Learning action effects in partially observable domains

Kira Mourão
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
k.m.t.mourao@sms.ed.ac.uk

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Mark Steedman
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
steedman@inf.ed.ac.uk

Abstract

We investigate the problem of learning action effects in partially observable STRIPS planning domains. Our approach is based on a voted kernel perceptron learning model, where action and state information is encoded in a compact vector representation as input to the learning mechanism, and resulting state changes are produced as output. Our approach relies on deictic features that embody a notion of attention and reduce the size of the representation. We evaluate our approach on a number of partially observable planning domains, adapted from domains used in the International Planning Competition, and show that it can quickly learn the dynamics of such domains, with low average error rates. Furthermore, we show that our approach handles noisy domains, and scales independently of the number of objects in a domain, making it suitable for large planning scenarios.

Introduction and motivation

An agent operating in a real-world domain often needs to do so with incomplete information about its environment. In particular, an agent must often act or make decisions with only partial or noisy information about the state of the world. Automated planning systems are effective at controlling the behaviour of agents in a variety of domains. However, such tools require a model of the domain in which the agent will operate. In real-world domains, such models may not be readily available, nor be sufficiently detailed to account for the subtleties inherent in complex environments.

Acquiring a domain model automatically through learning and experience gives an agent greater flexibility to handle unexpected situations, and avoids the need for a pre-existing model of the world. Learning the dynamics of a domain can be a challenging problem, however, especially in domains where an agent only has partial access to the world state, or external sensors that are susceptible to noise. Furthermore, since a learnt action model may be subsequently used for planning, the resulting learning method should be as accurate, fast, and scalable as possible.

Using machine learning techniques to induce action models is not a new idea, with the literature divided between two main approaches: high-level, logical approaches and low-level, sensor-driven approaches. High-level approaches work within the space of transition rules (Wang 1995; Gil 1994; Amir and Chang 2008; Pasula, Zettlemoyer, and Kaelbling 2007; Benson 1996) to find either a "good" set or all consistent sets of rules. These methods attempt to exploit relational structure in order to improve speed and generalisation performance. Such approaches have also been applied to partially observable (Amir and Chang 2008) or non-deterministic (Pasula, Zettlemoyer, and Kaelbling 2007) domains. Alternatively, low-level methods operate closer to the sensor level. Such approaches construct transition rules from actions and robot sensor data coded as sets of objects or raw sensor readings, and predicates derived from this data (Metta and Fitzpatrick 2003; Holmes and Isbell 2005; Doğar et al. 2007; Modayil and Kuipers 2008). Although many of these methods have had limited success at learning aspects of particular domains, few of them fully address the problem of learning partially observable domains, and fewer still are capable of handling noise.

In this paper we consider the problem of learning the effects of an agent's actions, that is, the transition rules between world states. We focus on learning the effects of STRIPS actions (Fikes and Nilsson 1971) in deterministic and partially observable domains. In particular, we consider actions which affect a subset of the propositional features that make up the world state. Following (Pasula, Zettlemoyer, and Kaelbling 2007; Benson 1996), we use deictic features that embody a notion of attention to produce a compact representation of the domain.

This paper builds on our previous work (Mourão, Petrick, and Steedman 2008) which also used deictic coding to generate a compact vector representation of the world state, and learnt action effects as a classification problem. However, the method only applied to fully observable domains, as the kernel perceptron classifier used there performs badly with noisy or partially observable data. Additionally, the approach was only tested on a single synthetic domain with simulated states which were not necessarily reachable by a sequence of actions in the domain.

Here we extend this work to partially observable (and noisy) domains using kernelised voted perceptrons (Aizerman, Braverman, and Rozoner 1964; Freund and Schapire 1999) to learn action transitions in the domains. Such methods are particularly useful since they provide good performance, in terms of both the training time and the quality of the learnt models. Furthermore, we test our approach against a set of standard planning domains taken from the

3rd International Planning Competition,[1] demonstrating that our method is fast, accurate, and scales independently of the number of objects in the world, thereby making it suitable for large planning scenarios.

Domain learning

Definitions

The action representations we will use are based on the logical representations typically found in planning systems. A *domain* $\mathcal{D}$ is defined as a tuple $\mathcal{D} = \langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$, where $\mathcal{O}$ is a finite set of world objects, $\mathcal{P}$ is a finite set of predicate (relation) symbols, and $\mathcal{A}$ is a finite set of actions. Each predicate and action also has an associated arity. Predicates of arity 0 are referred to as *object independent* properties, while those of arity at least 1 are *object dependent* properties.

