Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Domain Model Acquisition in Domains with Action Costs

Peter Gregory and **Alan Lindsay**
Digital Futures Institute,
School of Computing,
Teesside University, UK
firstinitial.lastname@tees.ac.uk

Abstract

This paper addresses the challenge of automated numeric domain model acquisition from observations. Many industrial and commercial applications of planning technology rely on numeric planning models. For example, in the area of autonomous systems and robotics, an autonomous robot often has to reason about its position in space, power levels and storage capacities. It is essential for these models to be easy to construct. Ideally, they should be automatically constructed.

Learning the structure of planning domains from observations of action traces has produced successful results in classical planning. In this work, we present the first results in generalising approaches from classical planning to numeric planning. We restrict the numeric domains to those that include fixed action costs. Taking the finite state automata generated by the LOCM family of algorithms, we learn costs associated with machines; specifically to the object transitions and the state parameters. We learn action costs from action traces (with only the final cost of the plans as extra information) using a constraint programming approach. We demonstrate the effectiveness of this approach on standard benchmarks.

Introduction

In this paper, we demonstrate a new domain model acquisition system that learns action costs in planning domains. Many commercial and industrial applications of automated planning technology rely on numeric state variables. For example, in constructing policies for the use of batteries (Fox, Long, and Magazzeni 2011), the construction of machine tool calibration plans (Parkinson et al. 2012) and spacecraft orbit planning (Surovik and Scheeres 2015). Within both board games and video games, numeric models are crucial in order to encode scoring systems, resource use, etc. Within interactive narrative settings, numeric variables represent varied structures, such as strength of relationships in social networks (Porteous, Charles, and Cavazza 2013; 2015) and the level of tension (Porteous et al. 2011) within a certain scene.

Modelling is the process of specifying a problem in a formal language, thus making it amenable for solving using general algorithmic techniques. Modelling is considered to be a bottleneck in the process of tackling combinatorial problems, due to the skills required to develop these models. Model generation is a crucial process within the planning community. Systems have been developed to aid a domain modeller, similar to Integrated Development Environments for use by software engineers. For example, the GIPO (Simpson, Kitchin, and McCluskey 2007) system, itSIMPLE (Vaquero et al. 2007) system and KIWI (Wickler, Chrpa, and McCluskey 2014) system. These modelling tools are useful for rapid development of domains by an experienced domain modeller.

Another avenue of research to aid in the modelling process is based on learning models from example solutions: namely that of domain model acquisition, which is the core topic of this work.

Domain Model Acquisition

Within this work, when we refer to domain model acquisition, we refer to the learning of a planning domain from example data that includes sequences of state transitions. There is interest in applying domain model acquisition across a range of research and application areas. For example within the business process community (Hoffmann, Weber, and Kraft 2012) and space applications (Frank et al. 2011). An extended version of the *LOCM* domain model acquisition system (Cresswell, Mccluskey, and West 2009) has also been used to help in the development of a puzzle game (Ersen and Sariel 2015) based on spatio-temporal reasoning. Web Service Composition is another area in which domain model acquisition techniques have been used (Walsh and Littman 2008). More generally, the area of learning action models from observation has been studied in robotics (Cakmak, Chao, and Thomaz 2010; Baranes and Oudeyer 2013), although typically this is in the context of reinforcement learning. In contrast to this, in the planning community, factored symbolic models are typically learnt.

We address the challenge of domain model acquisition in the presence of action costs. We aim to remain as close to the ideals of the *LOCM* family of algorithms (i.e. learning from minimal amounts of input) as is feasible. It seems reasonable to think that some knowledge of plan cost must be known in order to derive individual action costs. Provided this premise, we endeavour to produce hypotheses given a collection of plans and only the final value of the accu-

```
(define (domain transport)
  (:types
        location target locatable - object
        vehicle package - locatable
        capacity-number - object
  )
  (:predicates
     (road ?l1 ?l2 - location)
     (at ?x - locatable ?v - location)
     (in ?x - package ?v - vehicle)
     (capacity ?v - vehicle ?s1 - capacity-number)
     (capacity-predecessor ?s1 ?s2 - capacity-number)
  )
  (:functions
     (road-length ?l1 ?l2 - location) - number
     (total-cost) - number
  )
  (:action drive
    :parameters (?v - vehicle ?l1 ?l2 - location
                 ?c - capacity)
    :precondition (and
        (at ?v ?l1)
        (capacity ?v ?c)
        (road ?l1 ?l2)
      )
    :effect (and
        (not (at ?v ?l1))
        (at ?v ?l2)
        (increase (total-cost) (road-length ?l1 ?l2))
      ))
 (:action pick-up
    :parameters (?v - vehicle ?l - location
                 ?p - package ?s1 ?s2 - capacity-number)
    :precondition (and
        (at ?v ?l)
        (at ?p ?l)
        (capacity-predecessor ?s1 ?s2)
        (capacity ?v ?s2)
      )
    :effect (and
        (not (at ?p ?l))
        (in ?p ?v)
        (capacity ?v ?s1)
        (not (capacity ?v ?s2))
        (increase (total-cost) 1)
      ))
  (:action drop
    :parameters (?v - vehicle ?l - location
                 ?p - package ?s1 ?s2 - capacity-number)
    :precondition (and
        (at ?v ?l)
        (in ?p ?v)
        (capacity-predecessor ?s1 ?s2)
        (capacity ?v ?s1)
      )
    :effect (and
        (not (in ?p ?v))
        (at ?p ?l)
        (capacity ?v ?s2)
        (not (capacity ?v ?s1))
        (increase (total-cost) 1)
      )))
```

