:         ASK($i, Pred, Example$)
11:         if $Pred$ is induced then
12:             return SUCCEED
13:         end if
14:     end for
14:     invoke distributed ILP process
15:     if $Pred$ is induced then
16:         return SUCCEED
17:     else
18:         return FAIL
19:     end if
20: end if
```

Table 3: Algorithm for induction

consider it an open solution to this that the ILP be run in a distributed fashion to allow different agents to participate in the process.

In a team consisting of many agents, the interaction strategy can potentially involve as many agents as necessary to learn a piece of missing knowledge. To illustrate this, consider a sample scenario as follows:

Imagine agent k from Table 1 is asked to prove `same([2,2,2,2])`, if a list contains the same number 2, and is given a number of positive and negative examples. As one known definition for *same* is `same(L) :- range(L,0)` and agent i knows the definition for *range*, which in turn relies on j's definition of *sort*, ideally we expect the agents to figure this out through interaction. This can indeed be achieved using the algorithms described above as illustrated in Figure 2.

The interactive model is also resource conservative in a sense that it does not require an unrestricted exchange of knowledge between agents nor does it require agents to obtain full awareness of other agents' knowledge status. Instead, every agent starts by attempting to induce the missing definition alone and passes the task to another if it fails. And at any time only limited information, the examples, need to be communicated between agents.

At the current stage of the research, ILP is viewed as a centralized process and is thus treated as a black box process equipped by each agent to run given background knowledge and examples. Although this simplifies the problem so that we are abstracted away from the internal details of the ILP system, it also imposes further restrictions to us such that whenever the ILP system is called, the background knowledge has to be congregated into one agent already. Research has been done to distribute hypothesis searching for ILP, but only allowing partitioning of the example space. To our knowledge, there hasn't been a concurrent version of the ILP system that allows background knowledge to be distributed.

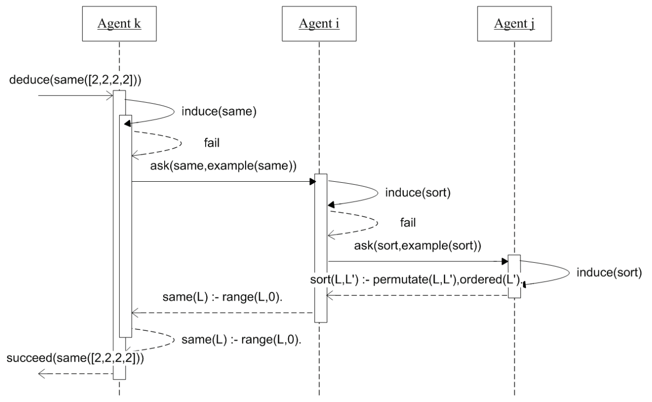

Figure 2: Agent interaction for sample scenario

Consider again the example in Table 1. If agent k has the background knowledge $ordered$ instead of agent j, none of agent i, j or k would be able to accomplish the task of inducing the definition for $sort$ alone without the involvement of the others. This requires agents j and k to combine their background knowledge in the inductive learning process. So far, we have avoided the discussion of such situations but the model we have proposed does handle this type of knowledge distribution. Therefore decentralized execution of the ILP process, allowing hypothesis searching involving distributed background knowledge, is potentially important towards a more complete model for multi-agent inductive learning.

Experimental Results

In order to test the proposed formalism and algorithms, we have implemented a system in Prolog to simulate a multi-agent environment and agent interaction. The existence of agents is simulated by having a background file, an example file and a knowledge file for each agent in the team. The communication between agents and information passing are simulated by having the interpreter invoke and pass the information to the relevant agents. The context switching from the execution of one agent to another is simulated by loading and unloading background knowledge of relevant agent into the interpreter. The user may issue a query to a specified agent through the interpreter. The agent would deduce based on its local background knowledge and, if necessary, induce missing knowledge in order to answer the query and may potentially invoke other agents in the team. The induced knowledge will be added into the background file of the inducer and will be used as background knowledge in the future. In some cases, agents other than the inducer may be aware of the knowledge being induced, it would record this fact in its knowledge file and further queries about this induced knowledge would be directed to the relevant agent.

The ILP system Aleph has been used in experiments for

```
?- deduce(i,range([2,5,8],Range)).
---------------- Deduction Attempt ----------------
Agent: i, Goal: range([2, 5, 8], _G378)
---------------------------------------------------
Current Goals: [pmin([2, 5, 8], _G1020), pmax([2, 5, 8], _G1023),
_G378 is _G1023-_G1020]
Current Goals: [psort([2, 5, 8], _G1069), first(_G1069, _G1020),
pmax([2, 5, 8], _G1023), _G378 is _G1023-_G1020]
---------------- Induction Attempt ----------------
Agent: i, Predicate: psort
Background: [last(_G1461, _G1462), range(_G1452, _G1453),
first(_G1443, _G1444), pmin(_G1434, _G1435), pmax(_G1425, _G1426)]
---------------- Induction Attempt ----------------
Agent: j, Predicate: psort
Background: [permutation(_G1789, _G1790), ordered(_G1781)]
---------------- Induction Success ----------------
Predicate: psort
Hypothesis: [(psort([_G188|_G189], [_G191|_G192]):-
permutation([_G188|_G189], [_G191|_G192]),
ordered([_G191|_G192])), psort([], [])]
---------------- Deduction Attempt ----------------
Agent: j, Goal: psort([2, 5, 8], _G100)
---------------------------------------------------
Current Goals: [permutation([2, 5, 8], [_G1665|_G1666]),
ordered([_G1665|_G1666])]
...
