

Toward Inductive Logic Programming for Collaborative Problem Solving

Jian Huang and Adrian R. Pearce

NICTA Victoria Research Laboratory

Department of Computer Science and Software Engineering

The University of Melbourne, Victoria, Australia

{jhua,adrian}@csse.unimelb.edu.au

Abstract—In this paper, we tackle learning in distributed systems and the fact that learning does not necessarily involve the participation of agents directly in the inductive process itself. Instead, many systems frequently employ multiple instances of induction separately. The paper’s main contribution is a new approach that tightly integrates processes of induction between distributed agents, based on inductive logic programming techniques, for a wider class of problem solving tasks. The approach combines inverse entailment with an epistemic approach to reasoning about knowledge, facilitating a systematic approach to the sharing of knowledge and invention of predicates only when required. We illustrate the approach for learning declarative program fragments and for a well-known path planning problem and compare results empirically to (multiple instances of) single agent-based induction over varying distributions of data. Given a chosen path planning algorithm, our algorithm enables agents to combine their local knowledge in an effective way to avoid central control while significantly reducing communication costs.

I. INTRODUCTION

While it has been widely recognized that the distribution of knowledge performs an important role in constructing hypotheses in distributed systems, 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 [1]. In those works, agents do not necessarily require the participation of other agents directly in learning.

However, according to [2], the problem of *true* multi-agent learning has far more complexity than simply having each agent perform localized induction in isolation. In the spirit of Weiß and Dillenbourg [3], we view induction as a cooperative negotiated search for a solution to the learning task.

The reason that interaction and cooperation during 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, 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 prior work published by the authors [4], inductive logic programming (ILP) has been proposed as an effective approach that facilitates interaction and collaboration during

learning among agents. Within the framework, agents are equipped with background knowledge, expressed as logic programs, and they reason about what they know, based on their collaborative engagement in learning tasks, through communicating positive and negative examples (based on the prior success and failure of goals from the perspective of each agent).

In this paper, we build on this work towards an integrated approach that combines deductive and inductive logic programming and epistemic reasoning within the same framework, for a wider range of problem solving tasks. As a first step, we illustrate the approach using two examples. First, the approach is demonstrated for learning declarative program fragments in a distributed programming setting. Second, we use the distributed path planning problem where path information (such as reachability and cost) is distributed over different agents and empirically show that the approach shows promise for avoiding central control and reducing communication costs involved.

In this paper, we focus on learning that utilizes inductive logic programming (ILP), based on the inverse resolution technique [5]. We follow in the tradition of prior work on context sensitive models and decision-theoretic ILP for efficiently constraining the search and finding optimal models [6]. For our knowledge-based needs of distribution, this involves scoring hypothesis during induction: which model is the optimal choice for the current context relative to each agents viewpoint and goals?

An important aspect of our approach is that it seeks to integrate processes of both deductive and inductive inferencing during problem solving. Our research views the synergy of combining both processes as an effective way of acquiring new knowledge while performing reasoning; such that an agent performing induction may have a number of deductive subroutines that can be used at its discretion, and vice versa.

In section II we define the collaborative ILP problem and formalize our inductive agents in section III. We then illustrate our approach using two examples: inducing logic program (fragments) in section IV and (empirically) using distributed path planning by deduction and induction in section V.

II. THE COLLABORATIVE ILP PROBLEM

The process of inductive logic programming is often defined as such: when provided with some background knowledge B and examples E of a concept, an ILP algorithm constructs a hypothesis H such that $B \wedge H \models E$. In multi-agent systems, ILP involves generating hypotheses using the *collective* background knowledge. More formally, the process of ILP in multi-agent systems can be defined as follows.

Definition 1: The collaborative ILP problem is defined by the set of agents \mathbb{A} ; the background knowledge \mathbb{B}_i , where $i \in \mathbb{A}$, for each agent i ; and the set of positive and negative examples \mathbb{E}_i^+ and \mathbb{E}_i^- for each agent i . Further, $\mathbb{B} = \bigcup_{i \in \mathbb{A}} \mathbb{B}_i$ is the set of all background knowledge and $\mathbb{E}^+ = \bigcup_{i \in \mathbb{A}} \mathbb{E}_i^+$ and $\mathbb{E}^- = \bigcup_{i \in \mathbb{A}} \mathbb{E}_i^-$ are the sets of all positive/negative examples. Then collaborative ILP can be viewed as the process of collaboratively generating the hypothesis H such that the following conditions hold:

- 1) Prior Satisfiability: $\mathbb{B} \wedge \mathbb{E}^- \not\models \square$
- 2) Posterior Satisfiability: $\mathbb{B} \wedge H \wedge \mathbb{E}^- \not\models \square$
- 3) Prior Necessity: $\mathbb{B} \not\models \mathbb{E}^+$
- 4) Posterior Sufficiency: $\mathbb{B} \wedge H \models \mathbb{E}^+$, and
- 5) $\neg \exists i \in \mathbb{A}$ such that $\mathbb{B}_i \wedge H \models \mathbb{E}^+$

The first four conditions are adapted from ILP in a single agent setting [5] and are generalized to allow hypothesis generation over the agents' total background knowledge. The fifth condition asserts that there is no individual agent who is able to induce the hypothesis based solely on its own background knowledge.

Due to constraints associated with resource bounded multi-agent systems, bringing distributed background knowledge together into one agent and execute a centralized ILP algorithm is often infeasible in practice. Therefore, inductively generating the hypothesis in multi-agent systems relies heavily on careful exchange of information between agents during learning, for which we believe epistemic reasoning plays an important role.

III. FORMALIZATION OF INDUCTIVE AGENTS

In order to integrate agent interaction, inductive logic programming and epistemic reasoning, a generic model of inductive agents has been developed by the authors in [4] based on four fundamental constructs: background knowledge, examples, capabilities and a knowledge base. Background knowledge and examples facilitate the execution of inductive logic programming algorithms. Capabilities and the knowledge base enable the agents to perform epistemic reasoning, inspired by the KARO framework [7]. Importantly, we distinguish background knowledge commonly defined in ILP from the knowledge base which facilitates epistemic reasoning.

a) *Background Set* $\mathbb{B} = \{b_1, \dots, b_n\}$: An agent's background set contains the background knowledge, or domain knowledge, that it uses for inducing hypothesis.

b) *Example Set* $\mathbb{E} = E^+ \cup E^-$: An agent's example set contains the positive and negative examples that it has gathered from previous experience or from other sources. The example

set, together with background set, can be used as training examples when performing inductive logic programming to learn new concepts.

c) *Capability Set* $\mathbb{C} = C^+ \cup C^-$: An agent's capability set keeps track of the capabilities (and incapacities) of itself and of other agents, in the form of $A_i\alpha$.

d) *Knowledge Base* $\mathbb{K} = K^P \cup K^E \cup K^C$: Finally, 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. Knowing what they know allows the agents to perform reasoning based on their knowledge status and capabilities, e.g. an agent may perform reasoning to find out that it doesn't have the capability of performing some action and can proactively seek out the capability among other agents in the system.

An agent can reason about its abilities (and inabilities) as well as its knowledge status about the world and about other 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 is not capable of performing *sort*, i.e. $K_k(\neg A_i \text{sort})$. Furthermore, if agent k knows that agent j knows about *sort* it can communicate this information to agent i . Please refer to [4] for how reasoning about capability and knowledge can be performed under the model.

IV. EXAMPLE: INDUCING PROGRAM FRAGMENTS

Based on the formalism detailed in the previous section, we illustrate how this formalism enables a team of agents to collaborate while learning missing program fragments. For this reason, we extend deduction to allow missing knowledge to be induced to overcome the problem of knowledge being distributed. Here we present a generic algorithm to be executed by the agents that facilitates systematic acquisition of missing knowledge among a team of agents and allows program execution based on incomplete knowledge.

The deduction process proceeds just like any Prolog interpreter, such that when asked to prove a goal, an agent keeps replacing the first goal in the list by its body 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.

Fig. 1 outlines the algorithm for the induction process. 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, through the communication of positive and negative examples, 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. The induction algorithm constructs the hypothesis in a bottom up fashion, i.e. it tries to resolve all undefined predicates that the hypothesis may

```

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
4:       INDUCE(b, example(b))
5:     end if
6:     Background ← Background ∪ {b}
7:   end if
8: end for
9: ILP(Pred, Background, Example)
10: if Pred is induced then
11:   return SUCCEED
12: else
13:   for each agent i in the team do
14:     ASK(i, Pred, Example)
15:     if Pred is induced then
16:       return SUCCEED
17:     end if
18:   end for
19:   if Pred is induced then
20:     return SUCCEED
21:   else
22:     return FAIL
23:   end if
24: end if

```

Fig. 1. Algorithm for inducing missing predicates involving collaboration among multiple agents.

possibly rely on first, and uses them as background knowledge to induce more background knowledge recursively.