Figure 1: Transport Domain: An example of a domain with action costs. Note that the pickup and drop operators have a fixed cost, whereas the drive operator has a cost defined by the problem instance. This domain provides a running example through the paper.

mulated cost variable. In order to learn the numeric action costs we use a constraint programming approach, in order to search for consistent sets of action costs across multiple plans. Models of the state transition systems will be learnt without action costs, and then each plan can be seen as a linear equation over the transitions within that plan. Each plan can be viewed as a set of constraints over its final action cost, and actions between certain subsets of the plans are required to have consistent valuations.

Background

Domain model acquisition is the problem of learning a formal domain model of a system from some form of input data. There are three main ways in which domain model acquisition systems vary: the nature of the input data they receive, the expressiveness of the target language and the query system by which they acquire the input data.

The ideal domain model acquisition system would be able to target the most expressive modelling language, using the minimal amount of available input data. The *LOCM*, *LOCM2* and *LOP* systems demonstrate the progress that has been made in learning models in increasingly complex modelling languages using a minimal amount of information. A key open question concerns the extent to which this process can be continued: what is the most expressive modelling language that can be targeted using minimal input? Since numeric planning is of critical importance to users of planning technology, this work is focssed on developing techniques to learn numeric planning models.

The domain model acquisition system that we introduce in this paper uses the *LOCM2* and *LOP* systems as preprocessing steps. In order to describe these two systems, and also our own, we introduce the Transport domain as a running example.

Running Example: The Transport Domain

Figure 1 shows the Transport domain, which is a typical logistics-type planning domain. It has three operators, each with an action cost. There is a unit cost of picking up or dropping off items from the truck. The cost of driving between locations is determined by the road length between those two places. Note that the domain is slightly modified from the benchmark domain. An extra parameter has been added to the drive action, along with a precondition in the same action. This parameter represents the current capacity of the truck, and the precondition simply ensures that the capacity parameter is indeed the capacity of the truck. This change does not alter the search space in any way, but as noted in (Gregory and Cresswell 2015), in order for *LOCM* and *LOCM2* to correctly learn the domain dynamics, the input plans are modified.

The *LOCM* Algorithms

Typical strategies for the acquisition of domain models from information-sparse input sets make restrictive assumptions about the form of the output domain. For example, the *LOCM* family of systems (Cresswell, Mccluskey, and West 2009; Cresswell and Gregory 2011; Gregory and Cresswell 2015) operate with the assumption that each object is represented by a parameterised finite state machine.

Their distinguishing feature is that they learn the domain dynamics from very sparse information. They use no other information besides the action sequences, such as those shown in Figure 2 for the Transport domain, i.e. no information about types, predicates, initial or final states. This is possible because of restricting assumptions about the form of the domain model. The key assumptions of the *LOCM* family of algorithms are: that the behaviour of each object is

```
PLAN 1: COST 143
drive  truck2 loc3 loc1 cpy3
pickup truck2 loc1 pkg1 cpy2 cpy3
drive  truck2 loc1 loc3 cpy2
pickup truck2 loc3 pkg3 cpy1 cpy2
pickup truck1 loc2 pkg4 cpy2 cpy3
drive  truck2 loc3 loc1 cpy1
drop   truck2 loc1 pkg1 cpy1 cpy2
drive  truck2 loc1 loc3 cpy2
drive  truck2 loc3 loc2 cpy2
pickup truck1 loc2 pkg2 cpy1 cpy2

PLAN 2: COST 151
pickup truck2 loc6 pkg1 cpy3 cpy4
pickup truck2 loc6 pkg3 cpy2 cpy3
drive  truck2 loc6 loc2 cpy2
pickup truck1 loc6 pkg4 cpy1 cpy2
drive  truck1 loc6 loc2 cpy1
```

Figure 2: Two plans from the Transport domain. Collections of plans form the input to many domain model acquisition systems.

described by one (or more in *LOCM2*) finite state machines whose arcs are the transitions that change the state of the object. And crucially, each of these transitions can only occur once in an object's FSM.

An example of the output of *LOCM* is shown in Figure 3. There are three state machines shown: one is the zero state machine, and one each for the Truck and the Package types. The meaning of the Package FSM is that a package can be in two states (in and out of a truck). Within a truck, it has an association with a truck, and when outside of a truck it has an association with a location (represented by the state parameters in square brackets). The Truck type is represented by a single state FSM, where the only interesting structure is in the state parameters. The zero machine is the FSM induced by adding a special zeroth parameter object to every action, and it represents enforced patterns in operator application.