Current Goals: []
---------------- Deduction Success ----------------
Current Goals: [first([2, 5, 8], _G62), pmax([2, 5, 8], _G65),
_G22 is _G65-_G62]
Current Goals: [pmax([2, 5, 8], _G65), _G22 is _G65-2]
Current Goals: [psort([2, 5, 8], _G2601), last(_G2601, _G65),
_G22 is _G65-2]
...
Current Goals: [last([8], _G65), _G22 is _G65-2]
Current Goals: [_G22 is 8-2]
Current Goals: []
---------------- Deduction Success ----------------

Range = 6
```

Table 4: Output showing the execution trace when agent i is asked to deduce `range([2,5,8],Range)`.

inducing predicate definition given background knowledge and examples. We have observed that although Aleph is the state-of-art ILP system available, it *does* require some fine-tuning of the settings and this make it difficult to guarantee success given the right inputs when being integrated into the whole system.

The query `deduce(i,range([2,5,8],Range))` is executed as an example and a trace is shown in Table 4, formatted slightly for presentation. In the sample, we have asked agent i to deduce `range([2,5,8],Range)`. Agent i keeps replacing goals until the point where it fails to find a definition for $psort$. It first attempts to induce the definition for it based on its own background knowledge, but fails. It then turns to agent j for the definition, who successfully induces it and gets back to agent i. Agent i then proceeds with the rest of the goals. Note here that since agent i has acquired the knowledge that agent j knows the definition for $psort$, it directs the current goal to agent j to deduce. Sure enough, agent j has the definition and returns success for the goal `psort([2,5,8], _G100)`. Agent i keeps going with the rest, and turns to agent j again the second time for the query regarding $psort$. After receiving the result, agent i finally finishes replacing all goals in the list and successfully determines that `Range = 6`.

Conclusion

In this paper, we present an inductive learning model that takes advantage of background knowledge of other agents in the team while learning new concepts. Although early work by Davies (Davies 1993) has looked at the problem of learning new concepts among collaborative agents, his work did not concern the type of ILP developed later (Muggleton 1995). In contrast, our work develops multi-agent induction based on integrating (Srinivasan 2001) and (Fagin *et al.* 1997) towards the aims of true mutli-agent learning, as identified by (Kazakov & Kudenko 2001). In addition, our model guarantees to find hypotheses as long as background knowledge exists and is also conservative of resources.

References

Davies, W. 1993. *ANIMALS A Distributed Heterogeneous Multi-Agent Machine Learning System*. Ph.D. Dissertation, University of Aberdeen.

Dejong, G., and Mooney, R. J. 1986. Explanation-based learning: An alternative view. *Machine Learning* 1(2):145–176.

Fagin, R.; Moses, Y.; Halpern, J. Y.; and Vardi, M. Y. 1997. Knowledge-based programs. *Distributed Computing* 10(4):199–225.

Georgeff, M., and Lansky, A. 1986. Procedural knowledge. *Proceedings of the IEEE (Special Issue on Knowledge Representation)* 74:1383–1398.

Kazakov, D., and Kudenko, D. 2001. Machine learning and inductive logic programming for multi-agent systems. In Luck, M.; Marik, V.; and Stepankova, O., eds., *Multi-Agent Systems and Applications*, volume 2086. Springer. 246–270.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation based learning A unifying view. *Machine Learning* 1(1):47–80.

Muggleton, S., and Raedt, L. D. 1994. Inductive logic programming: Theory and methods. *Journal of Logic Programming* 19/20:629–679.

Muggleton, S. 1995. Inverse entailment and progol. *New Generation Computing, Special issue on Inductive Logic Programming* 13:245–286.

Singh, M. P. 1999. Know-how. In Rao, A. S., and Wooldridge, M. J., eds., *Foundations of Rational Agency*, Applied Logic Series. Kluwer. 105–132.

Srinivasan, A. 2001. Extracting context-sensitive models in inductive logic programming. *Machine Learning* 44(3):301–324.

Stone, P., and Veloso, M. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots* 8(3):345–383.

van Linder, B.; van der Hoek, W.; and Meyer, J.-J. C. 1998. Formalising abilities and opportunities of agents. *Fundamenta Informaticae* 34(1-2):53–101.

Weiß, G., and Dillenbourg, P. 1999. What is 'multi' in multiagent learning? In Dillenbourg, P., ed., *Collaborative learning. Cognitive and computational approaches*. Pergamon Press. 64–80.