A *fluent* is an expression $p(c_1, c_2, \ldots, c_n)$, where $p \in \mathcal{P}$, n is the arity of p, and each $c_i \in \mathcal{O}$. A *state* is any set of fluents, and $\mathcal{S}$ is the set of all possible states. For any state $s \in \mathcal{S}$, a fluent p is true at s iff $p \in s$. The negation of a fluent, $\neg p$, is true at s (also, p is false at s) iff $p \notin s$.

Each action $a \in \mathcal{A}$ is defined by a set of *preconditions*, Pre_a, and a set of *effects*, Eff_a. Pre_a can be any set of fluents and negated fluents. In STRIPS actions each effect $e \in Eff_a$ has the form $add(p)$ or $del(p)$, where p is any fluent. Action preconditions and effects can also be parameterised. An action with all of its parameters replaced with objects from $\mathcal{O}$ is said to be an *action instance*.

Action instances are state transforming. Given a state s and an action instance A, A is *applicable* (or *executable*) at s iff each precondition $p \in Pre_A$ is true at s. An applicable action produces a new state s' that is identical to s, but updated with the effects of A as follows: for each $e \in Eff_A$, (i) if e is an effect $add(p)$ then p is added to s', and (ii) if e is an effect $del(p)$ then p is removed from s'.

Learning model

The task of the learning mechanism is to learn the associations between action-precondition pairs and their effects, that is, rules of the form $\langle A, Pre_A \rangle \rightarrow Eff_A$. As a result of the form of the planning actions we allow, effects of rules are assumed to be deterministic and disjunctive effects (i.e., effects of the form "either p_1 or p_2 changes") are not allowed. Instead, all effects are simply conjunctions of predicates, meaning it is sufficient to learn the rule for each effect predicate separately. Thus, we can treat the learning problem as a set of binary classification problems, with one problem for each effect predicate.

To address our particular learning problem we use the *perceptron* (Rosenblatt 1958), a simple yet fast binary classifier. The perceptron maintains a weight vector $\mathbf{w}$ which is adjusted at each training step. The i-th input vector $\mathbf{x}_i \in \{0, 1\}^n$ in a class $y \in \{-1, 1\}$ is classified by the perceptron using the decision function $f(\mathbf{x}_i) = sign(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle)$. If $f(\mathbf{x}_i)$ is not the correct class then $\mathbf{w}$ is set to $\mathbf{w} + y\mathbf{x}_i$; if $f(\mathbf{x}_i)$ is correct then $\mathbf{w}$ is left unchanged. Provided the data is linearly separable, the perceptron algorithm is guaranteed to converge on a solution in a finite number of steps (Novikoff 1963; Minsky and Papert 1969). If the data is not linearly separable then the algorithm oscillates, changing $\mathbf{w}$ at each misclassified input vector.

One solution for non-linearly separable data is to map the input feature space into a higher-dimensional space where the data is linearly separable. However, an explicit mapping may lead to a massive expansion in the number of features, making the classification problem computationally infeasible. Instead, an implicit mapping is achieved by applying the *kernel trick* to the perceptron algorithm (Freund and Schapire 1999), by noting that the decision function can be written in terms of the dot product of the input vectors:

$$f(\mathbf{x}_i) = sign(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle) = sign(\sum_{j=1}^{n} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle),$$

where α_j is the number of times the j-th example has been misclassified by the perceptron. By replacing the dot product with a *kernel function* $k(\mathbf{x}_i, \mathbf{x}_j)$ which calculates $\langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \rangle$ for some mapping ϕ, the perceptron algorithm can be applied in higher dimensional spaces without ever requiring the mapping to be explicitly calculated.

We represent each state s as a vector (see below) and learn state transitions using a bank of kernel perceptrons, one for each output bit, corresponding to a single predicate p. Since in general the problem of learning action effects is not linearly separable, the kernel perceptron is an appropriate choice for this problem. Kernel perceptrons obtain reasonable accuracy at acceptable training and prediction speeds, allowing us to use this approach in practical planning applications. Alternative non-linear classifiers, such as SVMs (Boser, Guyon, and Vapnik 1992), can be substantially slower (Surdeanu and Ciaramita 2007) while performance is not guaranteed to be better (Graepel, Herbrich, and Williamson 2000). To improve the speed of the classifier we use a variant of the kernel perceptron, the *voted perceptron* (Freund and Schapire 1999), which is computationally efficient and produces performance close to the best performing maximal-margin classifiers on similar problems. Preliminary tests using SVMs on our problem give similar results with longer computation times.

The kernel is chosen to allow the perceptron algorithm to run over conjunctions of features in the original input space, as this permits a more accurate representation of the exact conjunction of features (action and preconditions) corresponding to a particular effect. We use the polynomial kernel of degree 3, $k(x, y) = (x.y + 1)^3$ so that feature conjunctions of up to three features make up the feature space.

