Learning Static Constraints For Domain Modelling From Training Plans

Rabia Jilani

University of Huddersfield, UK

Outline

- Introduction
- Simple Domain Model Operator
- LOCM (Learning Object-Centred Models)
- Static Knowledge– What is it?
- LOCM Limitation
- ASCoL (Automated Static Constraint Learner)
- LOCM -The learning problem
- ASCoL -The learning problem
- ASCoL Algorithm (Order/Graph Detector OD)
- Process Summary
- Types of Graphs
- Evaluation
- Evaluation: Challenges
- Conclusion and Future Work

Introduction

- Intelligent agents performing in the real-world use AI Planning, which requires *domain models* of the world to plot their actions

NASA Mars Rover

Machine tool calibration

Robotics

Introduction

- Conventionally such models are hand-coded... time consuming, error prone

- Motivated by the importance of the knowledge formulation process to make intelligent agents fully automatic and planning engines more accessible, several systems learn models autonomously from training action plans traces–LOCM, ARMS, Opmaker etc.

  (Cresswell et al., 2009) (Yang, Q. et al., 2007) (McCluskey, T.L. et al., 2002)

- Particular challenges:
  - Input Requirements (amount of application knowledge as input)
  - Extent of Learning in output

Planning Domain Model Operator (Example)

Planning domain model is the specification of the Object types, states (predicates), and dynamics of the domain of planning.

```
(:action sail
      :parameters  (?from ?to - location)
      :precondition (and (not-eq ?from ?to)
                         (location ?from)
                          (location ?to)
                          (at-ferry ?from))
      :effect (and  (at-ferry ?to)
                (not (at-ferry ?from)))
)
```

LOCM (Learning Object-Centred Models)

(ICAPS 2009 by Cresswell, McCluskey and West)

- Learns domain models from logged sequences of actions (plans) only
- No need for information such as predicate specification or state information *(Cresswell et al., 2013)*

"The only exception to this is the option to specify a "static" precondition, necessary in some domains which require static knowledge." *(Cresswell et al., 2013)*

Static Knowledge – What is it?

Relationships/properties that never change in the world and are implicit in the domain model but would not be directly expressed in plan traces

- Static facts are conditions required by the environment. These are represented by predicates that appear only in preconditions of operators and that restrict the values of certain variables, that's why we call it Constraints

Let

O ={O_1, . . . ,O_n(O)} set of operators

P = {P_1, . . . , P_m(P)} set of all predicates

A predicate Pi ϵ P is *Static* iff there is no operator Oj ϵ O that has an effect that uses the predicate Pi. Otherwise the predicate is *Fluent* (Wickler, G. KEPS 2011)

Example

- The drive operator of the traveling domain model (IPC)
- ***connected*** predicate is a static relation between two places.
- There is no operator in the whole domain model that changes the ***connected*** predicate

```
(:action drive-ipc
 :parameters
 (?t - truck ?from ?to - place)
 :precondition
 (and
     (at ?t ?from)
     (connected ?from ?to))
 :effect
 (and
     (not (at ?t ?from))
     (at ?t ?to))
)
```

- What if the static predicate/s is/are not present in the operator?

LOCM Limitations

- LOCM declares static constraints manually as follows:

  static(*connected(C1,C2)*, *drive(C2,C1,_,_)*)

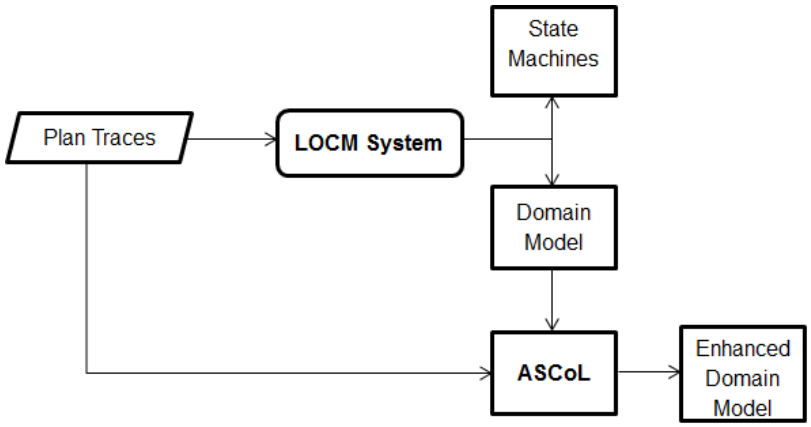
- the layout of roads in ***driverlog*** *domain* (**connect** location1, location2)
- the level of floors in ***miconic*** domain (**above** floor3, floor4)
- the fixed relationships between specific cards in ***freecell*** *(**Successor** D3, H5)*

