Collaborative Inductive Logic Programming for Path Planning

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 distributed systems, 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. In this paper, we develop and evaluate a new approach for learning in distributed systems that tightly integrates processes of induction between agents, based on inductive logic programming techniques. The paper's main contribution is the integration of an epistemic approach to reasoning about knowledge with inverse entailment during induction. The new approach facilitates a systematic approach to the sharing of knowledge and invention of predicates only when required. We illustrate the approach using the 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.

1 Introduction

The problem of *true* multi-agent learning has far more complexity than simply having each agent perform localized induction in isolation (see [Stone and Veloso, 2000; Kazakov and Kudenko, 2001]). 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 and Dillenbourg, 1999].

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. 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.

In prior work published by the authors [Huang and Pearce, 2006a], the possible synergy between program execution and induction has been demonstrated for inducing missing predicates in a distributed program setting. Under the multi-agent interactive learning framework, MAILS, agents are equipped with background knowledge, expressed as logic programs, and they reason about what they know, based on their collaboratively 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). Recently, the approach has been formalized for a wider range of problem solving tasks based on the (more generalized) problem of collaborative inductive logic programming (C-ILP) [Huang and Pearce, 2006b].

In this paper, we further develop the work by extending and integrating the epistemic aspects of the approach and better evaluate the approach for an extended task. We illustrate the approach using the well-known path planning problem in a distributed setting, where path information (such as reachability and cost) is distributed over different agents. Although each agent perceives only partial information about the environment, our approach enables them to combine their local perceptions in an effective way and collaboratively work out the path from a source to a destination, which no agent would be able to do in isolation. We empirically show that the approach shows promise for avoiding central control and reducing communication costs involved.

The collaborative inductive logic programming (C-ILP) technique is based on an inverse resolution approach to learning [Muggleton and Raedt, 1994]. 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 [Srinivasan, 2001]. 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 inferenc-

ing 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.

Section 2 defines the problem of collaborative ILP. Section 3 and 4 describe details of our proposed induction technique involving distributed knowledge sources when applied to the distributed path planning problem. Section 5 reports on the empirical results addressing the advantages of our approach. Section 6 looks at some existing work on path planning in multi-agent environments and other distributed problem solving techniques in situations where no centralized control is possible.

2 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 $B \wedge H \models E$ [Muggleton, 1995]. In multi-agent systems, ILP involves generating hypotheses using the *collective* background knowledge. More formally, the process of collaborative ILP (C-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 [Muggleton and Raedt, 1994] 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 to execute a centralized ILP algorithm is often infeasible in practice. Therefore, inductively generating the hypothesis in multi-agent systems relies on careful exchange of information between agents during learning, for which we believe epistemic reasoning plays an important role.

3 Path Planning using Induction

In this section and the sections following, we demonstrate an application of the C-ILP approach to the distributed path planning problem and 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 `reachable(a,b)` stands for "`b` is reachable from `a`". The term `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 the following background knowledge:

1. `link(A,B)` $\rightarrow$ `reachable(A,B)`
2. `reachable(A,B)` $\wedge$ `reachable(B,C)` $\rightarrow$ `reachable(A,C)`

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 `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 `reachable(A,B)`, whether it is reachable from one location to another.

On the other hand, under an inductive framework, given the same query `reachable(A,B)`, the car can come up with hypotheses H, together with its background knowledge B, to explain this query, i.e. $H \wedge B \models$ `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 `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.1 illustrates the process of inverse resolution while generating hypotheses to explain the query `reachable(a,g)` given a link history `link(c,d)`, `link(c,e)`, `link(d,e)`, `link(d,f)`, `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 $H_n =$ `reachable(a,c)` $\wedge$ `link(c,d)` $\wedge$ `link(d,f)` $\wedge$ `link(f,g)` in the example is interpreted as: `reachable(a,g)` (the query) is true so long as `reachable(a,c)` is true given that `link(c,d)`, `link(d,f)` and `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