Representation

We compute a reduced form of the input state space for each action using deictic coding (Agre and Chapman 1987). A deictic representation maintains pointers to objects of interest in the world, with objects coded relative to the agent or current action. Objects which cannot be indexed in this way are excluded from the reduced state for the current action.

[1] See `http://ipc.icaps-conference.org/`.

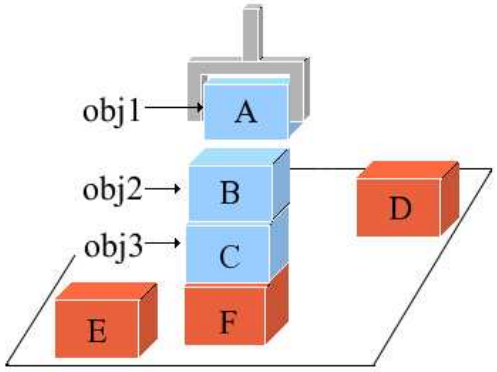


Figure 1: Action *stack*(A, B) results in objects A, B, and C being attended to, while unrelated objects D, E and F are ignored. Objects A, B, and C are referred to by the variables $obj1$, $obj2$ and $obj3$, respectively, in the vector representation shown in Figure 2.

For instance, such a deictic representation might arise from an attentional mechanism (Ballard et al. 1997).

For a given action instance A we construct the set of objects of interest $\mathcal{O}^{\mathcal{A}}$, by combining a primary set of objects, given by the parameters of the action, and a secondary set of objects which are directly related to the primary set, in the current state. We define a direct relation between objects c_i and c_j to exist in state $s \in \mathcal{S}$ if $\exists p \in \mathcal{P}$ such that $p(c_1, c_2, \ldots, c_n) \in s$ and $c_i, c_j \in \{c_1, c_2, \ldots, c_n\}$

The related objects are identified via deictic references, e.g., if a particular action grasps an object x, then the primary set of objects consists of the "grasped-object" x, and the secondary set might consist of the "object-under-the-grasped-object" y; other objects which are not directly related to x are not represented. We define $M = \max_A |\mathcal{O}^{\mathcal{A}}|$ to be the maximum possible number of objects of interest for any action instance.

Figure 1 presents an example from the BlocksWorld domain, in which an agent can manipulate a set of blocks on a table. Given the action *stack*(A, B), i.e., stack block A on top of block B, the primary set of objects is $\{A, B\}$. The only object related to A or B is C, since B is on C. Therefore the full set of objects of interest is $\{A, B, C\}$.

An input vector representing the state space is constructed as follows. Each action $a \in \mathcal{A}$, and each 0-ary predicate, is represented by a bit. Then for each object $o \in \mathcal{O}^{\mathcal{A}}$, all the possible relations between o and all other objects in $\mathcal{O}^{\mathcal{A}}$ must be represented. This requires at most $\binom{M-1}{n}$ bits for each n-ary predicate, for each object in $\mathcal{O}^{\mathcal{A}}$. The value of a bit is 1 (-1) if the corresponding predicate is true (false), or if the corresponding action is (not) the current action. If a bit corresponds to an unobserved predicate, the value is set to 0. When $|\mathcal{O}^{\mathcal{A}}| < M$ for some action instance A, bits for unused predicates are set to 0. Figure 2 shows the vector representation of the state in Figure 1.

The form of the output vectors representing an action's effects on a state are identical to the input vectors, except

Input vector	Corresponding action/predicate	
-1	*pickup*(*obj1*)	Actions
-1	*putdown*(*obj1*)	
1	*stack*(*obj1*, *obj2*)	
-1	*unstack*(*obj1*, *obj2*)	
-1	*armempty*	Object independent properties
	...	
1	*holding*	Properties of *obj1*
-1	*ontable*	
-1	*clear*	
-1	*on-obj1*	
-1	*on-obj2*	
-1	*on-obj3*	
	...	
-1	*holding*	Properties of *obj2*
-1	*ontable*	
1	*clear*	
-1	*on-obj1*	
-1	*on-obj2*	
1	*on-obj3*	
	...	
-1	*holding*	Properties of *obj3*, included as *obj3* is related to *obj2*
1	*ontable*	
-1	*clear*	
-1	*on-obj1*	
-1	*on-obj2*	
-1	*on-obj3*	
	...	