The *LOCM* and *LOCM2* algorithms only learn the dynamic aspects of the domain (i.e. state changes that occur due to action application). This is not sufficient since many domains use static relations to restrict the possible actions. Consider the Transport domain, where the road map is encoded as a binary static predicate. The *LOP* system (Gregory and Cresswell 2015) learns static relations by comparing optimal input plans with the optimal plans found using the induced domain model of *LOCM2*. Assuming that *LOCM2* has detected the dynamics of the problem correctly, then if the induced plan is shorter, then this provides evidence to support the hypothesis that some static relation has gone undetected. The learnt model can be translated into the STRIPS fragment of PDDL. Each FSM state is represented by a PDDL predicate having its associated object as first argument, with further arguments formed from state parameters. Operators are constructed from the transitions and their parameters, using the binding constraints discovered between action parameters and state parameters. Static relations are encoded using the output from *LOP*.

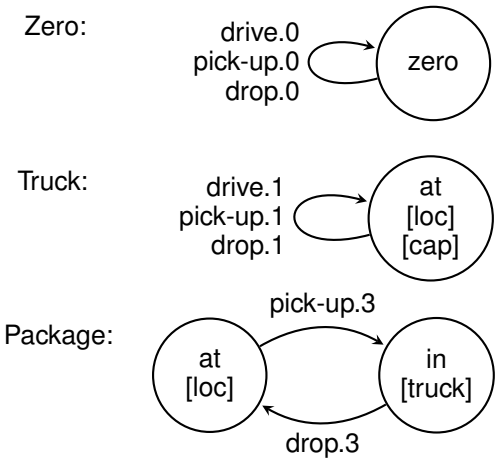

Figure 3: The finite state machines derived by *LOCM* for the truck type in the transport domain for the two interesting object types: package and truck. The truck state machine has a single state, with two state parameters for the location of truck and the capacity of the truck.

Modelling Action Costs

Within this section, we will introduce new techniques that allow the acquisition of numeric domain models from example transition traces. We will focus on doing this in an information-sparse setting, similar to the *LOCM* family of algorithms.

Action costs in planning

One common restriction of numeric variables in planning is that of action costs, mainly due to the influence of SAS+ (Bäckström and Nebel 1995) based planners. This restriction means that each ground action has a constant cost, and that the only numeric variable accumulates the sum of these individual action costs over the length of the plan. This accumulated value is the optimisation variable.

Action costs can be seen as a single constant value attributed to each ground action; indeed, this is how most formal definitions of planning problems proceed. However, in practice domain engineers may define the action cost through an accumulated sum of constant functions defined over subsets of operator parameters. For example, in the Transport domain, the cost of the drive action is defined by the distance between the start and end locations. This situation is described formally in Table 1. There are four operator arguments for the drive operator, and it is possible in the domain to model a constant function over any subset of the operator parameters. Therefore, the overall operator cost is determined by the sum of all of those contributing functions. Table 1 enumerates all templates and identifies the contributing ones for the drive action: namely the set containing the start and destination locations.

A domain model which satisfies this template must provide the machinery to define the cost function based on these

Template	Parameters	Contributes
$\{\}$	$\{\}$	0
$\{1\}$	$\{?v\}$	0
$\{2\}$	$\{?l1\}$	0
$\{3\}$	$\{?l2\}$	0
$\{4\}$	$\{?c\}$	0
$\{1,2\}$	$\{?v,?l1\}$	0
$\{1,3\}$	$\{?v,?l2\}$	0
$\{1,4\}$	$\{?v,?c\}$	0
$\{2,3\}$	$\{?l1,?l2\}$	1
$\{2,4\}$	$\{?l1,?c\}$	0
$\{3,4\}$	$\{?l2,?c\}$	0
$\{1,2,3\}$	$\{?v,?l1,?l2\}$	0
$\{1,2,4\}$	$\{?v,?l1,?c\}$	0
$\{1,3,4\}$	$\{?v,?l2,?c\}$	0
$\{2,3,4\}$	$\{?l1,?l2,?c\}$	0
$\{1,2,3,4\}$	$\{?v,?l1,?l2,?c\}$	0

Table 1: Table of contribution for each parameter template of the drive operator in the Transport domain. The only selected template is $\{2,3\}$, which corresponds to the start and destination locations.

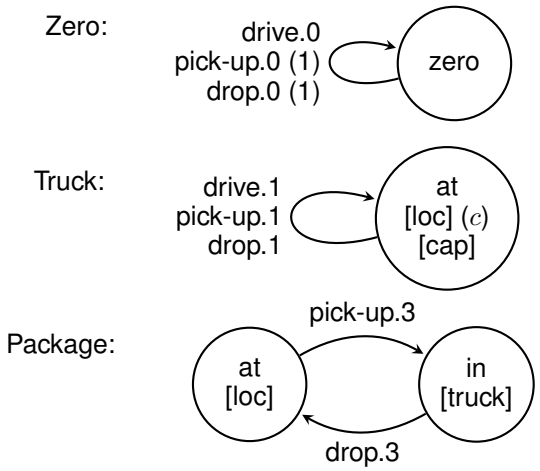

Figure 4: The finite state machines derived by *LOCM* annotated by numeric weights that refer to the cost of particular transitions in the state machines. The zero machine now has a cost associated with drop and pick-up, and the truck has a cost associated with the loc state parameter.

parameters (e.g., a function `cost-of-drive-2-3 ?l1 ?l2`) corresponding to `road-length` in the original domain. For each template identified, a set of ground instantiations of these functions must be provided in the initial state of each problem instance. For example, the drive template identified would require an initial state to contain assignments to the `cost-of-drive-2-3` function for all valid combinations of locations.

