

Distributed Interactive Learning in Multi-Agent Systems

Jian Huang Adrian R. Pearce

Department of Computer Science and Software Engineering
The University of Melbourne
{jhua,adrian}@csse.unimelb.edu.au

April 10, 2006



Outline

- 1 Overview
 - Multi-Agent Learning
 - Multi-Agent Interactive Learning
 - Multi-Agent Interactive Learning with ILP
- 2 MAILS
 - Inductive Learning Agents
 - Learning through Interaction
 - MAILS Implementation
 - Application
 - Future Work
- 3 Summary

Multi-Agent Learning

What is Multi-Agent Learning

What does Multi-Agent Learning Concern?

- **Machine Learning** + **Multi-Agent Environment**

Extra issues:

- More learning opportunities (e.g. roles, coordination etc)
- Learning by communication (apart from via experience)
- Trade-off due to resource constraints (to ask or to learn)

Learning strategies:

- multiplication, division and **interaction** [WD99]

Multi-Agent Learning

State of the Art

Characteristics of current approaches:

- deploy individual learners in multi-agent environment;
- utilize traditional machine learning techniques;
- agents model their peers as part of the environment;
- limited communication between learners.

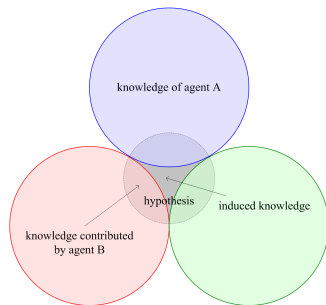
Improvements intended:

- design agents that are tightly integrated;
- develop specific strategies for multi-agent learning;
- agents model each others as colleagues;
- interaction and collaboration during learning.

Multi-Agent Interactive Learning

Why is Interaction Important?

- high level learning to model
- speed up learning
- avoid reinventing the wheel
- crucial knowledge is distributed



A Simple Example: Sorting

Agent i

- `min(List,Min) :- sort(List,L), first(L,Min).`
- `first([N|Ns],N).`

Agent j

- `permuate(List,[First|Perm]) :-
 select(First,List,Rest), permuate(Rest,Perm).`
- `select(Elem,[Head|Tail1],[Head|Tail2]) :-
 select(Elem,Tail1,Tail2).`
- `ordered([X,Y|Tail]) :-
 X =< Y, ordered([Y|Tail]).`

Hypothesis: `sort(L1,L2) :- permuate(L1,L2), ordered(L2).`



Multi-Agent Interactive Learning with ILP

Introduce MAILS

Multi-Agent Inductive Learning System (MAILS)

- contains:
 - formal specification of inductive agents
 - interaction mechanism between agents
 - implementation
- utilizes Inductive Logic Programming (ILP)

Multi-Agent Interactive Learning with Induction

Introduce ILP

Inductive Logic Programming (ILP) [MR94]

- Machine Learning + Logic Programming
- Acquires general concepts from specific examples
- Logic based approach (first-order Horn clauses)
- Utilizes background knowledge while inducing hypothesis
- In summary: $B \wedge H \models E$

A Simple Example: Sorting

Agent i

- `min(List,Min) :- sort(List,L), first(L,Min).`
- `first([N|Ns],N).`

Agent j

- `permuate(List,[First|Perm]) :-
 select(First,List,Rest), permuate(Rest,Perm).`
- `select(Elem,[Head|Tail1],[Head|Tail2]) :-
 select(Elem,Tail1,Tail2).`
- `ordered([X,Y|Tail]) :-
 X =< Y, ordered([Y|Tail]).`

Hypothesis: `sort(L1,L2) :- permuate(L1,L2), ordered(L2).`



Agent Constructs

ILP enabled agents enhanced with:

- 1 Background Set $\mathbb{B} = \{b_1, \dots, b_n\}$
- 2 Example Set $\mathbb{E} = E^+ \cup E^-$, e.g. $e_1^+ = \text{sort}([2, 1, 3], [1, 2, 3])$
- 3 Capability Set $\mathbb{C} = C^+ \cup C^-$, e.g. $c_1^+ = A_i \text{sort}$
- 4 Knowledge Base $\mathbb{K} = K^P \cup K^E \cup K^C$
 - K^P : e.g. $K_i(A_i \text{min} \leftarrow A_i \text{sort} \wedge A_i \text{first})$
 - K^E : e.g. $K_i(\text{sort}([2, 1, 3], [1, 2, 3]))$, $K_i(\setminus + \text{sort}([1, 3], [1, 2]))$
 - K^C : e.g. $K_i(A_i \text{first})$, $K_i(\setminus + A_i \text{min})$

Agent Reasoning

Extended Example

Agent *i*

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

Is agent *i* capable of performing *range*?

Agent Reasoning

Reasoning about Capability

Is agent i capable of performing *range*?

$$A_i \text{range} \leftarrow A_i \text{min} \wedge A_i \text{max} \quad (1)$$

$$A_i \text{min} \leftarrow A_i \text{sort} \wedge A_i \text{first} \quad (2)$$

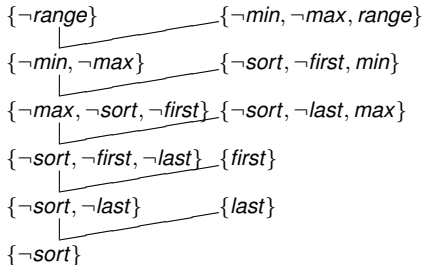
$$A_i \text{max} \leftarrow A_i \text{sort} \wedge A_i \text{last} \quad (3)$$

$$A_i \text{first} \quad (4)$$

$$A_i \text{last} \quad (5)$$

\therefore can't resolve $\neg \text{sort}$,

\therefore unable to perform *range*.