Figure 2: Input vector representation of the (fully observable) BlocksWorld *stack* action and prior state from Figure 1. The first 4 bits correspond to the 4 domain actions. The bit for *stack* is set to 1 since it is the current action. The 0-ary predicate *armempty* is represented by a single bit, set to -1 since the gripper is holding object A. The first set of object predicates represented in the vector are those for object A since it is the first parameter of *stack*. The second set of object predicates relate to object B, as the second parameter of *stack*, and finally the third and last set of object predicates relate to object C, as it is related to object B by the *on* predicate. If object B were being stacked on object A the predicates for object B would precede those for object A instead.

that the actions themselves are excluded from the vector, and bits are set to 1 if the corresponding predicate changes, -1 if the corresponding predicate does not change, and 0 if the corresponding predicate was not observed either before or after the current action. During learning, only examples with a known change, i.e. values 1 or -1, are used to train the kernel perceptrons. The ordering of object representations in the vectors is constrained so that two objects with the same role in the same action, but in two different instances of the action, must always be represented at the same position in the vectors.

Deictic coding has a number of benefits. It greatly reduces the size of the input for an algorithm learning to predict action effects, as information is discarded about objects unrelated to the action or its parameters. As a consequence, scalability is improved, because the size of the representation does not increase with the size of the universe.

The size of the representation required for relations between objects is also reduced. Firstly, any relations including discarded objects can be ignored. More importantly, deictic coding means that objects are represented by variables rather than by constants, and so whether we grasp object A sitting on object B, or grasp object C sitting on object D, the "on" relation is always represented as "the-object-I-will-grasp is on the-object-under-the-object-I-will-grasp". Thus, if the representation considers M objects of the possible $|\mathcal{O}|$ in the state space, the number of instances of each binary relation which needs to be represented drops from O($|\mathcal{O}|^2$) to O(M^2). $M < |\mathcal{O}|$ but is otherwise unrelated to $|\mathcal{O}|$, and instead typically depends on the complexity of the domain ($M < 8$ for the domains considered here). Finally, deictic coding creates a strong bias for generalisation.

Experiments

We tested the learning model on standard planning domains from the 3rd International Planning Competition (IPC): Depots and ZenoTravel, as well as a standard BlocksWorld domain. The domains are described in PDDL (McDermott et al. 1998), the standard representation language of the IPC. BlocksWorld is a very simple domain with 4 actions (maximum 2 parameters) and 5 predicates (maximum arity 2). All objects are blocks. The maximum possible number of objects of interest, M, is 3. Depots and ZenoTravel are more complex. Depots has 5 actions (maximum 4 parameters), 6 predicates (maximum arity 2) and 6 types of objects (represented as predicates of arity 1). ZenoTravel has 5 actions (maximum 6 parameters), 8 predicates (maximum arity 2) and 4 types of objects (again represented as predicates of arity 1). $M = 5$ and $M = 7$ for Depots and ZenoTravel respectively.

Sequences of random actions and resulting states were generated from PDDL domain descriptions and used as training and testing data.[2] The number of objects in the state space was higher in the test data than in the training data, to demonstrate that the learnt models could be applied across different instances of the same domain. Specific problems from the IPC were used to set the sizes of the initial states for each sequence.[3] BlocksWorld was initialised with 13 blocks for training and 30 blocks for testing. The actual initial states were generated at random using the IPC3 problem generator and a BlocksWorld state generator (Slaney and Thiébaux 2001).

Partial observability was simulated by randomly selecting a number of predicates from the world to observe after each action. The remaining predicates were discarded and the reduced state vector was generated from the observed fluents. The number of observed predicates was set to approximately 5-20% of the total number of predicates (including negations) required to fully describe the state (BlocksWorld: 209, ZenoTravel: 2116, and Depots: 1764).

[2]All data was generated using the Random Action Generator 0.5 available at `http://magma.cs.uiuc.edu/filter/`.

[3]Depots problem 5, and ZenoTravel problem 9 for training; Depots problem 19 and ZenoTravel problem 14 for testing.

To determine an error bound on our results, 10 runs with different randomly generated training and testing sets were used. Our testing environment was a 2.4 GHz quad-core system with 4 Gb of RAM. All experiments were run on a single CPU. Each training set consisted of sequences of actions and state observations of lengths ranging from 1000-20000, and each test set was a sequence of length 2000.

Results

Using our representation, learning in fully observable domains is easy. Accordingly, the action models in all three domains were learnt in under 250 examples, which was sufficient to fully predict the 2000 test examples (Figure 3(a)).

Partial observability reduces the number of useful examples which can be learnt from, and also reduces the number of useful bits in each example. Substantially more examples are therefore needed to learn the action model. Furthermore, the variance of the errors on the test set is much higher than in the fully observable case. The higher variance is due to the small number of observed predicates during training. Although the test case is fully observable, only a fraction of the state is used for prediction. This can cause the learner to mistake one action for another (a form of perceptual aliasing) and wrongly predict every instance of the action, resulting in a high number of errors on the test set.