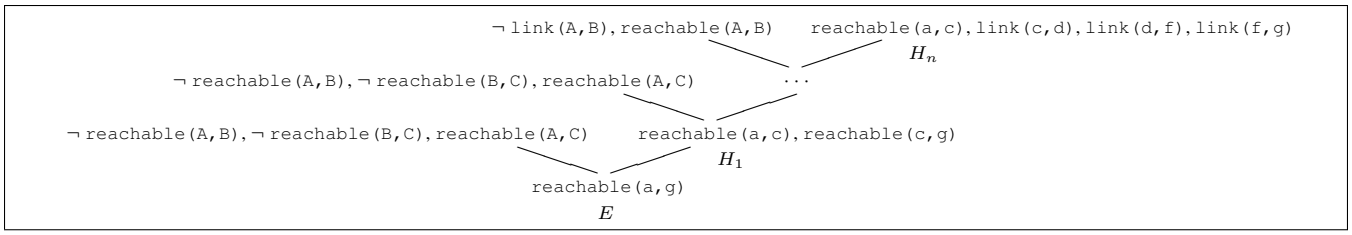

Figure 1: Inverse resolution: background knowledge used is shown on the left branches and hypotheses generated on the right.

GENHYPO($Query$)
1: $HypList \leftarrow \{Query\}, HypHistory \leftarrow \varnothing$
2: **while** $HypList \neq \varnothing$ **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** $HypAll = \varnothing$ **then**
9: $HypHistory \leftarrow \{H\} \cup HypHistory$
10: **else**
11: $HypList \leftarrow HypAll \cup HypList$
12: SCOREHYPO($HypHistory$)
13: **return** all H in $HypHistory$ in the order of score.

Figure 2: Algorithm for hypothesis generation utilizing deductive shortest path subroutine and scoring.

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 uncover the full path.

3.1 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. 2. 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.

3.2 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 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. Dijkstra is run on each `reachable` term contained in hypothesis `H` to uncover a path based on historical link information. For example, if `link(c,d)`, `link(d,f)` and `link(f,g)` are all known, then the path searching subroutine will uncover a path from `c` to `g` in hypothesis $H =$ `reachable(a,c)` $\wedge$ `reachable(c,g)` and will change the hypothesis to $H =$ `reachable(a,c)` $\wedge$ `link(c,d)` $\wedge$ `link(d,f)` $\wedge$ `link(f,g)` without having to go though 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.

4 Collaborative Path Planning

In the previous section, we have demonstrated how induction allows an agent to not only find out a path if it exists but also 'guess' a hypothetical path which can be pursued further. In this section, we explore the interactive aspects which enable multiple agents with distributed knowledge to collaboratively find out a path. Take the example in Fig. 3 and assume car A is interested in going from `a` to `l`. Posted as a collaborative ILP problem as defined in section 2, the path planning problem becomes: "collaboratively find a hypothesis H such that the example $E =$ `reachable(a,l)` is

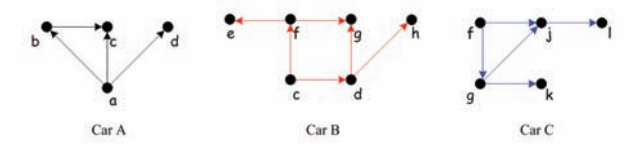

Figure 3: Example showing path information being distributed among four agents. One path from a to l is a-d-g-j-l.

explainable using the total background knowledge $\mathbb{B}$ of all agents". It can be observed from the figure that one such hypothesis would be $H = \texttt{link(a,d)} \wedge \texttt{link(d,g)} \wedge \texttt{link(g,j)} \wedge \texttt{link(j,l)}$.

Generally speaking, in a distributed setting, the pursuit of a global hypothesis involves the following key elements: (i) the global inductive problem must be decomposed into a series of localized inductive problems that can be solved by individual agents; (ii) an individual agent's inductive process must enable it to contribute a partial solution to the problem; and (iii) 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 element (ii) has been described thoroughly in the previous sections, here we provide a description to what is involved in (i) and (iii).