When searching for the most appropriate definition of the action costs underlying a domain, we define our hypothesis space as the space of all combinations of templates for each operator. In general, this is too large a space to search in: the number of templates scales exponentially as the number of parameters increases in the operators. We will refine this later using the structure of the *LOCM*-derived state machines.

Numeric State Machine Representation

It is possible to learn propositional action models in the *LOCM* family of algorithms because of assumptions made about the underlying model, as discussed in earlier sections. An important problem in creating a numeric domain model acquisition system is therefore identifying the most appropriate formalism required to model numeric domains. We propose a model that extends the finite state automata of *LOCM* to automata with numeric weights on the important features (i.e. the object transitions and the state parameters).

Figure 4 shows the desired output from our learning process. Figure 4 shows the state machines for the same object types as in Figure 3 but now labelled with costs. Costs are labelled in rounded brackets, and can be seen on the zero machine on the object transitions, and on the truck machine on the 'loc' state parameter. The meaning of a cost on a transition in the zero machine is that the operator associated with that transition costs the indicated amount for all groundings. This is reflected in the textual description of the domain, in which both pick-up and drop have an action cost of 1. The cost of driving is encoded as the distance between the start and destination. The relationship between start and destination is represented by the change in the loc state parameter in the truck state machine when a drive action occurs. Therefore, the loc state parameter is labelled with a variable cost c, which takes a different constant value, depending on the grounding of the operator.

We now describe how we can learn these domain models by using a constraint programming approach, which we name *NLOCM* (short for *Numeric LOCM*).

The *NLOCM* Constraint Model

The *NLOCM* system learns action costs by modelling the task as a constraint programming problem. In order to discuss this model, we introduce a formalisation of the problem input. As stated previously, our input is a collection of plans. We denote the input as Π, and for P input plans, we have:

$$\Pi = [\pi^1, ..., \pi^P]$$

We also define a cost function over these plans, based on the input costs, and a length function, based on the number of ground actions in a plan.

$$c(\pi^i) = \text{input cost} \qquad (1 \leq i \leq P)$$

$$l(\pi^i) = \text{input length} \qquad (1 \leq i \leq P)$$

We use the subscript to refer to ground actions within each plan, in the following way:

$$\pi^i = [\pi^i_1, ..., \pi^i_{l(\pi^i)}]$$

And finally, we can define the set of all observed ground actions as the union of all of the actions seen in all plans:

$$\mathcal{A} = \bigcup_{\pi \in \Pi} \{\pi_1, ..., \pi_{l(\pi)}\}$$

We also denote by $\mathcal{A}_O \subseteq \mathcal{A}$ the set of ground actions observed of the operator O. We now describe our constraint model in a number of stages, before describing the search strategies that we use in order to learn domain models.

Base Model

In the base model, we encode ground action costs and pose constraints over the input plans. Therefore, for each ground action, we define an integer variable:

$$A_{1..|\mathcal{A}|} : Integer \tag{V1}$$

Then for each input plan, we encode a linear constraint, such that the sum of the costs of its individual actions is equal to the cost of the entire plan:

$$\sum_{i=1}^{l(\pi^p)} (\pi^p_i) = c(\pi^p) \qquad (\pi^p \in \Pi) \tag{C1}$$

Any two identical ground actions in the input plans will be represented by the same variable in the constraint model. This is true even between different input plans, so long as they are drawn from the same state space (i.e. the same planning problem). The base model allows us to learn a model of the ground action costs for a specific set of problem instances. This is useful if all further planning will occur in the same state space. However, in domain model acquisition, we are interested in deriving the more general domain structure. Ideally, we want a finer grained model in which we know exactly how the action cost for each operator is calculated. This finer grained model will explore a subset of the contributors to action costs that we described in the background section. We now describe this extension to our model.

Encoding Operator and Ground Action Templates

For each operator, O, we define a set of templates, T_O over its arguments that we have determined as interesting. As discussed previously, the choice of template is driven pragmatically from the structure of our target formalism, and is informed by the output of *LOCM2* and *LOP*. We can define the templates generally as follows:

$$T_O \subseteq \mathbb{P}(\text{args}(O)) \qquad (O \in \mathcal{O})$$

We also define $T_A = T_O$, where A is an instantiating action of operator O. Now, given our templates for each operator, we can define a set of variables to represent whether they are used in the model. For each operator, O, we add the following variables:

$$\text{Active}^O_\tau : [0, 1] \qquad (\tau \in T_O) \tag{V2}$$

Each template within T_O defines a set of arguments that are assigned a constant cost in each problem instance. For example in the transport domain example, setting the template variable $\text{Active}^{\text{drive}}_{\{2,3\}} = 1$ means that, for the drive operator, the output domain will have a cost contribution from the second and third parameters. When trying to discover the contributing templates, however, we need to reason about the specific costs of actions in the input plans. Therefore, we require variables to represent ground template values. For each operator, O, we introduce the following variables:

$$\text{Ground}^A_\tau : Integer \quad (A \in \mathcal{A}_O, \tau \in T_O) \tag{V3}$$

Since in our model, we assume that these contributors are the only contributors to the cost of the operator, we can state that the ground action cost is the sum of the corresponding action templates. For each operator, O:

$$A = \sum_{\tau \in T_A} \text{Ground}^A_\tau \qquad (A \in \mathcal{A}_O) \tag{C2}$$

A constraint is added such that if a template is inactive, all of its groundings are set to zero (i.e. they do not contribute to the cost of the action):

$$(\text{Active}^O_\tau = 0) \implies \sum_{A \in \mathcal{A}_O} \text{Ground}^A_\tau = 0 \tag{C3}$$

Thus, the variable, Active^O_τ, controls whether the template, τ, can be used to explain plan cost. Given that we have an adequate selection of templates, we can now search for satisfying action cost configurations. In practice, there can be many assignments that satisfy the constraints that we have specified. We now discuss a heuristic method that we use in order to distinguish between these different solutions.