In the BlocksWorld domain, observing 30 randomly chosen predicates from the full state description (approximately 15% of the state) over 9000 examples is sufficient to fully predict all the test sets of 2000 examples (Figure 3(b)). Observing fewer predicates also produces good results: training on 9000 examples is sufficient to fully predict eight of the ten test sets for both 10 and 20 observations at each time step. The failures are all instances of the *stack* action. Most incorrectly predict all instances of *stack*, resulting in approximately 25% error on each test set. One case wrongly predicts the results of the *stack* action only when the block being stacked upon is already stacked on top of another block, producing an 8% error.

In the ZenoTravel domain, 300 observations at each step (again, approximately 15% of the state) allows for complete prediction of the test set in three of the ten test cases. Of the remaining test sets, five have only 1 or 2 prediction errors, while the last three wrongly predict every case of the *refuel* action (by not predicting the deletion of the fluent specifying the initial fuel level), resulting in approximately 27% errors on each test set. By increasing the number of observations at each step to 400, eight of the ten test sets can be fully predicted, with only one error and two errors respectively on the remaining test sets (Figure 3(c)). The Depots domain is more challenging for our method and 300 observations at each step (approximately 17% of the state) are only sufficient to completely predict the test set in half the test cases. The other test sets have around 15% error as every case of the *load* action is wrongly predicted. However, with 400 observations at each step, after 14000 steps, all of the test sets are fully predicted (Figure 3(d)).

The poorly predicted test cases in the three domains are all instances of the perceptual aliasing problem discussed above. The problem can be resolved by supplying an exam-

ple to the learner which it would predict incorrectly, so that a new prediction function can be learnt. With a randomly generated sequence of actions, many additional examples could be required before such an example was generated. However, in a planning scenario the necessary example could be identified from a plan execution failure and used to improve the prediction.

Scalability

Our empirical results demonstrate the effectiveness of our approach in a number of standard planning domains. Our approach can also be shown to scale, making it suitable for more complex problems in domains with large numbers of actions and objects. In particular, our approach takes time proportional to a measure of the complexity of the domain, and the number of mistakes made during learning.

For a reduced state vector of length l, there are $l - |\mathcal{A}|$ voted perceptrons each computing one bit of the output vector. In the fully observable case, each voted perceptron learns in time proportional to the number of training examples n, the length of the reduced state vector l, and k, the number of mistakes made so far ($k \leq n$). Thus, the complexity of the learning algorithm is O(l^2n^2). Predictions are made in time O(l^2n). The same analysis applies to the partially observable case, however many more observations are needed, firstly because many observations will not contain the necessary information about whether a predicate has changed as a result of an action, and secondly because there is less information in the partially observed world state from which to learn. The former is mitigated by the fact that the kernel perceptrons only train on "useful" training examples where the change in a predicate is known, and only these examples incur significant computational cost.

Recall that M is the maximum possible number of objects of interest for any action instance. Additionally, we define $P_i = \{p \in \mathcal{P} : arity(p) = i\}$ and $m = max_{p \in \mathcal{P}}(arity(p))$. Then the length of the reduced state vector is given by $l = |\mathcal{A}| + |P_0| + M \sum_{i=1}^{m} \binom{M-1}{i} |P_i|$. Thus, l depends on the number and arity of predicates in the domain, and the maximum possible number of objects of interest. Intuitively it makes sense that more complex domains with more predicates and more inter-relations between objects should require more time to learn and predict. In particular, M does not depend on the number of object instances in the domain. For the planning domains considered here, which only have predicates with arity below 3, M is typically not very large ($M < 8$).

When the domain is fully observable, the number of observations n required by our method to learn the action model does not depend on the number of objects in the world. However, this is not the case for partially observable domains. An observation is only useful to the learning process if it contains fluents relating to the set of objects of interest, and at least one which was observed immediately before the current action, so that it is known whether a change occurred. Under partial observability, the probability of observing 'useful' fluents decreases as the number of objects in the world increases, as the number of 'useful' fluents remains constant while the total number of fluents increases. Thus the number of observations needed to learn the domain increases with the number of objects in the world. However we can learn in a smaller state space and apply the results to a larger one, since the representation is not dependent on the number of objects in the world.

The complexity can be reduced by limiting the number of vectors the voted perceptron stores, which has the effect of fixing k to be constant, and reduces the learning complexity by a factor of n to O(l^2n). However, the reduction of the state to a vector of length l by the deictic representation remains crucial. Accuracy may also be affected if k is too small. Furthermore, our solution is *embarrassingly parallel*, since the learning and prediction of each output bit is independent of the others. Running the calculations for each output bit in parallel would further reduce the complexity of learning and prediction, by a factor of l.

