Domain Model Acquisition in the Presence of Static Relations in the LOP System

Peter Gregory
Digital Futures Institute,
School of Computing,
Teesside University, UK
p.gregory@tees.ac.uk

Stephen Cresswell
The Stationery Office
St. Crispins, Duke Street,
Norwich, NR3 1PD, UK
stephen.cresswell@tso.co.uk

Abstract

We present a new domain model acquisition algorithm, LOP, that induces static predicates by using a combination of the generalised output from *LOCM2* and a set of optimal plans as input to the learning system. We observe that static predicates can be seen as restrictions on the valid groundings of actions. Without the static predicates restricting possible groundings, the domains induced by *LOCM2* produce plans that are typically shorter than the true optimal solutions. LOP works by finding a set of minimal static predicates for each operator that preserves the length of the optimal plan.

1 Introduction

Modelling is well known as a bottleneck in the development of solutions to difficult combinatorial problems. The research field of automated modelling focuses on the task of constructing formal descriptions of problems automatically, often using solution data as input. Automated model acquisition is an active research area in constraint programming, general game playing and computer security (e.g. [O'Sullivan, 2010; Bessiere *et al.*, 2014; Björnsson, 2012; Aarts *et al.*, 2013]).

Domain model acquisition is automated modelling in a planning context: it is the problem of learning planning domain models from example data. The *LOCM* family of domain model acquisition systems [Cresswell *et al.*, 2009; Cresswell and Gregory, 2011; Cresswell *et al.*, 2013] learn planning domain models from collections of plans. For example, given the following two plans, a planning domain model acquisition system should learn operator schema of the type shown in Figure 1:

```
(drive-truck  truck2 driver1 loc3 loc1)
(load-truck   truck2 loc1 pkg1)
(drive-truck  truck2 driver1 loc1 loc3)
(unload-truck truck2 loc1 pkg1)

(load-truck   truck2 loc6 pkg3)
(drive-truck  truck2 driver2 loc6 loc2)
(drive-truck  truck1 driver1 loc6 loc4)
```

The *LOCM* family of algorithms only learn the dynamic aspects of the domain (i.e. state changes that occur due to action application). This is problematic since many domains use static relations to restrict the possible actions. Consider the Driverlog domain, where the road map is encoded as a binary static predicate. Figure 1 shows both the benchmark and *LOCM2* versions of the drive action. The conn predicate encodes the road map in the hand coded benchmark. This is not modelled in the induced drive action. Some of these static relations could be inferred given extra information about intermediate states. However, it is more desirable to be able to infer these static relations using only the minimal information available to the *LOCM* system, since intermediate state information may not always be available. In this paper, we extend the *LOCM2* system in order to detect static relations. The key assumption in our approach is that the input is drawn from optimal goal-directed plans. Assuming that *LOCM2* has detected the dynamics of the problem correctly, then if the induced plan is shorter, then this is a good clue to the fact that some static relation has gone undetected. Static conditions supporting the length of the input plans can then be hypothesised. We call our system *LOP* [Gregory and Cresswell, 2015] (standing for *LOCM* with Optimised Plans).

```
(:action drive-truck-benchmark
 :parameters (?t - truck ?d - driver ?l1 - loc ?l2 - loc)
 :precondition (and
                    (at ?t ?l1)
                    (conn ?l1 ?l2)
                    (driving ?d ?t))
 :effect (and
                    (not (at ?t ?l1)) (at ?t ?l2)))

(:action drive-truck-locmii
 :parameters (?t - truck ?d - driver ?l1 - loc ?l2 - loc)
 :precondition (and
                    (at ?t ?l1)
                    (driving ?d ?t ?l1))
 :effect (and
                    (not (at ?t ?l1)) (at ?t ?l2)
                    (not (driving ?d ?t ?l1))
                    (driving ?d ?t ?l2)))
```

Figure 1: The Driverlog drive action as encoded in the benchmark domain (top) and by the *LOCM2* system (bottom).

Related Work

Within the planning literature, there are many domain model acquisition systems. These systems each have varying levels of detail in their input observations. *LOCM*-derived systems use a minimal amount of input (only plan traces) whereas most other systems use predicates, initial and goal states and possibly intermediate states. The compromise is that

Algorithm 1 The Preserve Optimality testing algorithm.