Consider the example in Fig. 3 again. The interaction steps can be summarized as follows:

1	A :	$E = reachable(a,l)$
2	A ASKS C :	$E = reachable(a,l)$
3	C INDUCES:	$H = reachable(a,g), link(g,j), link(j,l)$
4	C REPLIES:	$H = reachable(a,g)$
5	A DEDUCES:	$K_1 = K_a(K_c(reachable(g,l)))$
		$K_2 = K_a(\exists i K_i(reachable(a,g)) \rightarrow K_a(reachable(a,l)))$
6	A :	$E = reachable(a,g)$
7	A ASKS B :	$E = reachable(a,g)$
8	B INDUCES:	$H = reachable(a,c), link(c,d), link(d,g)$
9	B REPLIES:	$H = reachable(a,c)$
10	A DEDUCES:	$K_3 = K_a(K_b(reachable(c,g)))$
		$K_4 = K_a(\exists i K_i(reachable(a,c)) \rightarrow K_a(reachable(a,g)))$
11	A :	$E = reachable(a,c)$
12	A INDUCES:	$H = link(a,c)$

Car A starts (step 1) with $E = \texttt{reachable(a,l)}$ which it wants to construct an H to explain. Assume car A asks car C first. Car C will perform the induction and obtain a hypothesis, say $H = \texttt{reachable(a,g)} \wedge \texttt{link(g,j)} \wedge \texttt{link(j,l)}$. It returns reachable(a,g) back to car A. Car A infers that car C knows reachable(g,l) (by performing reasoning on the definition of reachable) (step 5). Therefore, it knows that as long as someone knows (or can explain) reachable(a,g), the path can be found. At this stage (step 6), the overall problem has changed. Car A is now interested in constructing an H to explain $E = \texttt{reachable(a,g)}$. Later on, through collaboration with car B, they would be able to induce the path from a to g. Since car A remembers that car C knows reachable(g,l), the full path from a to l would thus be found eventually. Of course, this requires car A to go back to its knowledge base and retrieve what has been inferred before about who knows what. The actual transfer of link information will then take place. The interaction among the three cars in this example is shown in Fig. 4.

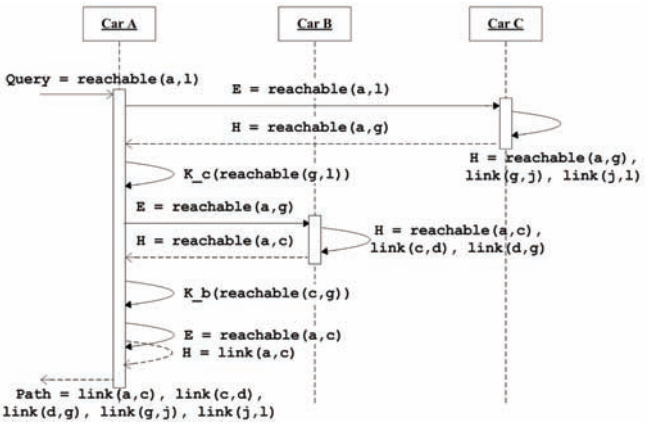

Figure 4: Interaction and information passing among the cars in Fig. 3 when searching for a path from a to l.

```
CPP(Query, Know)
1: Hyp ← GENHYPO(Query) // induce the path by itself
2: if PATHFOUND(Hyp) then
3:       Path ← RETRIEVEPATH(Know)
4:       return Path
5: for ∀i ∈ A do
6:       Hyps ← Hyps ∪ ASK(i, Query)
7: if Hyps = ∅ then
8:       return FAIL
9: if PATHFOUND(Hyps) then
10:      Path ← RETRIEVEPATH(Know)
11:      return Path
12: while Hyps ≠ ∅ do
13:      Query ← CHOOSEBEST(Hyps) // hypothesis to pursue next
14:      Know ← GENKNOW(Query) // generate new knowledge
15:      Hyps ← Hyps − {Query}
16:      CPP(Query, Know)
17: return FAIL
```

Figure 5: Algorithm for collaborative path planning that combines agent interaction, induction and epistemic reasoning.