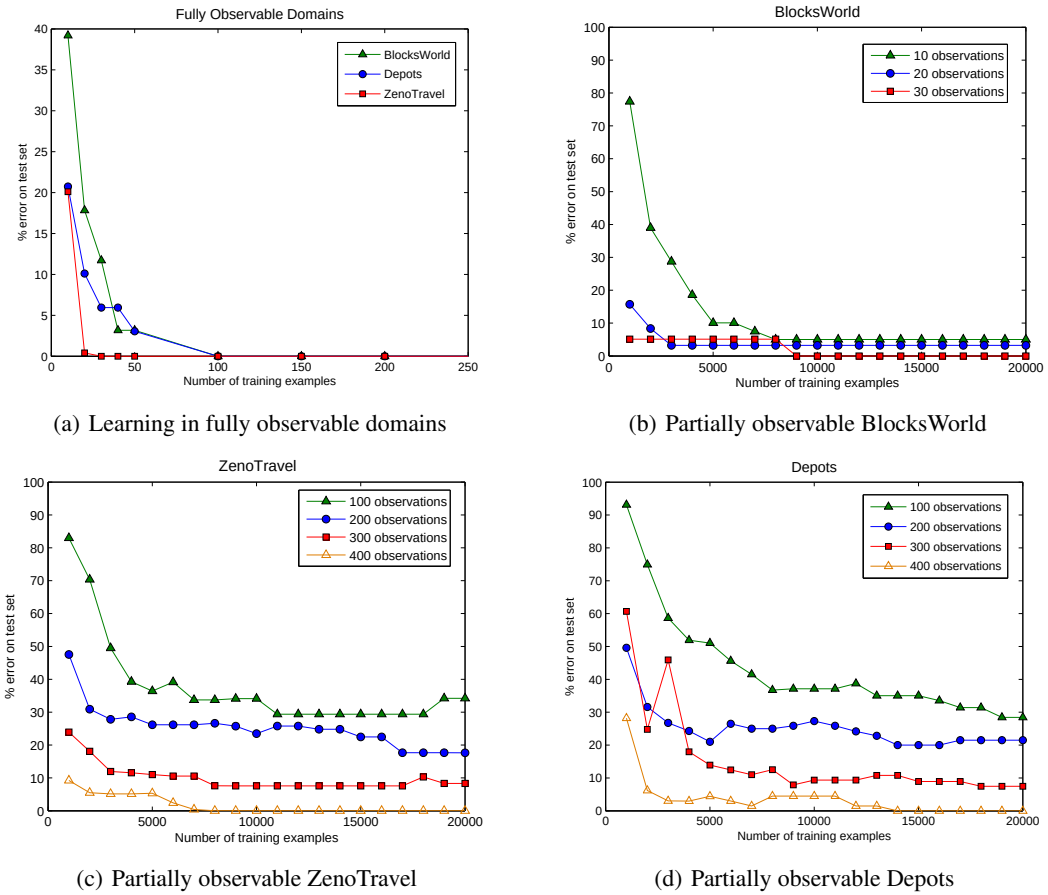
Currently our approach does not directly support typed domains. Instead, types are represented by adding new fluents to the domain description. As with the introduction of new objects, these additional fluents increase the number of observations needed to learn the domain. However, by supporting typed domains, we could also significantly improve the performance of our approach.

Discussion

Our method is dependent on the existence of a structured parametric representation, abstracted from the grounded sensory manifold itself. In particular, actions must be specified whose parameters are exactly those objects acted on, so that the correct set of objects is passed into the reduced state vector. We anticipate such information would be provided by an attentional model applied to a visual scene which picks out the necessary actions and their parameters. Such a model exists (Satish and Mukerjee 2008), and could be used to provide grounded parameterised actions.

Our method also depends on the use of a deictic representation, which both introduces a bias for generalisation and limits the number of objects considered by the learning and prediction process. Deictic representations have previously been applied to learning domain dynamics. (Benson 1996) converts the first-order logic description of the state space into a propositional description by representing objects with deictic variables. Our use of deictic variables is essentially the same. However (Benson 1996) uses the transitive closure of the relations among objects rather than the single step computation which we use. Similarly, (Pasula, Zettlemoyer, and Kaelbling 2007) use deictic references to objects to provide a generalisation bias and to reduce the search space of transition rules.