Require: $\mathcal{M}$: an output domain model from *LOCM2*
Require: $\mathcal{T}$: a set of *LOCM2* generated problems
Require: P : a subset of the parameters for each operator
  function preserveOptimality
    for $t \in \mathcal{T}$ **do**
      optP $\leftarrow$ optimal input plan for t
      dom $\leftarrow \mathcal{M}$ plus statics defined by P
      prob $\leftarrow t$ plus ground statics defined by optP
      optT $\leftarrow$ optimal solution to dom, prob
      if length(optT) $<$ length(optP) **then**
        return false
      end if
    end for
    return true
  end function

the target language in *LOCM* is simpler than many other systems. The *Opmaker2* system [McCluskey *et al.*, 2009; Richardson, 2008] learns models in the target language of OCL [McCluskey and Porteous, 1997] and requires a partial domain model, along with example plans as input. The ARMS system [Wu *et al.*, 2007], can learn STRIPS domain models with partial or no observation of intermediate states in the plans, but does at least require predicates to be declared.

The LAMP system [Zhuo *et al.*, 2010] can target PDDL representations with quantifiers and logical implications. Systems that learn planning models in the presence of noisy and incomplete data [Mourao *et al.*, 2012] have also been studied. Other types of learning include [Mehta *et al.*, 2011] considering the task of learning a single state space with no input plans, but with a system by which an oracle can validate plan hypotheses. A form of transfer learning has been considered [Zhuo *et al.*, 2011] where action schema are constructed via a combination of analysing exisiting domains and using web queries to match operator names. The closest system to *LOP*, specifically designed to detect static information is the recent ASCOL system [Jilani *et al.*, 2015].

2 Static Relations in Planning

Static relations are often thought of by their semantic interpretations: road maps in Driverlog, successor relations in Zenotravel and Freecell, for example. This is seen in Figure 1 for the Driverlog domain. We now present an alternative purely syntactic interpretation of static relations that is critical to our work.

Defining Static Relations

Static relations can be seen as restrictions on the groundings of each operator. Instead of thinking of how the objects are related, we now think of which combinations of ground operator parameters are valid. The static relations in a domain can be defined as, for each operator, a table of all the valid groundings for that operator. The static relations for any domain can be encoded as a single relation per operator.

This observation allows us to construct our hypothesis space of potential static relations. Each operator can be seen to have a static relation associated with it, and we have to identify the minimal subset of parameters that permit exactly the valid groundings of an operator. We can now present definition of the minimal static relation identification problem:

Definition 1 (Minimal Static Relation Identification)**.** Given an operator template:

$$\mathrm{op}(p_1, ..., p_n)$$

We define a minimal static relation as the subset $P_m \subseteq p_1, ..., p_n$ such that all $p_m \in P_m$ are members of static preconditions of op.

Thus, this problem is simply to identify the parameters which play a part in the static relation.

Universal Static Relations

Some statics are defined for specific instances and are different in different instances depending on the problem objects. These include the road maps in Driverlog and the locations of the hoists in Depot, for example. In contrast to these types of static relations, there are also those that hold in all instances. For example, in Peg Solitaire, all instances use the same underlying board. In Zeno Travel, the static relation that encodes the successor relation for the fuel levels is the same in all problem instances. If a static relation holds between all instances, we call that a *universal* static relation.

Issues in Detecting Statics

The key issue in detecting statics using plan traces as input is that there are domains which look very similar, but which vary in the static relations. For example, Blocksworld and Freecell, in which the goal in both is to rearrange stacks of objects. In Blocksworld there are no static relations, in Freecell there are static relationships between the ranks and colours of the cards. In order to address this issue we use additional information about the plans, other than simply the plan traces. Namely, we identify a set of optimal plans to be used in the learning phase. In the following sections, we detail exactly how we use optimal plans to discover static relations.

3 The *LOP* Algorithm

We now present the *LOP* algorithm that we use to detect static relations. *LOP* has the assumption that *LOCM2* returns the correct domain model dynamics. The algorithm is sketched as follows:

1. Run *LOCM2* on both the optimal and suboptimal training data, constructing the *LOCM2* domain model.
2. Identify minimal subsets of each action's parameters that 'preserve' the optimal length of all plans.
3. Split each relation into a minimal partition that preserve optimal plan lengths.
4. Identify the universal static relations.
5. Return the valid ground relations and the problem-specific templates.