ASCoL (Automated Static Constraint Learner)

- A tool that exploits graph analysis for automatically identifying static relations (relationships/properties that never change in the world) in order to enhance planning domain models by observing a set of training plan traces

- We enhance the output domain model of the LOCM system to capture static domain constraints from the same set of input training plans as used by LOCM

- We then generate an enhanced domain model by adding in learnt static facts (constraints)



Inputs and Output general structure of ASCoL

LOCM -The learning problem

sequence_task(15, [board(c12, l2), sail(l2, l1), debark(c12, l1), board(c1, l1), sail(l1, l3), debark(c1, l3), board(c11, l3), sail(l3, l4), debark(c11, l4), board(c3, l4), sail(l4, l1), debark(c3, l1), board(c2, l1), sail(l1, l3), debark(c2, l3), board(c24, l3), sail(l3, l4), debark(c24, l4), board(c9, l4), sail(l4, l2), debark(c9, l2), board(c15, l2), sail(l2, l0), debark(c15, l0), board(c4, l0), sail(l0, l4), debark(c4, l4), board(c10, l4), sail(l4, l0), debark(c10, l0), board(c6, l0), sail(l0, l4), debark(c6, l4), board(c14, l4), sail(l4, l1), debark(c14, l1), board(c8, l1), sail(l1, l3), debark(c8, l3), board(c25, l3), sail(l3, l0), debark(c25, l0), board(c13, l0), sail(l0, l2), debark(c13, l2), board(c17, l2), sail(l2, l0), debark(c17, l0), board(c21, l0), sail(l0, l1), debark(c21, l1), board(c16, l1), sail(l1, l4), debark(c16, l4), board(c18, l4), sail(l4, l0), debark(c18, l0), board(c26, l0), sail(l0, l3), debark(c26, l3)], _, _). ....

Input Plan Traces


```
(define
  (domain ferry)
  (:requirements :typing)
  (:types c l)
  (:predicates
    (c_state_1_1 ?v1 - c)
    (c_state_1_2 ?v1 - c)
    (c_state_1_3 ?v1 - c)
    (l_state_1_1 ?v1 - l)
    (l_state_1_2 ?v1 - l))

  (:action
  board
  :parameters
  (?C1 - c ?L2 - l)
  :precondition
  (and
      (l_state_1_1 ?L2)
      (c_state_1_1 ?C1))
  :effect
  (and
      (c_state_1_2 ?C1)
      (not (c_state_1_1 ?C1))))

  (:action
  debark
  :parameters
  (?C1 - c ?L2 - l)
  :precondition
  (and
      (l_state_1_1 ?L2)
      (c_state_1_2 ?C1))
  :effect
  (and
      (c_state_1_3 ?C1)
      (not (c_state_1_2 ?C1))))

  (:action
  sail
  :parameters
  (?L1 - l ?L2 - l)
  :precondition
  (and
      (l_state_1_2 ?L2)
      (l_state_1_1 ?L1))
  :effect
  (and
      (l_state_1_1 ?L2)
      (not (l_state_1_2 ?L2))
      (l_state_1_2 ?L1)
      (not (l_state_1_1 ?L1)))))
```

Output Domain Definition

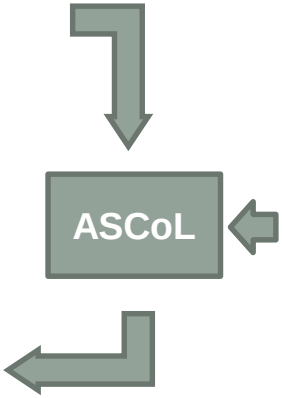
$$A_i(O_{i1}, ..., O_{ij}) \quad for \quad i = 1, ..., N$$