In a team consisting of many agents, the proposed 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 i has the background knowledge defined as follows:

```

range(List, Range) :-
  min(List, Min), max(List, Max), Range is Max - Min.

min(List, Min) :- sort(List, L), first(L, Min).
max(List, Max) :- sort(List, L), last(L, Max).

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

first([N|Ns], N).

```

Further, agent j has the definition for *permutation* and *ordered* in its background knowledge. Agent k is asked to prove $\text{same}([2, 2, 2, 2])$, if a list contains the same number, and is given a number of positive and negative examples. As one known definition for *same* is $\text{same}(L) :- \text{range}(L, 0)$ and agent i knows the definition for *range*, which in turn relies on j 's definition of *sort* defined by *permutation* and *ordered*, ideally we expect the agents to figure this out through interaction by themselves. This can indeed be achieved using the algorithms described above and has been verified through our implementation. The interactions between agent i , j and k are illustrated in Fig. 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

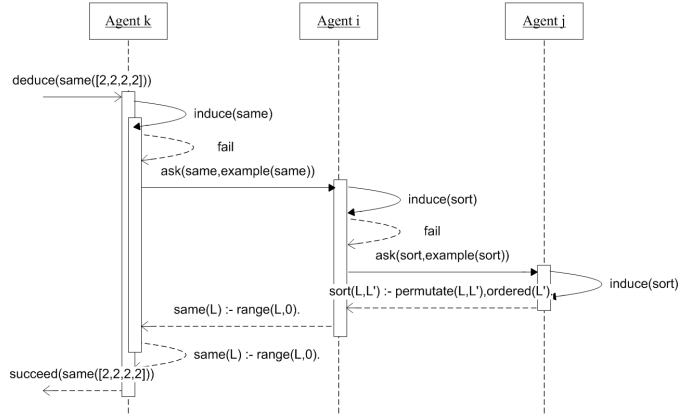


Fig. 2. Agent interaction for the sample scenario.

alone and invokes others if it fails. In this way, the global hypothesis is learnt through a collaborative approach and by drawing knowledge together from a team of agents. And at any time only limited information, the examples, need to be communicated between agents.

However, in this illustrative example, the proposed algorithm only guarantees the hypothesis be found if an agent has all the necessary background knowledge before an ILP process is run. That is, the algorithm doesn't handle the case when background knowledge needed to induce a definition is distributed over multiple agents.

The proposed formalism and algorithm have been verified by our implementation integrating Prolog with the ILP system Aleph [6]. In our implementation, the user just issues a query to any agent in the team and it is entirely up to the agents to work out the missing fragments to answer the query. When asked to prove a goal, the agents would deduce based on its local background knowledge and, if necessary, induce missing knowledge in order to proceed with the theorem proving and may potentially invoke other agents in the team. For details of the implementation and results, refer to [4].

V. EXAMPLE: DISTRIBUTED PATH PLANNING

In this section, we use the distributed path planning problem to show empirically the advantages of the approach in reducing communication costs while allowing agents to collaboratively learn through interaction. As in logic programming domain, capital letters are used to denote free variables and small letters bound. The term $\text{reachable}(a, b)$ stands for "b is reachable from a". The term $\text{link}(a, b)$ means "there exists a link from a to b". The link term can also include extra arguments containing information about the link (such as cost) but for illustration we stick to the two-argument form.

We assume each car is equipped with some background knowledge such as the following:

- 1) $\text{link}(A, B) \rightarrow \text{reachable}(A, B)$
- 2) $\text{reachable}(A, B) \wedge \text{reachable}(B, C) \rightarrow \text{reachable}(A, C)$