3.1 Preserving Optimality

All stages of the *LOP* algorithm rely on testing whether or not optimality is preserved for a set of input plans. We now define exactly what we mean by *preserving optimality*. Consider the following optimal input plan:

Algorithm 2 Minimal static relation hypothesis algorithm.

```
function MSR
    for o : operators do
        msr(o) ← MSRo(o)
    end for
    return msr
end function
function MSRo(o : operator)
    minS ← parameters(o)
    for p ∈ minS do
        minS′ ← (minS \ {p})
        if preserveOptimality(minS′) then
            minS ← minS′
        end if
    end for
    return minS
end function
```

```
(drive truck loc1 loc2)
(drive truck loc2 loc3)
```

Now, consider the drive action and PDDL problem fragment induced by *LOCM2*:

```
(:action drive
 :parameters (?t - truck ?l1 ?l2 - loc)
 :precondition (at ?t ?l1)
 :effect (and (not(at ?t ?l1))
              (at ?t ?l2))
)
(:init (at truck loc1))
(:goal (at truck loc3))
```

Note that the dynamics of the drive action are correct: the truck moves from the start location to the destination location. However, if we solve this output using an optimal planner we find the following plan:

```
(drive truck loc1 loc3)
```

This plan is shorter than the input plan which we *knew* to be optimal. Therefore, we say that the drive action does not *preserve optimality*. In order to restore the optimality we add static relations both in the action preconditions and the corresponding ground predicates in the initial state. For example, the set of all parameters yields the following action and problem fragment:

```
(:action drive
 :parameters (?t - truck ?l1 ?l2 - loc)
 :precondition (and (at ?t ?l1)
                    (drive_static ?t ?l1 ?l2)
 :effect (and (not(at ?t ?l1))
              (at ?t ?l2))
)
(:init (at truck loc1)
       (drive_static truck1 loc1 loc2)
       (drive_static truck1 loc2 loc3)
)
(:goal (at truck loc3))
```

Algorithm 1 presents the algorithm for testing if optimality is preserved for a particular parameter set of the operators. In essence, it checks that each input plan preserves optimality for that parameter set. The algorithm for discovering minimal static relations (or MSRs) is presented in Algorithm 2 which attempts to remove each parameter in turn from a candidate relation, adding it back in if it leads to shorter optimal solutions. Depending on the current MSR hypotheses for each operator, Algorithm 2 may return different results. For this reason, the algorithm is run repeatedly until a fix-point is reached, when the MSRs do not change following an iteration.

Algorithm 3 Minimal static relation splitting algorithm. In this algorithm rank refers to the rank of the partition (the size of the largest block minus the total number of blocks).

```
function MSR_Split(P : a partition of the MSR)
    if preserveOptimality(P) = false then
        return ⊥
    end if
    min ← P
    for P′ ∈ refinement(P) do
        minP′ ← MSR_Split(P′)
        if evaluate(minP′) < evaluate(min) then
            min ← minP′
        end if
    end for
    return min
end function
function evaluate(P : a partition of the MSR)
    if P = ⊥ then
        return ∞
    else
        return rank(P)
    end if
end function
```

3.2 Splitting The Static Relation

Once a minimal static relation is found for each operator, we test if it can be divided into multiple smaller static relations. As an example, the Freecell domain has the operator: move(?card, ?cols, ?ncols, ?cells, ?ncells) where there is a static relation between cols and ncols and between cells and ncells. The minimal static relation will be across all four of these parameters, where it would clearly be beneficial to identify two separate relations. In order to split the relation, we need to check the possible partitions of the parameters in the minimal static relation. Checking every partition is computationally prohibitive for actions with a large number of parameters. For this reason, we propose a pruning technique based on dominated partition refinements. A description of the algorithm is given in Algorithm 3.

3.3 Testing for Universality

As a final step, we test the static relations for universality. If a static relation is universal, then all instances share the same groundings for that relation. Intuitively, we collect all of the instances of the static relations found in the earlier stages of *LOP* and combine the groundings across instances. When we combine these ground instances, if the combined relation preserves optimality then we accept the relation as universal.

