Distributed Interactive Learning in Multi-Agent Systems

Conference & Conversion Seminar

Department of Computer Science and Software Engineering
The University of Melbourne

May 26, 2006
Outline

1. Overview
   - Inspiration
   - Multi-Agent Learning
   - MAILS Approach

2. MAILS
   - Inductive Learning Agents
   - Learning through Interaction
   - Implementation

3. Application
   - Intelligent Home
   - Vehicle Navigation

4. Summary
   - Future Work
   - Summary
Change computational paradigm:
- wireless sensor networks
- intelligent environment
- ubiquitous computing

Key Issues:
- communication
- learning

Learning through interaction!
Inspiration
Traffic Example
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 Inductive Learning System (MAILS) Overview

- Consists of:
  - Specification of inductive agents
  - Interaction mechanism between agents

- Features:
  - Incorporate distributed knowledge
  - Communication conservative
  - Model learning at high level

- Demonstrated for inducing declarative program fragment

- Utilizes Inductive Logic Programming (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: \( B \land H \models E \)
Example

A Simple Example: Sorting

Hypothesis: `sort([L1], [L2]) :- permutate([L1], [L2]), ordered([L2]).`

Background

Background

```
min(List, Min) :- sort(List, L), first(L, Min).
first([N|Ns], N).
permutate(List, [First|Perm]) :-
  select(First, List, Rest), permutate(Rest, Perm).
select(Elem, [Head|Tail1], [Head|Tail2]) :-
  select(Elem, Tail1, Tail2).
ordered([X,Y|Tail]) :- X =< Y, ordered([Y|Tail]).
min([1,2,3,4,5], 1).
min([6,4,8,2], 2).
min([4,3,5], 3).
```

Example

```
min([1,2,3,4,5], 1).
min([6,4,8,2], 2).
min([4,3,5], 3).
```

Hypothesis:

`sort(L1, L2) :- permutate(L1, L2), ordered(L2).`

Jian Huang

Distributed Interactive Learning in Multi-Agent Systems
Agent Constructs

ILP enabled agents with:

1. Background Set \( \mathbb{B} = \{b_1, \ldots, b_n\} \)
2. Example Set \( \mathbb{E} = E^+ \cup E^- \), e.g. \( e_1^+ = sort([2, 1, 3], [1, 2, 3]) \)
3. Capability Set \( \mathbb{C} = C^+ \cup C^- \), e.g. \( c_1^+ = A_i sort \)
4. Knowledge Base \( \mathbb{K} = K^P \cup K^E \cup K^C \)
   - \( K^P \): e.g. \( K_i(A_i min \leftarrow A_i sort \land A_i first) \)
   - \( K^E \): e.g. \( K_i(sort([2, 1, 3], [1, 2, 3])), K_i(\neg sort([1, 3], [1, 2])) \)
   - \( K^C \): e.g. \( K_i(A_i first), K_i(\neg A_i min) \)
Deduction with Induction

**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
- else
  - **INDUCE**(g, example(g))
  - if g is induced then continue
  - else
    - return FAIL
  - 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

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

Jian Huang

Distributed Interactive Learning in Multi-Agent Systems
Illustrative Example

Inducing Program Fragment

Agent $i$

```
rang($L$,R) :-
    min($L$,Min), max($L$,Max), R = Max - Min.
min($L$,Min) :- sort($L$,L), first(L,Min).
max($L$,Max) :- sort($L$,L), last(L,Max).
lst([N|Ns],L) :- lst(Ns,L).
first([N|Ns],N).
```

Agent $i$: knows about range, min and max; knows the examples about sort.
Illustrative Example
Inducing Program Fragment

Agent $j$

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

Agent $j$: knows about `permutate`, `select` and `ordered`.