Our general algorithm for collaborative path planning is given in Fig. 5. When engaged in answering a Query, an agent first attempts to solve it by itself (line 1 to 4). If it succeeds, it goes back to its knowledge base, retrieves the Path, returns it and finishes. If it fails to find the path itself, it asks every other agent in the team (line 5 to 11). If no hypothesis is received from any other agent, it fails since the path doesn't exist. If any hypothesis indicates the path has been found, it goes back to its knowledge base, retrieves the Path, returns it and finishes. Otherwise (line 12 to 17), it picks the most promising hypothesis among all returned by the other agents and proceeds with it as the new Query, until all those hypotheses have been tried in which case it fails since the path doesn't exist. During retrieval of the Path, it bases on the knowledge in its knowledge base to decide whom to request for the actual link information.

Here, communication saving comes from three aspects: 1)

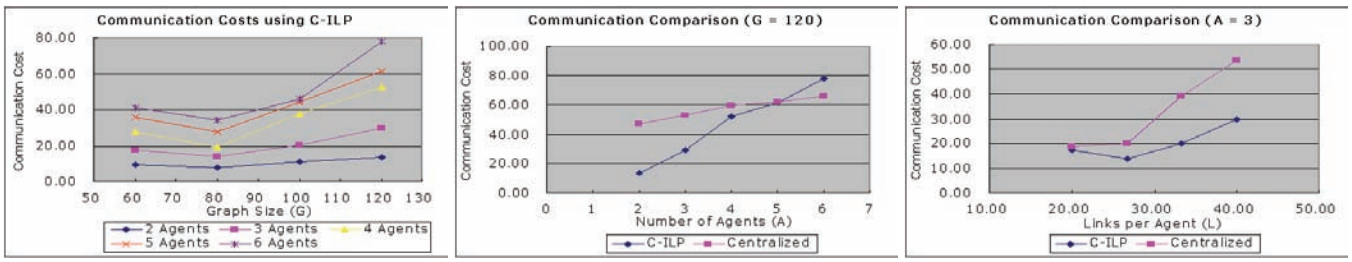

Figure 6: LEFT: shows communication costs (amount of information transferred) associated with the C-ILP approach to collaborative path planning. MIDDLE: compares communication costs using the C-ILP approach with a centralized approach, with a graph of 120 links for different numbers of agents. RIGHT: shows communication savings as the information per agent increases, using 3 agents and graphs of different sizes.

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) metric 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.

5 Experiments and Analysis

Some preliminary experimental results were shown in [Huang and Pearce, 2006b] focusing on pairwise communication costs during interaction between two agents. More thorough experiments have been performed and the results reported in this paper analyze the effect in communication costs with different numbers of agents, A, and with varying graph sizes, G. In our experiments A varies from 2 to 6 and G varies from 60 to 120. The experiments involve distributing the total G links randomly over the A agents. We ran 100 trials for every different value of A and G using the C-ILP approach and compared this against a centralized approach in terms of the total communication cost involved (measured as the number of "terms" transferred). The centralized approach involves having every agent (one at a time) transferring across distributed path information to the query initiator before executing a path finding algorithm.

Results are summarized and plotted in Fig. 3. The one on the left shows the increasing of communication costs associated with the proposed C-ILP approach as the total number of agents in the system arises. The middle graph compares communication costs using the C-ILP approach against the centralized approach, with a graph of 120 links and varying number of agents. We noticed that communication savings with the C-ILP approach are significant when the total number of agents is small. As the number of agents increases (along with the entropy of the system) the benefit decreases and finally the cost using C-ILP exceeds that of the centralized approach. A similar situation occurs when using graphs of other sizes. We believe this can be explained in terms of the participation of agents during the solution of the C-ILP problem. As the number of agents increases, the link information is more evenly distributed over agents and each agent solves a smaller partial sub-problem. As a result, each agent has less knowledge which makes it harder to come up with useful hypotheses. Consequently, communication costs can override the benefit gained from induction. We explore this aspect and illustrate in the graph on the right the relationship between communication costs and partial knowledge represented as "links per agent", L. It is found that the C-ILP approach outperforms the centralized approach uniformly with different numbers of agents and graph sizes when each agent has, roughly speaking, 30 links or more.