Domain	Operators			Input Plans		*LOCM2*		*LOP* Optimal			*LOP* Satisficing		
	#	#Static	#Univ.	#Opt	#Sub	Valid	#St.	#St.	#Uni.	#Er.	#St.	#Uni.	#Er.
AoP Freecell	8	3	3	60	60	✓	1	2	2	0	2	2	0
Blocks	4	0	0	28	35	✓	0	0	0	0	0	0	0
Depot	4	2	0	17	19	✓	2	0	0	0	0	0	0
Driverlog	6	2	0	13	15	✓	0	2	1	1	2	1	1
Freecell[1]	10	10	10	14	16	✓*	8	-	-	-	3	1	6
Grid	5	2	2	12	15	✓	0	2	2	0	2	2	0
Gripper	3	0	0	7	20	✓	0	0	0	0	0	0	0
Logistics	6	1	0	20	28	✓	1	0	0	0	0	0	0
Miconic	4	4	2	141	92	✓	0	2	0	2	2	0	2
Mystery[2]	3	3	0	16	19	✓*	0	3	0	0	3	0	0
Parking	4	0	0	12	20	✓	0	0	0	0	0	0	0
Peg Solitaire	3	2	2	16	20	✓	0	2	2	0	2	2	0
Rovers	9	9	0	15	15	✗	-	-	-	-	-	-	-
Satellite	5	4	0	17	18	✗	-	-	-	-	-	-	-
Scanalyzer	4	4	4	15	13	✗	-	-	-	-	-	-	-
Sokoban	3	3	0	18	20	✓	0	3	3	0	3	3	0
Storage	5	5	2	15	14	✓	0	5	2	0	5	2	0
TPP	4	4	4	16	25	✗	-	-	-	-	-	-	-
Visitall	2	2	2	10	20	✓*	0	2	2	0	2	2	0
Zenotravel[2]	5	3	3	12	13	✓*	0	3	3	0	3	3	0

Table 1: Table of results running *LOP* on a collection of benchmark domains. Headings refer to # (number of operators) #Static (number of operators with static precondition) #Univ. (number of operators with universal preconditions) #Opt (number of optimal training plans) #Sub (number of suboptimal training plans) Valid (did *LOCM2* output complete dynamics) a star means the input had to be modified to meet the *LOCM2* assumptions #St. (number of operators for which statics are found) #Uni. (number of operators for which universal statics were found) #Er. (the number of errors made in all stages of *LOP*).

4 Evaluation

Our experimental setup is as follows: we used the Fast Downward [Helmert, 2006] planner, with the LMCut heuristic [Helmert and Domshlak, 2009] with a fifteen minute cutoff to find optimal plans to as many of the benchmark planning problems as possible. These form the optimal training plans needed for the static analysis. In our first experiment we have a set of plans which we know to be optimal and we use these plans for the *LOP* analysis. In our second experiment, we treat goal-directed plans that are not proven to be optimal the same way as we treat optimal plans in our previous analyses. For this, we use the LAMA planner [Richter and Westphal, 2010], with a 120 second time cutoff.

Table 1 shows the results of both experiments. Generally the results are very positive: of the domains that have static relations and had valid dynamic output from *LOCM2*, nine out of twelve had their static relations discovered error free. *LOP* can be seen as a *LOCM*-like system in that it produces an overly general result when incorrect. We now present a discussion of some of the more interesting results (more detail on individual domains is presented in [Gregory and Cresswell, 2015]). In the Driverlog domain, the structure of both statics are detected correctly. These are the roads that the trucks drive on and the path that the drivers can walk on. The paths are incorrectly identified as universal statics. We are unsure whether this happens because the path maps are completely consistent with the input data, or that the path maps are very dense, leading to many alternative route.

Zeno Travel encodes the possible transitions of the fuel level of the planes as universal static relations. These restrict the groundings of the fly, zoom and refuel operators. *LOP* successfully induces these relations in their scope and universality, and can be said in a way to improve on the original domain. In the original domain, the zoom operator, $(\mathrm{zoom\ plane\ loc1\ loc2\ f3\ f2\ f1})$ encodes that the plane in question uses two units of fuel. The preconditions encode this as the two predicates $(\mathrm{next\ f3\ f2}), (\mathrm{next\ f2\ f1})$. The static relation identified by *LOP* is between the two parameters $\mathrm{f3}$ and $\mathrm{f1}$ and ignores $\mathrm{f2}$. This may initially be viewed as a mistake, but on closer inspection, the universal relation identified encodes the $\mathrm{n\_plus\_two}$ relation. The parameter $\mathrm{f2}$ is, in fact, redundant. Summing up, our empirical analysis demonstrates that the *LOP* algorithm is effective at discovering static relations for a wide range of problems. We have also demonstrated domains in which *LOCM2* fails to discover the correct domain dynamics, possibly suggesting new direction.