Agent $k$

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

Agent $k$: is asked to deduce `same([2,2,2,2])`. 
Illustrative Example
Inducing Program Fragment

Agent k

- `deduce(same([2,2,2,2]))`
- `induce(same)`
- `ask(same, example(same))`

Agent i

- `succeed(same([2,2,2,2]))`
- `same(L) :- range(L,0)`
- `sort(L,L') :- permutate(L,L'), ordered(L')`
- `induce(same)`
- `fail`
- `ask(same, example(same))`

Agent j

- `induce(same)`
- `fail`
- `ask(same, example(same))`
Implementation

- Prolog + Aleph [Sri01]
- simulated multi-agent environment
- simulated message passing
- can query a specific agent, e.g.
  \[\text{deduce}(i, \text{range}([2, 5, 8], R))\]
- figure out missing knowledge at run time among the agents
- quick demo
Application 1: 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 1: Intelligent Home (Cont.)

Clock Agent

- \( \text{time}_\text{to}_\text{get}_\text{up}(\text{Date}, \text{Time}) :\) \\
  Date is among [monday, tuesday, friday], \\
  Time is 8am.

- \( \text{time}_\text{to}_\text{get}_\text{up}(\text{Date}, \text{Time}) :\) \\
  Date is among [wednesday, thursday], \\
  Time is 9am.

Clock Agent + PDA Agent

- \( \text{time}_\text{to}_\text{get}_\text{up}(\text{Date}, \text{Time}) :\) \\
  \( \text{first}_\text{task}_\text{time}(\text{Date}, \text{Task}_\text{Time}), \) \\
  Time is Task_Time – 1.
Application 2: Vehicle Navigation

Background

\[
\text{gone}(A,B) \rightarrow \text{going}(A,B) \\
\text{going}(A,B) \land \text{going}(B,C) \rightarrow \text{going}(A,C)
\]

Car A

gone(a,b)  
gone(a,c)  
gone(b,c)  
gone(c,d)

Car B

gone(c,d)  
gone(c,e)  
gone(d,f)  
gone(f,g)

Car A asks car B: going(a,g)?
Application 2: Vehicle Navigation (Cont.)

Background: B

- gone(A,B) → going(A,B)
- going(A,B) ∧ going(B,C) → going(A,C)
- gone(c,d), gone(c,e), gone(d,f), gone(f,g)

Example: E

- going(a,g)

Task for car B: \( B \land H \models E \)
### Application 2: Vehicle Navigation (Cont.)

#### Background: B
- \( \text{gone}(A, B) \rightarrow \text{going}(A, B) \)
- \( \text{going}(A, B) \land \text{going}(B, C) \rightarrow \text{going}(A, C) \)
- \( \text{gone}(c, d), \text{gone}(c, e), \text{gone}(d, f), \text{gone}(f, g) \)

#### Example: E
- \( \text{going}(a, g) \)

#### Hypothesis: H
- \( \text{going}(a, c) \land \text{gone}(c, d) \land \text{gone}(d, f) \land \text{gone}(f, g) \)

such that: \( B \land H \models E \)
Application 2: Vehicle Navigation (Cont.)

Background

- $\text{gone}(A,B) \rightarrow \text{going}(A,B)$
- $\text{going}(A,B) \land \text{going}(B,C) \rightarrow \text{going}(A,C)$

Car A

- $\text{gone}(a,b)$
- $\text{gone}(a,c)$
- $\text{gone}(b,c)$
- $\text{gone}(c,d)$

Car B

- $\text{gone}(c,d)$
- $\text{gone}(c,e)$
- $\text{gone}(d,f)$
- $\text{gone}(f,g)$

Car A asks car B:
$\text{going}(a,g)$?

Car B replies:
$H = \text{going}(a,c) \land \text{gone}(c,d) \land \text{gone}(d,f) \land \text{gone}(f,g)$
Future Work

- Simulate path planning and analyse
- Evaluate against existing work
- Investigate the role of epistemic reasoning
MAILS approach

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

Applications


Questions
Stephen Muggleton and Luc De Raedt.  
Inductive logic programming: Theory and methods.  

Ashwin Srinivasan.  
Extracting context-sensitive models in inductive logic programming.  

Gerhard Weiß and Pierre Dillenbourg.  
What is ’multi’ in multiagent learning?  