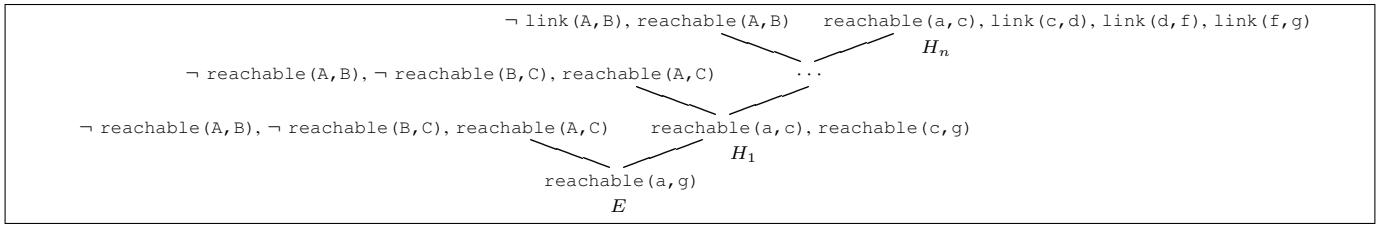


Fig. 3. Inverse resolution: background knowledge used is shown on the left branches and hypotheses generated on the right.

The first background knowledge simply captures the meaning that if there exists a link from A to B, then it's reachable from A to B. Likewise, the second clause says that if it's reachable from A to B and it's reachable from B to C, then it's reachable from A to C. A car also records links that it has gone through previously in history in its knowledge base, in the form of $\text{link}(A, B)$. In another word, if a car could perform deductive reasoning, it would infer based on the background knowledge and the knowledge it has gained historically, given any query in the form $\text{reachable}(A, B)$, whether it is reachable from one location to another.

On the other hand, under an inductive framework, given the same query $\text{reachable}(A, B)$, the car can come up with hypotheses H , together with its background knowledge B , to explain this query, i.e. $H \wedge B \models \text{reachable}(A, B)$. Viewed from a slightly different perspective, the inductive technique can be used to uncover a path from one location to another since if a path does exist, the inductive process will at some stage generate a hypothesis containing only link terms, which effectively corresponds to the actual path from A to B.

Equipped with an inductive process and some simple background knowledge as the above, our approach allows a car to issue a query in the form $\text{reachable}(A, B)$ while seeking a path from A to B. When engaged in answering this query, the cars being consulted attempt to induce, by performing ILP technique such as inverse resolution, a series of hypotheses to explain this query based on their own background knowledge.

Fig. 3 illustrates the process of inverse resolution while generating hypotheses to explain the query $\text{reachable}(a, g)$ given a link history $\text{link}(c, d)$, $\text{link}(c, e)$, $\text{link}(d, e)$, $\text{link}(d, f)$, $\text{link}(f, g)$. The background knowledge used at each step is shown on the left and the hypotheses generated along the way are shown on the right branches. The hypothesis $H_n = \text{reachable}(a, c) \wedge \text{link}(c, d) \wedge \text{link}(d, f) \wedge \text{link}(f, g)$ in the example is interpreted as: $\text{reachable}(a, g)$ (the query) is true so long as $\text{reachable}(a, c)$ is true given that $\text{link}(c, d)$, $\text{link}(d, f)$ and $\text{link}(f, g)$ are all known to be true.

Surely, a hypothesis generated by a single agent doesn't always correspond to a path since knowledge of an individual is often incomplete. Just as what happens in the previous example, based on its local knowledge, the car in the example is unable to find out a path from a to g. However, returning a hypothesis such as H_n is more helpful than simply failing in a multi-agent environment and we will show shortly how this *partial* solution can be used during future endeavors to

```

GENHYPO(Query)
1: HypList ← {Query}, HypHistory ← ∅
2: while HypList ≠ ∅ do
3:   Choose hypothesis H from HypList
4:   Generate all subsequent hypotheses HypAll based on H
5:   if HypAll = ∅ then
6:     HypHistory ← {H} ∪ HypHistory
7:   else
8:     HypList ← HypAll ∪ HypList
9:   end if
10: end while
11: return HypHistory

```

Fig. 4. Algorithm for hypothesis generation.

uncover the full path.

A. Hypothesize Paths by Induction

Discovering the partial solution relies on generating the hypotheses in a controlled fashion. The basic algorithm for doing so is sketched out in Fig. 4. The algorithm starts by choosing the first hypothesis H from HypList, initialized to contain only the Query, and applying inverse resolution using H and each background knowledge. The resulting hypotheses, if there is any, is then stored back to the HypList. If no further hypothesis is returned by the inverse resolution process, then take H out of the HypList and store it into HypHistory. The algorithm keeps picking up the next hypothesis in the HypList until it becomes empty. In this way, all possible hypotheses that can be generated starting from the Query itself are explored.