ASCoL -The learning problem

Input plan traces contain tacit knowledge about constraints validation/acquisition

sequence_task(15, [board(c12, l2), sail(l2, l1), debark(c12, l1), board(c1, l1), sail(l1, l3), debark(c1, l3), board(c11, l3), sail(l3, l4), debark(c11, l4), board(c3, l4), sail(l4, l1), debark(c3, l1), board(c2, l1), sail(l1, l3), debark(c2, l3), board(c24, l3), sail(l3, l4), debark(c24, l4), board(c9, l4), sail(l4, l2), debark(c9, l2), board(c15, l2), sail(l2, l0), debark(c15, l0), board(c4, l0), sail(l0, l4), debark(c4, l4), board(c10, l4), sail(l4, l0), debark(c10, l0), board(c6, l0), sail(l0, l4), debark(c6, l4), board(c14, l4), sail(l4, l1), debark(c14, l1), board(c8, l1), sail(l1, l3), debark(c8, l3), board(c25, l3), sail(l3, l0), debark(c25, l0), board(c13, l0), sail(l0, l2), debark(c13, l2), board(c17, l2), sail(l2, l0), debark(c17, l0), board(c21, l0), sail(l0, l1), debark(c21, l1), board(c16, l1), sail(l1, l4), debark(c16, l4), board(c18, l4), sail(l4, l0), debark(c18, l0), board(c26, l0), sail(l0, l3), debark(c26, l3)], _, _). ....

Input Plan Traces (P)



Static Preconditions (Constraints)

connected(loc1 loc2)

...

Every domain has separate static background

Output (R)

```
(define
  (domain ferry)
  (:requirements :typing)
  (:types c l)
  (:predicates
    (c_state_1_1 ?v1 - c)
    (c_state_1_2 ?v1 - c)
    (c_state_1_3 ?v1 - c)
    (l_state_1_1 ?v1 - l)
    (l_state_1_2 ?v1 - l))

  (:action
  board
  :parameters
  (?C1 - c ?L2 - l)
  :precondition
  (and
       (l_state_1_1 ?L2)
       (c_state_1_1 ?C1))
  :effect
  (and
       (c_state_1_2 ?C1)
       (not (c_state_1_1 ?C1))))

  (:action
  debark
  :parameters
  (?C1 - c ?L2 - l)
  :precondition
  (and
       (l_state_1_1 ?L2)
       (c_state_1_2 ?C1))
  :effect
  (and
       (c_state_1_3 ?C1)
       (not (c_state_1_2 ?C1))))

  (:action
  sail
  :parameters
  (?L1 - l ?L2 - l)
  :precondition
  (and
       (l_state_1_2 ?L2)
       (l_state_1_1 ?L1))
  :effect
  (and
       (l_state_1_1 ?L2)
       (not (l_state_1_2 ?L2))
       (l_state_1_2 ?L1)
       (not (l_state_1_1 ?L1)))))
```

Input Types' Information (T)

Input : a tuple (P, T), **P** = plan traces ,**T** = types of action arguments in P

Output : **R** = constraint repository

Process Summary

1. Read the partial domain model and the plan traces.
2. Identify, for all operators, all the pairs of arguments involving the same object types.
3. For each of the pairs, generate a directed graph by considering the objects involved in the matching actions from the plan traces.
4. Analyze the order of graphs and extract hidden static relations between arguments.
5. Run inequality/reflexivity check.
6. Return the extended domain model that includes the identified static relations.

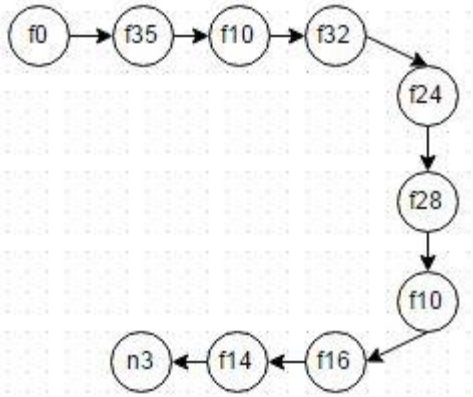
Algorithm (Order Detector OD)

Conn ← A set of binary pairs, length L
Non-Eq ← True
Order ← Empty string that will contain the order result
if *Conn* is not empty **then**
    Iterate through $Conn_i$, i ∈ 1,..., L as follows:
        if (*OD* detects atleast one cycle) **then**
            if (reflexivity) **then**
                Non-Eq = False
            if (connectivity) **then**
                return *Order* = "Connected"
            return Order = "No order"
        End if
        else if (for each pair (a, b) in *Conn*, either $a \leq b$ or $b \leq a$) **then**
            if (reflexivity)
                Non-Eq = False
            return Order = "total order"
        end else if
        else return Order = "partial order"
    end
end if