Our training and testing data was generated to mimic data collected by an agent exploring the world, and corresponds to sequences of actions and observations taken from random walks through the state space. In some domains (e.g. Grid, Freecell) certain actions occur infrequently, if at all, under these conditions, and so learning of such actions may fail. A more guided exploration of the state space may be necessary to learn in these domains. In particular, we do not use exam-



(a) Learning in fully observable domains

(b) Partially observable BlocksWorld

(c) Partially observable ZenoTravel

(d) Partially observable Depots

Figure 3: Results of learning action models in standard planning domains

ple plans as training data, since this presupposes an existing model of the domain which is to be learnt.

Tractability is a significant issue in learning action models in partially observable domains: methods such as HMMs, Dynamic Bayes Nets and Reinforcement Learning scale poorly (Amir and Chang 2008). More tractable methods use schema learning (Holmes and Isbell 2005), build a CNF representation of all possible transition models (Amir and Chang 2008), or convert the problem into a weighted MAXSAT problem (Yang, Wu, and Jiang 2007). In terms of tractability, our approach is competitive with these methods. A direct comparison is not straightforward as our method currently does not produce explicit rules which could be compared.

We learn action models in partially observable *noiseless* domains. Our approach also performs well in fully observable *noisy* domains (submitted). Figure 4 shows the results of our method applied to learning the ZenoTravel domain with 5% and 10% uniform noise, under full observability. We believe our approach will extend to learning in noisy partially observable domains. Future work will investigate this claim. Some earlier work learns action models in probabilistic, partially observable, noisy domains using schema learning (Holmes and Isbell 2005). However, the action models apply to sensor values rather than features of the domain, and so objects and relations between objects are not modelled. Other work uses noiseless domains (Amir and Chang 2008; Yang, Wu, and Jiang 2007), and these methods have not been shown to work in noisy domains.

The form of partial observability can also vary. We follow (Amir and Chang 2008) where a fixed number of randomly chosen fluents are observed after each action in a random sequence of actions. Additionally, (Amir and Chang 2008) do not require knowledge of an initial state, fluents, or the size of the domain in their approach. In contrast, our method currently requires prior knowledge of the fluents and objects in the domain in order to build the state vector representation. (Yang, Wu, and Jiang 2007) describe a different form of partial observability, where the full state is observed at intervals after a fixed number of actions in a plan are executed, in combination with knowledge of the initial state and goal state. Since we rely on observing state changes before and after action application, our approach is not directly applicable to this form of partial observability.

We also believe the form of partial observability we use allows our model to be extended to sensing actions. While prior approaches (including ours) have primarily focused on learning the effects of ordinary actions—actions which

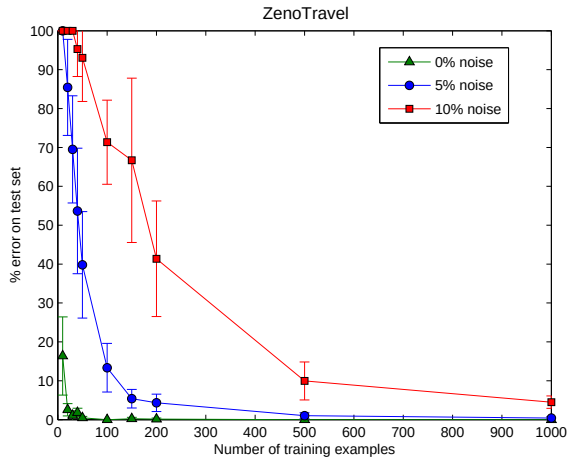


Figure 4: Learning results in noisy and fully observable versions of the ZenoTravel domain. Noise at level p% was simulated by flipping each bit in the state vector with probability p (for $p = 0\%$, $p = 5\%$ and $p = 10\%$). The test sets were noiseless, fully observable sequences of observations and actions, of length 2000.

change both the world state as well as the agent's knowledge state—they have ignored the role of *sensing actions* in partially observable domains. Sensing actions are particularly useful since they provide an agent with information about the state of the world, thereby changing the agent's knowledge state, but without necessarily changing the world state. We admit sensing actions into our account by assuming such actions have parameters (fluents and parameters of the fluents) and that these actions only alter knowledge relating to those parameters, or objects directly related to the parameters of the fluents. Furthermore we disallow sensing actions which are non-deterministic or have disjunctive effects. Then a reduced knowledge state vector can be constructed in the same manner as the reduced world state vector. In the input, the value of each bit indicates whether the corresponding fluent is known or not, and in the output indicates changes to the knowledge state. The learning model can then operate on both the knowledge and world state vectors. Standard actions are learnt as before, with effects now including changes to the knowledge state. Sensing actions are learnt in the same way, but changes to the world state will have to be ignored, since these are unpredictable. The addition of sensing actions would allow our method to be integrated with knowledge-based representations such as those used by the PKS planner (Petrick and Bacchus 2004).