Because every hypothesis, which explains the query, can be the one that contains the solution, they all need to be generated. Therefore, the complexity of the algorithm in the worst case involves expanding starting from Query with each of the background knowledge and unifying with every possible known location until all hypotheses become longer than the total of known path. If l denotes the number of known locations, k denotes the number of known links and b denotes the number of background knowledge, then the complexity of the algorithm in the worst case is $O((l \times b)^k)$. However, because many of the generated hypotheses can be discarded halfway through the search before they become meaningless, the average case complexity is significantly lower.

B. Deduction Directed Search

Inductively generating hypotheses in an uninformed way described above can make search space intractable very quickly. For this reason, the basic algorithm is extended so that it allows

```

GENHYPO(Query)
1: HypList  $\leftarrow$  {Query}, HypHistory  $\leftarrow$   $\emptyset$ 
2: while HypList  $\neq$   $\emptyset$  do
3:   Choose hypothesis  $H$  from HypList
4:   if  $\exists T$  in  $H$  such that DIJKSTRA( $T$ , Path) is true then
5:     Replace  $T$  with Path and store  $H$  into HypHistory
6:   else
7:     Generate all subsequent hypotheses HypAll based on  $H$ 
8:     if HypAll =  $\emptyset$  then
9:       HypHistory  $\leftarrow$  { $H$ }  $\cup$  HypHistory
10:    else
11:      HypList  $\leftarrow$  HypAll  $\cup$  HypList
12:    end if
13:  end if
14: end while
15: return HypHistory

```

Fig. 5. Extended algorithm for hypothesis generation utilizing deductive shortest path algorithm.

a path searching subroutine, such as Dijkstra’s algorithm, to be employed as a heuristic for identifying promising hypotheses and pruning away search space in a mindful way. The revised algorithm is shown in Fig. 5. Unlike in the basic version, shown in Fig. 4, Dijkstra is run on each `reachable` term contained in hypothesis H to uncover a path based on historical link information. For example, if $\text{link}(c, d)$, $\text{link}(d, f)$ and $\text{link}(f, g)$ are all known, then the path searching subroutine will uncover a path from c to g in hypothesis $H = \text{reachable}(a, c) \wedge \text{reachable}(c, g)$ and will change the hypothesis to $H = \text{reachable}(a, c) \wedge \text{link}(c, d) \wedge \text{link}(d, f) \wedge \text{link}(f, g)$ without having to go through the inverse resolution a large number of times to arrive at the same hypothesis.

By integrating both deductive and inductive processes, our approach allows an agent performing induction to employ deductive subroutines that can be used at its discretion. For example, an agent may have a subroutine for finding the shortest path while also having a subroutine finding *any* particular path. This aspect of the research has its own significance because deductive inference and inductive inference often take two independent paths. Our approach has shown that they can be combined tightly together as an effective way of acquiring new knowledge while performing reasoning. It also sheds a light on programming inductive agents at a higher level of abstraction in which what algorithm an agent actually runs doesn’t have to be hard coded. Instead, a number of different deductive subroutines may be employed and these deductive subroutines can be utilized by agents when required. Based on what an agent is committed to do at a particular moment, it selects the suitable subroutine to execute as part of problem solving while executing the same induction process.

C. Path Planning Involving Multiple Agents

Take the example in Fig. 6 and assume car A is interested in going from a to l . Posted as a collaborative ILP problem as defined in section II, the path planning problem becomes: “collaboratively find a hypothesis H such that the example $E = \text{reachable}(a, l)$ is explainable using the total background knowledge \mathbb{B} of all agents”. It can be observed

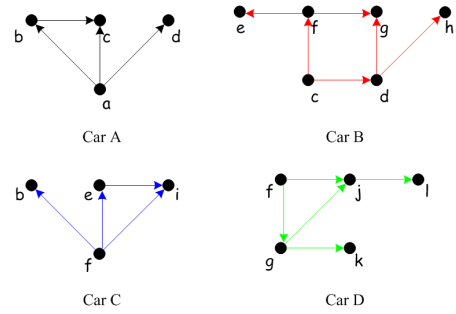


Fig. 6. Example showing path information being distributed among four agents. One path from a to l is $a-d-g-j-l$.

from the figure that one such hypothesis would be $H = \text{link}(a, d) \wedge \text{link}(d, g) \wedge \text{link}(g, j) \wedge \text{link}(j, l)$.