5 Conclusions

Domain model acquisition is useful whenever a knowledge engineer has access to a controllable system that acts, but for which the structure of those actions has not been formally specified. We have presented a solution to the problem of domain model acquisition under the presence of static relations. The *LOP* algorithm relies on the quality of the input plans serving as a guide to refining an overly general model generated by the *LOCM2* system.

References

[Aarts *et al.*, 2013] Fides Aarts, Joeri De Ruiter, and Erik Poll. Formal Models of Bank Cards for Free. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 461–468. IEEE, March 2013.

[Bessiere *et al.*, 2014] Christian Bessiere, Remi Coletta, Abderrazak Daoudi, Nadjib Lazaar, Younes Mechqrane, and El Houssine Bouyakhf. Boosting Constraint Acquisition via Generalization Queries. In *European Conference on Artificial Intelligence*, 2014.

[Björnsson, 2012] Y Björnsson. Learning Rules of Simplified Boardgames by Observing. In *European Conference on Artificial Intelligence*, pages 175–180, 2012.

[Cresswell and Gregory, 2011] Stephen Cresswell and Peter Gregory. Generalised Domain Model Acquisition from Action Traces. In *International Conference on Automated Planning and Scheduling*, pages 42 – 49, 2011.

[Cresswell *et al.*, 2009] Stephen Cresswell, Thomas Leo McCluskey, and Margaret Mary West. Acquisition of object-centred domain models from planning examples. In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *International Conference on Automated Planning and Scheduling*. AAAI, 2009.

[Cresswell *et al.*, 2013] SN Cresswell, TL McCluskey, and MM West. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2):195 – 213, 2013.

[Gregory and Cresswell, 2015] Peter Gregory and Stephen Cresswell. Domain Model Acquisition in the Presence of Static Relations in the LOP System. In *International Conference on Automated Planning and Scheduling*, pages 97–105, 2015.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *International Conference on Automated Planning and Scheduling*, pages 162–169, 2009.

[Helmert, 2006] Malte Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Jilani *et al.*, 2015] Rabia Jilani, Andrew Crampton, Diane E. Kitchin, and Mauro Vallati. Ascol: A tool for improving automatic planning domain model acquisition. In *AI*IA 2015, Advances in Artificial Intelligence - XIVth International Conference of the Italian Association for Artificial Intelligence, Ferrara, Italy, September 23-25, 2015, Proceedings*, pages 438–451, 2015.

[McCluskey and Porteous, 1997] Thomas Lee McCluskey and Julie Porteous. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence*, 95(1):1–65, 1997.

[McCluskey *et al.*, 2009] T. L. McCluskey, S. N. Cresswell, N. E. Richardson, and M. M. West. Automated acquisition of action knowledge. In *International Conference on Agents and Artificial Intelligence (ICAART)*, pages 93–100, January 2009.

[Mehta *et al.*, 2011] Neville Mehta, P Tadepalli, and Alan Fern. Autonomous learning of action models for planning. In *Advances in Neural Information Processing Systems*, 2011.

[Mourao *et al.*, 2012] K Mourao, LS Zettlemoyer, Ronald P A Petrick, and Mark Steedman. Learning STRIPS Operators from Noisy and Incomplete Observations. In *Uncertainty in Artifical Intelligence*, pages 614 – 623, 2012.

[O'Sullivan, 2010] B O'Sullivan. Automated Modelling and Solving in Constraint Programming. *AAAI Conference on Artificial Intelligence*, pages 1493–1497, 2010.

[Richardson, 2008] N. E. Richardson. *An Operator Induction Tool Supporting Knowledge Engineering in Planning*. PhD thesis, School of Computing and Engineering, University of Huddersfield, UK, 2008.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

[Wu *et al.*, 2007] Kangheng Wu, Qiang Yang, and Yunfei Jiang. ARMS: An automatic knowledge engineering tool for learning action models for AI planning. *The Knowledge Engineering Review*, 22(2):135–152, 2007.

[Zhuo *et al.*, 2010] Hankz Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li. Learning complex action models with quantifiers and logical implications. *Artificial Intellgence*, 174:1540–1569, December 2010.

[Zhuo *et al.*, 2011] HH Zhuo, Qiang Yang, Rong Pan, and Lei Li. Cross-Domain Action-Model Acquisition for Planning via Web Search. In *International Conference on Automated Planning and Scheduling*, pages 298–305, 2011.