More sophisticated epistemic reasoning 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 know who is more likely to know the answer to a particular query; they may even know based on some prior knowledge whom to avoid asking some particular queries. In general, we believe performing reasoning at epistemic level will play a key role in tackling these problems [van der Hoek, 2001]. Nevertheless, our model accommodates the treatment of reasoning at epistemic level using the $\mathbb{K}$ operator and the knowledge base.

6 Related Work

Exploratory work by Davies [Davies, 1993] 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 [Fagin *et al.*, 1997]. This includes integrating capabilities and utilizing induction [Muggleton, 1995] towards the aims of true multi-agent learning, as identified by [Kazakov and Kudenko, 2001], using inductive logic programming (ILP).

Our work shares some similarities with abductive logic programming (ALP) [Denecker and Kakas, 2002] 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 [Lachiche, 2000], 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 [Srinivasan, 2001].

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 [Li and Vitanyi, 1989]). 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.

7 Conclusion

In this paper, we have looked at the problem of path planning in multi-agent environments by combining partial knowledge of the individuals and have demonstrated a distributed inductive approach as an effective technique for problem solving. Experiments have showed the promise of our inductive approach not only for overcoming the problem of knowledge being distributed but also for saving communication costs while allowing the agents to combine localized knowledge and collaboratively find solutions. We have also demonstrated how deductive algorithms could be embedded as an heuristic within the process of induction to achieve an informed way of performing induction. We claim although the problem we looked at in this paper is specific, the approach that involves the integration of deduction and induction processes while solving problems involving collaboration is useful in general.

References

[Davies, 1993] Winton Davies. *ANIMALS A Distributed Heterogeneous Multi-Agent Machine Learning System*. PhD thesis, University of Aberdeen, 1993.

[Denecker and Kakas, 2002] Marc Denecker and Antonis C. Kakas. Abduction in logic programming. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, pages 402–436. Springer-Verlag, London, UK, 2002.

[Fagin *et al.*, 1997] Ronald Fagin, Yoram Moses, Joseph Y. Halpern, and Moshe Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10(4):199–225, 1997.

[Huang and Pearce, 2006a] Jian Huang and Adrian R. Pearce. Distributed interactive learning in multi-agent systems. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 666–671. AAAI Press, 2006.

[Huang and Pearce, 2006b] Jian Huang and Adrian R. Pearce. Toward inductive logic programming for collaborative problem solving. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*. IEEE Press, 2006.

[Kazakov and Kudenko, 2001] Dimitar Kazakov and Daniel Kudenko. Machine learning and inductive logic programming for multi-agent systems. In M. Luck, V. Marik, and O. Stepankova, editors, *Multi-Agent Systems and Applications*, volume 2086, pages 246–270. Springer, 2001.

[Lachiche, 2000] Nicolas Lachiche. Abduction and induction from a non-monotonic reasoning perspective. In Peter A. Flach and Antonis C. Kakas, editors, *Abduction and Induction*, Applied Logic Series, pages 107–116. Kluwer Academic Publishers, 2000.

[Li and Vitanyi, 1989] M. Li and P. M. B. Vitanyi. Inductive reasoning and kolmogorov complexity. In *Structure in Complexity Theory Conference*, pages 165–185, 1989.

[Muggleton and Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

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

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

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

[van der Hoek, 2001] Wiebe van der Hoek. Logical foundations of agent-based computing. In *Mutli-agents Systems and Applications*, pages 50–73. Springer-Verlag New York, Inc., 2001.

[Weiß and Dillenbourg, 1999] Gerhard Weiß and Pierre Dillenbourg. What is 'multi' in multiagent learning? In Pierre Dillenbourg, editor, *Collaborative learning. Cognitive and computational approaches*, pages 64–80. Pergamon Press, 1999.