In a distributed setting, the pursue of this global hypothesis involves the following key elements: 1) the global inductive problem must be decomposed into a series of localized inductive problems that can be solved by individual agents; 2) an individual agent’s inductive process must enable it to contribute a partial solution to the problem; 3) the agents must carefully maintain their knowledge and systematically exchange information such that the partial solutions can be integrated together to form the final solution. While step 2) has been described thoroughly in the previous sections, here we provide a high level description to what is involved in step 1) and 3) as the exact details deserve a comprehensive treatment on its own.

Consider the example in Fig. 6 again. Car A starts with $E = \text{reachable}(a, l)$ that it wants to construct an H to explain. Assume car A asks car D first. Car D will perform the induction and obtain a hypothesis, say $H = \text{reachable}(a, g) \wedge \text{link}(g, j) \wedge \text{link}(j, l)$. It returns $\text{reachable}(a, g)$ back to car A. Car A infers that Car D knows $\text{reachable}(g, l)$, i.e. $K_a(K_d(\text{reachable}(g, l)))$ (by performing reasoning on the definition of `reachable`). Therefore, it knows that as long as someone knows (or can explain) $\text{reachable}(a, g)$, the path can be found, i.e. $K_a(\exists i K_i(\text{reachable}(a, g)) \rightarrow K_a(\text{reachable}(a, l)))$. At this stage, the overall problem has changed. Car A is now interested in constructing an H to explain $E = \text{reachable}(a, g)$. Later on, by collaboration with car B, they would be able to induce the path from a to g . Since car A remembers car D knows $\text{reachable}(g, l)$, the full path from a to l would thus be found eventually.

Here, communication saving comes from three aspects: 1) communication of `link` information doesn’t happen until a path has been fully worked out by keeping track of who knows what as epistemic information; 2) if car D knows a thousand nodes, only the ones that lead to node l will ever be transferred across; 3) instead of returning all generated hypotheses back to the initiator, a scoring method can be adopted to discriminate the hypotheses further. For example, we have used a variation of the ‘minimum description length’

(MDL) metrics to favor shorter hypotheses but using more link terms in our implementation. However, it is noticed that regardless of what scoring method is being used, just returning the hypothesis with the highest score to the query initiator often makes the entire approach incomplete. This is due to the fact that each agent can make no assumption about other agents’ knowledge status while evaluating a hypothesis and the scores assigned only reflect its own knowledge of the world. Therefore, it can often be the case that a suboptimal hypothesis to one agent may well be what the other party is actually looking for. Generally speaking, the best one can do is to increase the likelihood of returning a better hypothesis in earlier attempts by having the agents progressively returning starting from the hypothesis with the highest score, one at a time, until either all hypotheses are returned or the other party is satisfied.

D. Results

Experiments were performed to gather empirical data about communication costs during interaction using our inductive approach. The experiments were performed on a 10×6 grid environment containing 61 known links and 50 known locations. Since our approach allows just two agents to communicate at any particular time, in the experiment, we choose to investigate communication cost by just focusing on pair-wise interaction between agents. In the experiment, Dijkstra’s shortest path algorithm has been used as the deductive algorithm and for simplicity we have assigned the same unit cost to every link.

Agent A is chosen to be the initiator of queries. Data are gathered about the amount of information communicated using our inductive approach and are compared with a centralized approach of combining distributed path information before executing a path finding algorithm. Table I shows this comparison, where the knowledge distribution between the two agents is varied. The communication involved using the centralized approach (third column) is compared with the average communication using the inductive approach with no scoring of hypothesis (forth column), i.e. agents return generated hypothesis in arbitrary order. The last column shows the communication cost of the inductive approach using the MDL scoring method previously mentioned and progressively returning starting from the best hypothesis.

As expected, in general the communication cost is higher when knowledge is evenly distributed, which indicates more knowledge need to be shared in order to arrive at the solution. However, as can be seen from the table, using the inductive approach (with or without scoring) almost always outperforms an approach where sharing complete knowledge is involved in terms of communication cost, except for in special cases (the last two rows) where the agent to be consulted has very little knowledge, in which case it is actually cheaper to simply ask the agent to transfer across all its knowledge. The use of scoring cuts off communication cost further in most cases.

The top half of the table represents the cases where the initiator (Agent A) has more knowledge and the bottom half the opposite. These two different cases have been plotted

TABLE I
COMMUNICATED MESSAGES FOR CENTRALIZED, ILP AND ILP+SCORING IN A 10×6 GRID AMONG PAIRS OF CARS WITH NON-OVERLAPPING LOCAL KNOWLEDGE. DISTRIBUTION OF KNOWLEDGE IS SHOWN ON LEFT (AS NUMBER OF TERMS KNOWN).