Finally, the relative difficulty of learning action models in different domains is not well understood. For example, from the domain description, the Depots domain appears to be simpler than the ZenoTravel domain since the maximum number of parameters of any action is lower, and there are fewer predicates. Our results show that Depots is harder to learn, however, at least for our learning method. In fact, the method presented in (Yang, Wu, and Jiang 2007) also produces more errors in the Depots domain than in the ZenoTravel domain, suggesting that the additional difficulty may be a feature of the Depots domain rather than the learning method. In general, further investigation of the relative difficulties of learning different domains is important for further research in this area.

Conclusions and Future Work

We have presented a method for learning deterministic action models which is fast, scalable and handles partial observability of the world state. Furthermore, the error rate of the predictions made by the model is low. The speed, scalability and accuracy make the approach highly suitable for use in planning applications. It is straightforward to extend our method to learn the effects of sensing actions. Our method also performs well in noisy domains, and a key step will be to apply it to partially observable noisy domains.

In future work we plan to link our learning mechanism to a planning system applied to more complex domains, such as the problem of learning and planning actions for the ARMAR-III humanoid robot (Asfour et al. 2006) in a real-world robot environment. We also plan to extend the mechanism to learn more sophisticated action representations beyond STRIPS, such as those requiring functions.

Acknowledgements

This work was partially funded by the European Commission through the PACO-PLUS project (FP6-2004-IST-4-27657) and the UK EPSRC/MRC through the Neuroinformatics Doctoral Training Centre, University of Edinburgh.

References

Agre, P. E., and Chapman, D. 1987. Pengi: an implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, 268–272.

Aizerman, M. A.; Braverman, E. M.; and Rozoner, L. I. 1964. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control* 25:821–837.

Amir, E., and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research* 33:349–402.

Asfour, T.; Regenstein, K.; Azad, P.; Schröde, J.; and Dillmann, R. 2006. ARMAR-III: A humanoid platform for perception-action integration. In *2nd International Workshop on Human-Centered Robotic Systems (HCRS'06)*.

Ballard, D. H.; Hayhoe, M. M.; Pook, P. K.; and Rao, R. P. 1997. Deictic codes for the embodiment of cognition (with commentary). *Behavioral and Brain Sciences* 20.

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

Boser, B. E.; Guyon, I. M.; and Vapnik, V. N. 1992. A training algorithm for optimal margin classifiers. In *COLT*

'92: Proceedings of the fifth annual workshop on Computational learning theory, 144–152. New York, NY, USA: ACM Press.

Doğar, M. R.; Çakmak, M.; Uğur, E.; and Şahin, E. 2007. From primitive behaviors to goal directed behavior using affordances. In *Proceedings of Intelligent Robots and Systems (IROS 2007)*.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Freund, Y., and Schapire, R. 1999. Large margin classification using the perceptron algorithm. *Machine Learning* 37:277–296.

Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proceedings of the International Conference on Machine Learning (ICML-94)*. MIT Press.

Graepel, T.; Herbrich, R.; and Williamson, R. C. 2000. From margin to sparsity. In *Advances in Neural Information Processing Systems (NIPS) 13*, 210–216.

Holmes, M., and Isbell, C. 2005. Schema learning: Experience-based construction of predictive action models. In *Advances in Neural Information Processing Systems (NIPS) 17*, 585–562.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - the planning domain definition language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Metta, G., and Fitzpatrick, P. 2003. Early integration of vision and manipulation. *Adaptive Behavior* 11(2):109–128.

Minsky, M. L., and Papert, S. A. 1969. *Perceptrons*. The MIT Press.

Modayil, J., and Kuipers, B. 2008. The initial development of object knowledge by a learning robot. *Robotics and Autonomous Systems* 56(11):879–890.

Mourão, K.; Petrick, R. P.; and Steedman, M. 2008. Using kernel perceptrons to learn action effects for planning. In *Proceedings of the International Conference on Cognitive Systems (CogSys 2008)*.

Novikoff, A. B. 1963. On convergence proofs for perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, 615–622.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic world domains. *Journal of Artificial Intelligence Research* 29:309–352.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of Automated Planning and Scheduling (ICAPS-04)*, 2–11. AAAI Press.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* 65(6):386–408.

Satish, G., and Mukerjee, A. 2008. Acquiring linguistic argument structure from multimodal input using attentive focus. In *Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*, 43–48.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125(1-2):119–153.

Surdeanu, M., and Ciaramita, M. 2007. Robust information extraction with perceptrons. In *Proceedings of the NIST 2007 Automatic Content Extraction Workshop (ACE07)*.

Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of the International Conference on Machine Learning (ICML-95)*, 549–557.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence* 171(2-3):107–143.