Optimisation

An important heuristic used for choosing between hypothesised explanations of training data is to select the most limited language. In this work, this relates to the number of templates that must be added into the domain in order to explain the plan costs. However, it also relates to the complexity of the required templates. For example, in general, it is easier to define a problem's cost model using arity 1 templates rather than arity 4 templates. We therefore add a bias, which promotes simpler explanations through the use of fewer and more general templates.
We introduce a variable and constraint to represent the complexity of the active templates:

$$\text{Complexity} : Integer \tag{V4}$$

$$\text{Complexity} = \sum_{O \in \mathcal{O}} \sum_{\tau \in T_O} \text{Active}^O_\tau \times (arity(\tau) + 1) \tag{C4}$$

The complexity of each active template is its arity (plus 1 to account for operator cost), which penalises high arity templates. The optimisation problem solved by *NLOCM* is to minimise the Complexity variable.

Search Strategy

As mentioned previously, there are different templates available for what contributes to the action cost of each operator in the domain. The *NLOCM* search strategy is to successively solve more and more complex template sets until a satisfying assignment is found. The assignment that minimises the complexity cost is selected, in other words the simplest *language* that describes the action cost. The ordering of the templates is as follows:

1. Operator costs: Operator costs are those fixed costs associated to an operator in all problem instances. The most obvious example is the unit operator cost in pure STRIPS domains.
2. Action costs: Action costs are instance-specific costs assigned to operators. These are less common in the benchmark domains.
3. State Parameters: As discussed in the background section, *LOCM* and *LOCM2* identify state parameters by finding object associations within operator argument lists. Certain operators change the object identified in the state parameter. For example, the location state parameter in the truck machine of the Transport domain is changed by arguments 2 and 3 of the drive action. Because these relationships are known to be significant, we add their corresponding templates as potential contributors to the action costs.
4. Static Parameters: Similarly to state parameters, we now consider adding templates based on the static relations identified by the *LOP* system. Recall that *LOP* identifies a static relation for each operator. The scope of these static relations, thus forms a new template for each operator.
5. Single Transition Costs: This step allows costs to be associated with individual operator arguments (imagine, for example, if in the transport domain a fee had to be paid to enter a city). This step consists of a number of substeps intended to reduce the search space and increase the likelihood of finding solutions. In each step, we allow an incrementally increasing number of single parameters to contribute to the action cost. STC_O is used to denote, for each operator, the set of templates added at this layer. The following constraint is relaxed incrementally, by increasing the size, p, from 1 to the max action arity:

$$|\{\tau : \tau \in \mathrm{STC}_O \ \wedge \ \mathrm{Active}_\tau = 1\}| \leq p \qquad \text{(C5)}$$

If these levels are exhausted and no valid cost model is found then we return failure. Note that although *NLOCM* relies on *LOCM2* and *LOP* output, if neither of these systems return a solution then *NLOCM* can still proceed without steps 3 and 4 defined above.

Empirical Analysis

We now provide a discussion of our empirical evaluation on a large collection of benchmark planning domains. The results of our evaluation are shown in Table 2. All of our experiments are run on Mac OSX version 10.11.1 using an Intel 2 GHz i7 CPU with 8 GB system memory. *NLOCM* is implemented in Java (version 1.8.65) using the Choco constraint library (Prud'homme, Fages, and Lorca 2014) version 3.3.1. We use the domain over weighted degree variable selection heuristic, with a geometric restart policy. We use a ten minute time cutoff for any one constraint search.

Training Data Generation

We generate the training data for our evaluation using small sets of benchmark problem instances (typically 10) for each domain. Each problem is used as a starting point to generate a collection of related plans through a series of random walks. Each random walk is generated by first making a random number of steps from the initial state to select a starting state and then taking a fixed number of random steps from that state. These actions are recorded and their costs are summed to calculate the plan's cost.

In cases where certain operators are not used in any of the generated plans, we first attempt increasing the length of the walks and if there are still missing operators then we change the selected problems. This latter step is required in domains where increased complexity in problem models requires (or allows) the use of additional operators, as is the case in the Pipesworld domains. In some cases we have found that the generated constraint model is too large to solve and we have generated shorter plans. We comment on these below. We did not test our system on domains Cybersec, Parc Printer and PSR Small, as the benchmark instances are pre-compiled grounded instances and the original domains are unavailable.

Discussion of Results

We discuss the results from Table 2 in more detail now. We break the domains into three categories: STRIPS domains, easy action-cost domains and hard action-cost domains (where easy and hard relate to the level of template complexity).