Extended Deduction

Deduction with Induction

- Deduction is extended to allow acquisition of new knowledge.
- Induction involves collaboration among agents.

Example

- Agent i : knows about `range`, `min` and `max`; knows the examples about `sort`.
- Agent j : knows about `permutate`, `select` and `ordered`.
- Agent k : is asked to deduce `same ([2, 2, 2, 2])`.

Extended Deduction

DEDUCE(*Goal*)

```

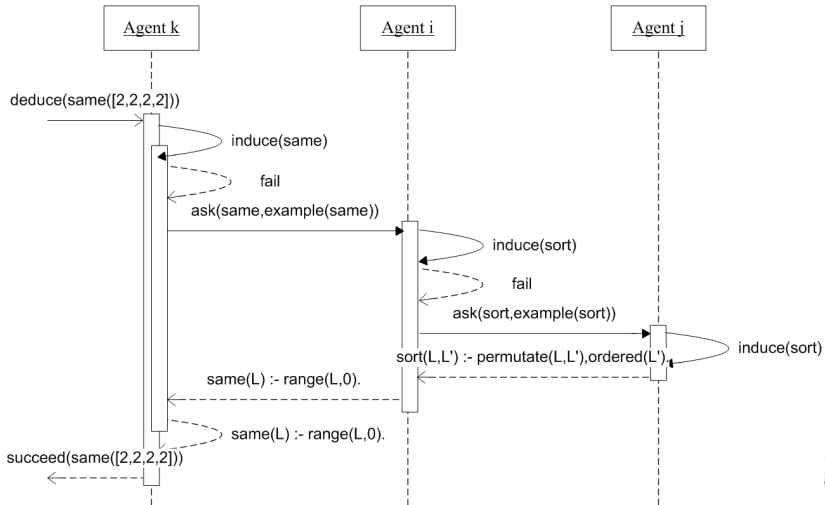
initialize goal list with Goal
while goal list is not empty do
  pick the first goal g
  if g is defined then
    if g is resolvable then
      replace g with its body
    else
      return FAIL
    end if
  end if
else
  INDUCE(g, example(g))
  if g is induced then
    continue
  else
    return FAIL
  end if
end if
end while
return SUCCEED
  
```

INDUCE(*Pred*, *Example*)

```

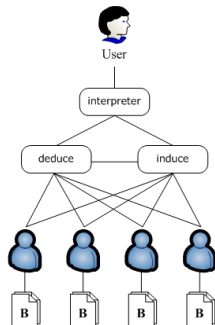
for all background predicate b do
  if b does not depend on Pred then
    if b is not defined then
      INDUCE(b, example(b))
    end if
    Background ← Background ∪ b
  end if
end for
ILP(Pred, Background, Example)
if Pred is induced then
  return SUCCEED
else
  for each agent i in the team do
    ASK(i, Pred, Example)
    if Pred is induced then
      return SUCCEED
    end if
  end for
  invoke distributed ILP process
  if Pred is induced then
    return SUCCEED
  else
    return FAIL
  end if
end if
  
```

Extended Deduction



MAILS Implementation

- Prolog + Aleph [Sri01]
- simulated multi-agent environment
- simulated message passing
- can query a specific agent, e.g.
`deduce(i, range([2, 5, 8], R))`
- figure out missing knowledge at run time among the agents
- quick demo



Application: Intelligent Home

Clock Agent

- `wake_up(Date, Time) :-
 time_to_get_up(Date, Time),
 not_weekend(Date).`
- `not_weekend(Date) :-
 \+ Saturday(Date), \+ Sunday(Date).`

- `time_to_get_up(2-1-2006, 8am).`
- `time_to_get_up(3-1-2006, 8am).`
- `time_to_get_up(4-1-2006, 9am).`
- `time_to_get_up(5-1-2006, 9am).`
- `time_to_get_up(9-1-2006, 8am).`
- ...

Application: Intelligent Home (Cont.)

Clock Agent

- `time_to_get_up(Date, Time) :-`
Date is among [monday, tuesday, friday],
Time is 8am.
- `time_to_get_up(Date, Time) :-`
Date is among [wednesday, thursday],
Time is 9am.

Clock Agent + PDA Agent




- `time_to_get_up(Date, Time) :-`
first_task_time(Date, Task_Time),
Time is Task_Time - 1.

Future Work

- Evaluate against existing work
- Investigate various trade-offs
- Investigate interaction preference
- Better worked out epistemic reasoning
- Application in more realistic domain

Summary

- Multi-agent learning as a growing research field
- Isolated Learners Vs. Interactive Learners
- MAILS:
 - Formalism + Algorithm + Implementation
 - Integrates individual agents, reasoning and learning
 - All in logic programming terms
 - Models learning as team behaviour
 - Demonstration in logic programming scenario
- Slides available at:
http://www.cs.mu.oz.au/~jhua/research/agentlab_06.pdf
- Questions

-  **Stephen Muggleton and Luc De Raedt.**
Inductive logic programming: Theory and methods.
Journal of Logic Programming, 19/20:629–679, 1994.
-  **Ashwin Srinivasan.**
Extracting context-sensitive models in inductive logic programming.
Machine Learning, 44(3):301–324, 2001.
-  **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.