A’s knowl- edge	B’s knowl- edge	Entropy	Cost of Communication		
			sharing knowledge (Central- ized)	no scoring (ILP)	scoring (ILP)
4	57	0.35	17	12	12
8	53	0.56	21	17	16
13	48	0.75	26	18	15
17	44	0.85	30	18	19
21	40	0.93	34	19	16
25	36	0.98	38	22	22
28	33	1.00 ¹	35	22	17
33	28	1.00 ¹	30	23	10
38	23	0.96	25	27	16
42	19	0.90	21	27	11
47	14	0.78	16	28	8
51	10	0.64	12	23	8
54	7	0.51	9	24	14
57	4	0.35	6	3	6

separately in Fig. 7, in which we have used entropy as the measure of knowledge distribution. We calculate entropy $\sum_{i=1}^n -\frac{|K_i|}{|K_T|} \ln \frac{|K_i|}{|K_T|}$ based on the number of terms that each agent knows, denoted $|K_i|$, normalized by the total knowledge $|K_T|$. The dotted line corresponds to the third column in the table, the dashed line fourth and the solid line fifth. Note that the communication cost is not symmetrical with respect to entropy in these two different cases. This is mainly because if the initiator has the majority of the knowledge, the centralized approach performs well since it requires all knowledge to be transferred to the initiator such that the communication of hypotheses back to the initiator is not necessary any more. Nevertheless, as shown in the graphs, our implementation using induction with scoring (the solid line) not only always has the lowest communication cost but also has the lowest rate of increase as knowledge becomes more evenly distributed.

The epistemic reasoning possibilities of this work also promise to extend the basic techniques presented above to make it more cost effective. As previously commented, better scoring techniques than we have used here are possible, facilitating the exchange of additional epistemic information which would lead to additional savings in communication. For example, agents may benefit from gathering and considering hypotheses from more than one agent before pursuing further; they may discriminate information from one source against another; they may even know based on some prior knowledge whom to avoid asking some particular queries. In Fig. 6, for example, if car A can somehow foresee that car C is totally ignorant of the region car A is exploring, or car D is more likely to know the answer car A is seeking, further communication saving will be possible. In general, we believe performing reasoning at epistemic level will play a key role in tackling these problems [8]. Nevertheless, our model

¹The numbers have been rounded.

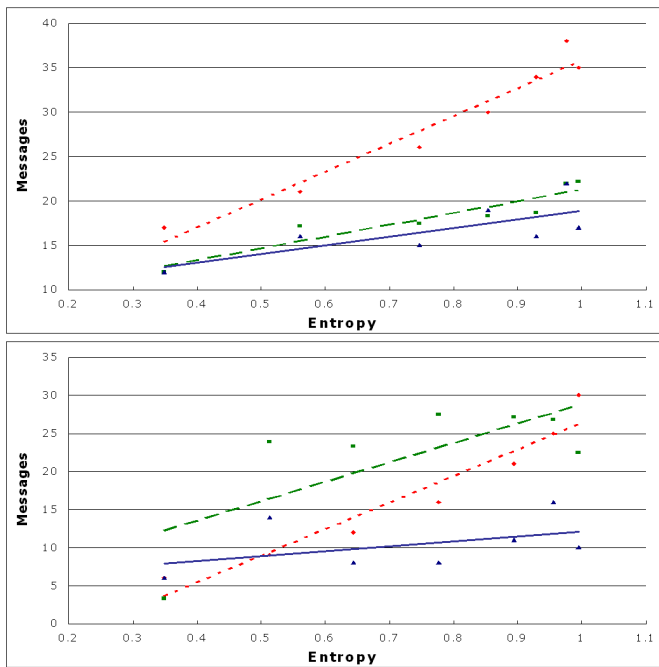


Fig. 7. Communication versus knowledge distribution for centralized (dotted line), ILP (dashed line) and ILP+Scoring (solid line).

accommodates the treatment of reasoning at epistemic level using the \mathbb{K} operator and the knowledge base.

VI. RELATED WORK

Exploratory work by Davies [9] has also investigated learning new concepts among multiple agents using decision tree techniques. However, our work develops multi-agent induction based on a more formal treatment and integration of knowledge [10]. This includes integrating capabilities and utilizing induction [11] towards the aims of true multi-agent learning, as identified by [2], using inductive logic programming (ILP).