STRIPS Domains: Benchmark domains that do not explicitly define action costs can be seen as simply having unit action costs. These domains vary in the number of operators from one in visitall to thirty in tidybot. In most cases we have used plan sets of 100 random walks of length 10 from the first 10 benchmark problems. The initial random walks did not include all of the operators in five domains. Generating longer walks was sufficient for the Thoughtful domain and a different set of benchmark problems (6-15) solved the problem for the three Pipesworld domains. In the Tidybot domain the number of problem instances and the length of the walks were extended before all 30 operators were observed in the random walks. In each case, *NLOCM* correctly stopped at the first language level, indicating that the plan costs could be explained by attributing costs to operators only. In all domains except AOP Freecell, *NLOCM* identified that each operator in the domain had an associated operator cost and that each of these costs was 1. In AOP Freecell (a remodelled version of Freecell), a hand mediates card movements and therefore random walks alternate between pickups and putdowns. The generated model attributes cost 2 to each of the three putdown operators. The correct model was selected when we used random walks with an odd length.

Easy Action Cost Domains: Of the benchmark domains that define action costs, nine are solvable with only operator

Domain	Operators		Input Plans			*LOCM* Performance			*NLOCM* Performance			
	#Ops.	Comp.	#Prob.	Len.	#Plan	*LOCM*	*LOCM2*	*LOP*	Time	Comp.	Valid	Error
Airport	4	4	10	10	100	✗	✗	✗	1.0	4	✓	0
AoP Freecell	8	8	10	10	100	✓	✓	✓	0.7	3	✲	5
Barman	12	12	10	10	100	✗	✗	✗	2.2	12	✓	0
Blocks-reduced	4	4	10	10	100	✗	✓	✓	1.8	4	✓	0
Childsnack	6	6	10	10	100	✗	✓	✓	1.4	6	✓	0
Cybersec	–	–	–	–	–	✗	✗	✗	–	–	–	–
Depot	5	5	10	10	100	✗	✓	✓	1.7	5	✓	0
Driverlog	6	6	10	10	100	✗	✓	✓	1.8	6	✓	0
Elevators	6	12	10	4	250	✗	✓	✓	80.7	12	✲	0
Floortile	7	7	10	10	100	✗	✗	✗	1.9	7	✓	0
Freecell	10	10	10	10	100	✗	✗	✗	1.9	10	✓	0
Ged	21	6	10	10	100	✓	✓	✓	1.7	6	✓	0
Grid	5	5	5	10	100	✓	✓	✓	1.0	5	✓	0
Gripper	3	3	10	10	100	✓	✓	✓	1.7	3	✓	0
Hiking	7	7	10	10	100	✗	✓	✓	1.8	7	✓	0
Logistics00-reduced	6	6	10	10	100	✓	✓	✓	1.7	6	✓	0
Mprime	4	4	10	10	100	✗	✗	✗	2.0	4	✓	0
Mystery	3	3	10	10	100	✗	✓*	✓	1.3	3	✓	0
No Mystery	3	3	10	10	100	✗	✓*	✓	1.5	3	✓	0
Openstacks	4	1	10	10	100	✗	✗	✗	1.1	1	✓	0
Parc Printer	–	–	–	–	–	✗	✗	✗	–	–	–	0
Parking	4	4	10	10	100	✓	✓	✓	2.4	4	✓	0
Peg Solitaire	3	1	10	10	100	✓	✓	✓	1.1	1	✓	0
Pipesworld	6	6	10	10	100	✗	✗	✗	2.2	6	✓	0
PW no Tankage	6	6	10	10	100	✗	✗	✗	2.0	6	✓	0
PW Tankage	6	6	10	10	100	✗	✗	✗	2.2	6	✓	0
PSR Small	–	–	–	–	–	✗	✗	✗	–	–	–	–
Rovers	9	9	10	10	100	✗	✗	✗	2.1	9	✓	0
Satellite	5	5	10	10	100	✗	✗	✗	1.8	5	✓	0
Scanalyzer	4	4	10	10	100	✗	✓	✓	1.8	4	✓	0
Sokoban	3	2	10	10	100	✓	✓	✓	0.8	2	✓	0
Storage	5	5	10	10	100	✓	✓	✓	1.8	5	✓	0
Tetris	6	6	10	10	100	✗	✗	✗	1.9	6	✓	0
Thoughtful	21	21	5	50	100	✗	✗	✗	1.7	21	✓	0
Tidybot	30	30	15	100	100	✓	✓	✓	9.7	30	✓	0
TPP	4	4	10	10	100	✗	✗	✗	1.6	4	✓	0
Transport	3	5	10	4	250	✗	✓*	✓	52.3	5	✲	0
Visitall	1	1	10	10	100	✗	✓*	✓	1.3	1	✓	0
Woodworking	13	17	10	1	1000	✗	✗	✗	605.2	20	✲	3
Zenotravel	5	5	10	10	100	✗	✓*	✓	1.6	5	✓	0

Table 2: Table of results running *NLOCM* on a collection of benchmark domains. Headings refer to #Ops. (number of operators) Comp. (complexity of benchmark model) #Prob. (number of plan sets/problems) Len. (length of random walks) #Plan (plans per plan set) *LOCM*, *LOCM2* and *LOP* (whether the output from the *LOCM* family was valid) Time (time to select model in seconds) Comp. (complexity of model selected by *NLOCM*) Valid (whether the model matched the benchmark, stars indicate that the correct model was identified after the original training data was changed) Error (number of errors in selected model).

costs: these are Sokoban, Floortile, Parking, Ged, No Mystery, Tetris, Peg Solitaire, Barman and Openstacks. *NLOCM* correctly identifies that only operator costs are required and it identifies the correct operator cost for each operator for these domains. Of these domains, Parking and No Mystery define unit cost for each operator and therefore are equivalent to the cases above. In the three domains Tetris, Floortile and Barman, the operators each have an associated cost but are not all unit cost. For example, it costs more to move larger pieces in the Tetris domain. *NLOCM* finds the correct values for each operator in these domains.