Types of Graphs

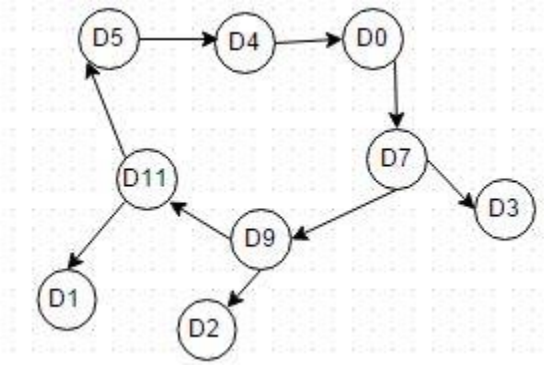
Linear Graph

sendCardsHome (card, card, suit, num, card, num)



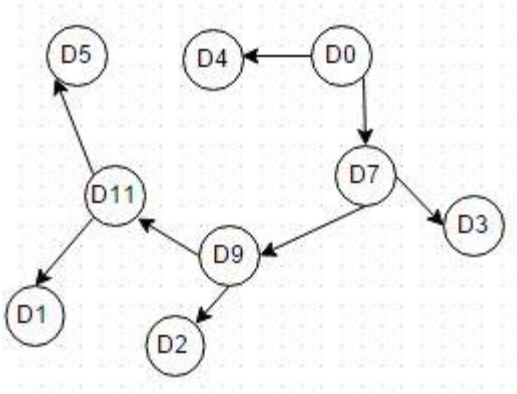
Cyclic Graph

Drive (driver, truck, locA, locB)



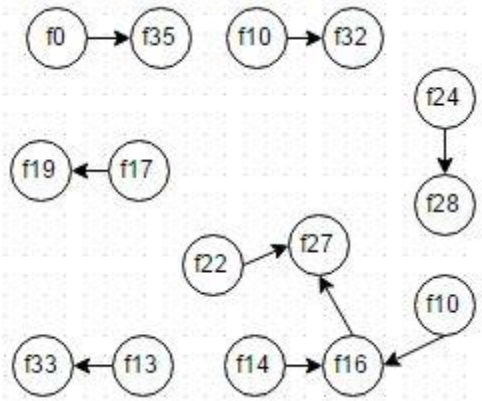
Connected Graph

Walk (person, path1, path2)



Disconnected Graph

sendCardsHome (card, card, suit, num, card, num)



Evaluation

number of operators **(# Operators**), total number of static relations (**# SR**) are presented, number of identified static relationships (**Learnt SR**), number of additional static relations provided (**Additional SR**) , number of plans (**#Plans**), average number of actions per plan (**A/P**), **CPU-time** in milliseconds

Domain	# Operators	# SR	Learnt SR	Additional SR	# Plans	Avg. A/P	CPU-time (ms)
TPP	4	7	7	0	7	28	171
Zenotravel	5	4	6	2	4	24	109
Miconic	4	2	2	0	1	177	143
Storage	5	5	5	0	24	15	175
Freecell	10	19	13	0	20	60	320
Hanoi	1	0	1	1	1	60	140
Logistics	6	0	1	1	3	12	98
Driverlog	6	2	2	0	3	12	35
Mprime	4	7	7	0	10	30	190
Spanner	3	1	1	0	1	8	144
Gripper	3	0	1	1	1	14	10
Ferry	3	1	2	1	1	18	130
Barman	12	3	3	0	1	150	158
Gold-miner	7	3	1	0	13	20	128
Trucks	4	3	3	0	6	25	158

- Input plans generated using **Metric-FF planner** on randomly generated problems
- Implemented in Java, and run on a **Core 2 Duo/8GB** processor

Evaluation: Challenges

- For plan traces to exist there must be a comprehensive domain that should exist at least in the toy domain category, to evaluate the results of the system

- Traces only provide examples of valid operating states, so these cannot be used to change an invalid constraint to a valid constraint

- Traces may be inadequate to fully learn states and constraints (Grant, T. 2010)

- No information about predicates or initial, goal or intermediate state descriptions for the input example action sequences

Conclusion and Future Work

- We introduced ASCoL, an efficient and effective method for identifying static knowledge missed by domain models automatically acquired
- To extend our approach for considering static relations that involve more than two arguments
- To extend the approach for merging graphs of different pairs of arguments
- To identify heuristics for extracting useful information also from acyclic graphs Analysis and evaluation of the ASCoL with other constraint acquisition systems

Thank you