Our work shares some similarities with abductive logic programming (ALP) [12] given that it integrates deduction and induction to constrain explanations. Consequently, the kind of induction we tackle can also be viewed as abduction, or explanatory induction as defined in [13], as opposed to the (harder to compute) descriptive induction.

Since ILP, in the limit, can be intractable unless the search is effectively constrained, traditional implementations of ILP frequently rely on the incorporation of domain knowledge to constrain the search: based on contextual information, in the spirit of [6].

In relation to the hypothesis scoring technique used in this paper, since it chooses the minimum size hypothesis it is essentially a minimum description length (MDL) approach, also known as minimum message length (MML) in some literature. It is well known that the MDL heuristic, in general, does not necessarily guarantee solutions due to the limitation of independence of evidence assumption inherent in minimum hypothesis formulation (for details, see [14]). However, the key to our approach does not rely specifically on MDL

and can utilize any (possibly even complete) technique for scoring hypothesis, given the specific constraints of individual applications.

VII. CONCLUSION

In this paper, we have demonstrated a solution to inductive logic programming problem in multi-agent environments and its application in collaborative problem solving. We have illustrated the approach using two examples. Experimental results indicate that our inductive approach not only has the potential to overcome the problem of knowledge being distributed, but also reduces communication costs while allowing agents to combine localized knowledge for collaboratively deriving solutions.

We have also demonstrated how deductive algorithms, such as the Dijkstra path finding algorithm, can be embedded as (deductive) heuristics within the process of induction, to achieve an informed way of performing induction. In spite of this, our approach does not rule out any general-purpose ILP solution. Our approach moves towards integrating deductive and inductive processes and promises improved problem solving characteristics in collaborative settings.

REFERENCES

- [1] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.
- [2] D. Kazakov and D. Kudenko, "Machine learning and inductive logic programming for multi-agent systems," in *Multi-Agent Systems and Applications*, M. Luck, V. Marik, and O. Stepankova, Eds. Springer, 2001, vol. 2086, pp. 246–270.
- [3] G. Weiß and P. Dillenbourg, "What is 'multi' in multiagent learning?" in *Collaborative learning. Cognitive and computational approaches*, P. Dillenbourg, Ed. Pergamon Press, 1999, pp. 64–80.
- [4] J. Huang and A. R. Pearce, "Distributed interactive learning in multi-agent systems," in *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. AAAI Press, 2006, pp. 666–671.
- [5] S. Muggleton and L. D. Raedt, "Inductive logic programming: Theory and methods," *Journal of Logic Programming*, vol. 19/20, pp. 629–679, 1994.
- [6] A. Srinivasan, "Extracting context-sensitive models in inductive logic programming," *Machine Learning*, vol. 44, no. 3, pp. 301–324, 2001.
- [7] B. van Linder, W. van der Hoek, and J.-J. C. Meyer, "Formalising abilities and opportunities of agents," *Fundamenta Informaticae*, vol. 34, no. 1-2, pp. 53–101, 1998.
- [8] W. van der Hoek, "Logical foundations of agent-based computing," in *Multi-agents Systems and Applications*. Springer-Verlag New York, Inc., 2001, pp. 50–73.
- [9] W. Davies, "ANIMALS A distributed heterogeneous multi-agent machine learning system," Ph.D. dissertation, University of Aberdeen, 1993.
- [10] R. Fagin, Y. Moses, J. Y. Halpern, and M. Y. Vardi, "Knowledge-based programs," *Distributed Computing*, vol. 10, no. 4, pp. 199–225, 1997.
- [11] S. Muggleton, "Inverse entailment and progol," *New Generation Computing, Special issue on Inductive Logic Programming*, vol. 13, pp. 245–286, 1995.
- [12] M. Denecker and A. C. Kakas, "Abduction in logic programming," in *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*. London, UK: Springer-Verlag, 2002, pp. 402–436.
- [13] N. Lachiche, "Abduction and induction from a non-monotonic reasoning perspective," in *Abduction and Induction*, ser. Applied Logic Series, P. A. Flach and A. C. Kakas, Eds. Kluwer Academic Publishers, 2000, pp. 107–116.
- [14] M. Li and P. M. B. Vitanyi, "Inductive reasoning and kolmogorov complexity," in *Structure in Complexity Theory Conference*, 1989, pp. 165–185.