The remaining domains (Sokoban, Ged, Peg Solitaire and Openstacks) are more interesting because they include zero cost operators. For example, in Peg Solitaire beginning a move costs a single unit, but continuing a move or ending a move has no cost. In each of these domain *NLOCM* correctly identifies the cost of each operator. In Ged, *NLOCM* correctly identifies the six operators that have non-zero cost and correctly identifies each of these costs. For example, attributing cost of two to `begin-transverse-splice` and one to `begin-inverse-splice`.

Hard Action Cost Domains: In the remaining three do-

mains it is not possible to explain the data through operator costs alone. The Elevators domain has six *LOCM2* state parameters. The floor is transferred in the floor move actions, and the passenger transfers between a floor and a lift using the board and leave actions. The state parameter layer therefore generates six templates that are added to the operator cost templates. Similarly, in the Transport domain, *LOCM2* uncovers the four state parameters. In our first test (with 100 walks of length 10) the constraints solver does not find a solution. In our tests we have found that using smaller problem instances can lead to the solver finding a solution. However, in the table we report an alternative set of random walks with more walks of shorter length for the original 10 problems. In both domains, *NLOCM* correctly identifies the underlying contribution of the subset of state parameter templates and identifies the associated cost for each of the pairs for the input problems.

Because the *LOCM2* analysis failed, neither state parameters nor statics are known for the Woodworking domain templates. The domain is encoded using four operator costs that depend on a single parameter and standard operator costs for the other operators. This solution should therefore be reachable for *NLOCM* in the first level of step 5 (single parameter costs). However, the solver does not return a solution for any level of step 5 for our standard amount of training data. We have tested the domain using several training data sets: when the solver has returned a solution single parameters have been selected for several operators. However, some of these are not consistent with the benchmark model. To explore this further we used a collection of 1000, single step input plans, for 10 problems, which provides the solver with costings of individual ground actions. With this rich data, *NLOCM* identified the four operators and parameters that are encoded in the domain. However, it also used other single parameters, although on closer inspection all costs across these were equal and could be translated into a correct cost.

Related and Future Work

There is a long heritage of domain model acquisition systems in the field of automated planning, with more recent endeavours perhaps starting with the TRAIL system (Benson 1996). *NLOCM* follows in the aims of the *LOCM*-derived systems (work that started as targeting the OCL (McCluskey and Porteous 1997) language, including the Opmaker systems (McCluskey et al. 2009; Richardson 2008) in addition to the systems described in this work) and uses a minimal amount of input. Most other systems use other information, such as predicates, initial and goal states and possibly intermediate states. Another system, like *LOP*, that derives static information is the recent ASCOL (Jilani et al. 2015). There are relatively few works that target the numeric fragment of PDDL. In (Lanchas et al. 2007) complete knowledge of all intermediate states, including the cost of each action, is used to learn operator costs. However, the approach does not discover the relevant parameterisation underlying the cost function (so therefore cannot learn problem-specific costs). Some systems can target richer propositional fragments than *NLOCM*, but require more input information (examples include ARMS (Wu, Yang, and Jiang 2007) and LAMP (Zhuo et al. 2010), which can target part of the ADL fragment of PDDL. Systems that learn planning models in the presence of noisy and incomplete data (Mourao et al. 2012) have also been studied. In addition to the planning community, there is wide and active interest in automatic model acquisition in many of the sub-fields of combinatorial search and beyond, for example in constraint satisfaction (O'Sullivan 2010; Bessiere et al. 2014), general game playing (Björnsson 2012; Gregory, Björnsson, and Schiffel 2015), and software engineering (Reger, Barringer, and Rydeheard 2015).

Future work will concentrate on developing algorithms to learn more complex numeric properties of planning domains. One possible approach could be to take inspiration from model learning in the constraint programming community. Preconditions can be viewed as constraints over the current state that have to be satisfied in order to apply the action. Effects can be seen as constraints over the current state and the successor state. We intend to adapt techniques used in an interactive setting in constraint acquisition (Bessiere et al. 2014) to use in a domain model acquisition system.

Conclusions

Numeric planning is important in many commercial and industrial applications of planning. Domain model acquisition technology can assist engineers in creating domain models. A goal of this line of research is to allow the synthesis of more complex domain models from sparse information.

In this work, we have introduced the *NLOCM* domain model acquisition system that can learn domain models with action costs using a constraint programming approach. Our results demonstrate that *NLOCM* is effective, providing good coverage over a wide range of benchmark domains. This class of planning domain is an important subset of numeric planning domains. *NLOCM* is the first domain model acquisition system to target this fragment of PDDL, and does so using minimal input, requiring only the final cost of each input plan, and the observed actions.

Acknowledgements

This work is supported by EPSRC Grant EP/N017447/1.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11(4):625–655.

Baranes, A., and Oudeyer, P. Y. 2013. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems* 61(1):49–73.

Benson, S. S. 1996. *Learning Action Models for Reactive Autonomous Agents*. Ph.D. Dissertation.

Bessiere, C.; Coletta, R.; Daoudi, A.; Lazaar, N.; Mechqrane, Y.; and Bouyakhf, E. H. 2014. Boosting Constraint Acquisition via Generalization Queries. In *European Conference on Artificial Intelligence*.

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

Cakmak, M.; Chao, C.; and Thomaz, A. 2010. Designing Interactions for Robot Active Learners. *IEEE Transactions on Autonomous Mental Development* 2(2):108–118.

Cresswell, S., and Gregory, P. 2011. Generalised Domain Model Acquisition from Action Traces. In *International Conference on Automated Planning and Scheduling*, 42 – 49.

Cresswell, S. N.; Mccluskey, T. L.; and West, M. M. 2009. Acquisition of Object-Centred Domain Models from Planning Examples. In *ICAPS*, 338 – 341.

Ersen, M., and Sariel, S. 2015. Learning behaviors of and interactions among objects through spatio-temporal reasoning. *Computational Intelligence and AI in Games, IEEE Transactions on* 7(1):75–87.

Fox, M.; Long, D.; and Magazzeni, D. 2011. Automatic construction of efficient multiple battery usage policies. In *International Conference on Automated Planning and Scheduling*, 74–81.

Frank, J. D.; Clement, B. J.; Chachere, J. M.; Smith, T. B.; and Swanson, K. J. 2011. The Challenge of Configuring Model-Based Space Mission Planners. In *International Workshop on Planning and Scheduling for Space*.

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

Gregory, P.; Björnsson, Y.; and Schiffel, S. 2015. The GRL System : Learning Board Game Rules With Piece-Move Interactions. In *GIGA*.

Hoffmann, J.; Weber, I.; and Kraft, F. M. 2012. SAP speaks PDDL: Exploiting a software-engineering model for planning in business process management. *Journal of Artificial Intelligence Research* 44:587–632.

Jilani, R.; Crampton, A.; Kitchin, D. E.; and Vallati, M. 2015. 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*, 438–451.

Lanchas, J.; Jiménez, S.; Fernández, F.; and Borrajo, D. 2007. Learning action durations from executions. In *Proceedings of the ICAPS'07 Workshop on AI Planning and Learning*.

McCluskey, T. L., and Porteous, J. 1997. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence* 95(1):1–65.

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

Mourao, K.; Zettlemoyer, L.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS Operators from Noisy and Incomplete Observations. In *Uncertainty in Artifical Intelligence*, 614 – 623.

O'Sullivan, B. 2010. Automated modelling and solving in constraint programming. In *AAAI*.

Parkinson, S.; Longstaff, A.; Crampton, A.; and Gregory, P. 2012. The Application of Automated Planning to Machine Tool Calibration. In *International Conference on Automated Planning and Scheduling*.

Porteous, J.; Teutenberg, J.; Pizzi, D.; and Cavazza, M. 2011. Visual programming of plan dynamics using constraints and landmarks. In *International Conference on Automated Planning and Scheduling*, 186–193.

Porteous, J.; Charles, F.; and Cavazza, M. 2013. NetworkING: using character relationships for interactive narrative generation. In *International Conference on Autonomous Agents and Multi-agent Systems*, 595–602.

Porteous, J.; Charles, F.; and Cavazza, M. 2015. Using Social Relationships to Control Narrative Generation. In *AAAI*, 4311–4312.

Prud'homme, C.; Fages, J.-G.; and Lorca, X. 2014. *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S.

Reger, G.; Barringer, H.; and Rydeheard, D. 2015. Automata-based Pattern Mining from Imperfect Traces. *ACM SIGSOFT Software Engineering Notes* 40(1):1–8.

Richardson, N. E. 2008. *An Operator Induction Tool Supporting Knowledge Engineering in Planning*. Ph.D. Dissertation, School of Computing and Engineering, University of Huddersfield, UK.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *Knowledge Eng. Review* 22(2):117–134.

Surovik, D. A., and Scheeres, D. J. 2015. Heuristic Search and Receding-Horizon Planning in Complex Spacecraft Orbit Domains. In *International Conference on Automated Planning and Scheduling*, 291–295.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itsimple 2.0: An integrated tool for designing planning domains. In *International Conference on Automated Planning and Scheduling*, 336–343.

Walsh, T. J., and Littman, M. L. 2008. Efficient Learning of Action Schemas and Web-Service Descriptions. In *AAAI*, 714 – 719.

Wickler, G.; Chrpa, L.; and McCluskey, T. L. 2014. KEWI - A knowledge engineering tool for modelling AI planning tasks. In *International Conference on Knowledge Engineering and Ontology Development*, 36–47.

Wu, K.; Yang, Q.; and Jiang, Y. 2007. ARMS: An automatic knowledge engineering tool for learning action models for AI planning. *The Knowledge Engineering Review* 22(2):135–152.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intellgence* 174:1540–1569.