Learning Regions for Building a World Model from Clusters in Probability Distributions

Witold Słowiński
Computing Science, University of Aberdeen, Scotland
r03ws8@abdn.ac.uk

Frank Guerin
Computing Science, University of Aberdeen, Scotland
f.guerin@abdn.ac.uk

***Abstract*—A developing agent learns a model of the world by observing regularities occurring in its sensory inputs. In a continuous domain where the model is represented by a set of rules, a significant part of the task of learning such a model is to find appropriate intervals within the continuous state variables, such that these intervals can be used to define rules whose predictions are reliable. We propose a technique to find such intervals (or regions) by means of finding clusters on approximate probability distributions of sensory variables. We compare this cluster-based method with an alternative landmark-based algorithm. We evaluate both techniques on a data log recorded in a simulation based on OpenArena, a three-dimensional first-person-perspective computer game, and demonstrate the results of how the techniques can learn rules which describe walking behaviour. While both techniques work reasonably well, the clustering approach seems to give more "natural" regions which correspond more closely to what a human would expect; we speculate that such regions should be more useful if they are to form a basis for further learning of higher order rules.**

I. Introduction

A developing agent may be faced with the problem of learning from scratch how to act in the world, based only on sensory observations. This involves building an internal model of how the world responds to actions and in general what rules it is governed by. In a world with continuous variables, with continuous time, and in which events happen independently of the actions of the agent this can be a complex problem.

We propose a technique for tackling a subset of this problem. Where the model consists of a set of rules describing the world, a major part of learning such rules is to find the right constraints on variables. Our technique aims to build such constraints by finding clusters on an approximate probability distribution of a variable. An intuitive justification for why such an approach might work is that in certain domains, variables will spend most time in certain attractive regions (with relatively high probability) and will move between those clusters through transition regions (with relatively low probability). Hence it is beneficial to aim to make rules which target these regions and find which other conditions must be met in order to get a variable into these regions.

We evaluate this technique and an alternative approach in a simulated environment based on a three-dimensional computer game and demonstrate the results. Section II describes the learning algorithm; Section III describes the evaluation; Section IV describes related work; Section V concludes and discusses future work.

II. Learning Rules

The objective of a model-learning algorithm is to allow an agent to learn a model of the world based on observations without having any prior knowledge of the particulars of the domain. It takes as input a sequence of states, where each state contains variables that represent the agent's observations. The output is a set of rules, where each rule applies in certain states and predicts aspects of the future, with a certain probability.

A. State and Model Representation

Our work is based on that of Mugan and Kuipers [1], [2]. The set of values that a variable can take is discretised into a finite set of intervals. Rules then incorporate predicates that test for the presence of variable values in intervals. However, our algorithm abandons the idea of maintaining a complete partition of value ranges into non-overlapping sets as part of a qualitative space.

The agent's observations are represented as a set of states identified by contiguous natural integers: $\mathcal{S} \subset \mathbb{N}$. Each state contains a mapping from variable identifiers to values: $value : \mathcal{V} \times \mathcal{S} \rightarrow \mathbb{R}$. The value of the variable v in state i is denoted by $v_i = value(v, i)$. Each state contains a $time$ value, representing the time, in seconds, elapsed since the first observed state. This information is used to construct a set of rules. Each rule $r \in \mathcal{R}$ is a tuple $r = \langle pre(r), post(r), deadline(r) \rangle$, where $pre(r)$ is the precondition, $post(r)$ is the postcondition and $deadline(r) \in \mathbb{R}$ is the rule's deadline in seconds. The postcondition is a predicate $p \in \mathcal{P}$, where p is a function $p : \mathcal{S} \rightarrow \{true, false\}$. The precondition is a set of predicates, interpreted together as a logical conjunction, and the deadline states that no more than the given amount of time can pass until the postcondition is true. In other words, a rule is a statement saying that, given a state where all the predicates in the precondition are true, there will come a state before the deadline passes in which the postcondition holds.

This structure of a rule is notably different from the one in [2] in that the precondition and postcondition are used to select variable values, and do not require that an event takes place, i.e. that a variable changes value between the previous and current states. It is dependent on the domain whether representing events in rules is required and in our experiments we found it to be unnecessary.

In our work, all predicates in rules contain a variable identifier $v \in \mathcal{V}$ and an interval c, and are of the form:

$$p_{v,c}(i) = \begin{cases} true & \text{if } v_i \in c \\ false & \text{otherwise} \end{cases}.$$

Additionally, in order to reason about deadlines, given some predicate p and deadline d, let the predicate $within_{d,p} : \mathcal{S} \rightarrow \{true, false\}$ be defined as:

$$within_{d,p}(i) = \begin{cases} true & \text{if } \exists j \in \mathcal{S} \text{ s.t.} \\ & \quad p(j) \wedge 0 \leq time_j - time_i < d \\ false & \text{otherwise} \end{cases}$$

The probability of a predicate being true is defined as:

$$\mathrm{P}(p) = \frac{|\{i \in \mathcal{S}\colon p(i) = true\}|}{|\mathcal{S}|},$$

and the conditional probability of b, given that a is true, as:

$$\mathrm{P}(b \mid a) = \frac{|\{i \in \mathcal{S}\colon a(i) = b(i) = true\}|}{|\{i \in \mathcal{S}\colon a(i) = true\}|}.$$

Hence, the probability of success of a rule r is defined as $\mathrm{P}(r) = \mathrm{P}(within_{deadline(r),post(r),} \mid pre(r))$.

B. Compressible Distributions

A compressible distribution is a simple data structure which stores an approximate probability distribution of a real-valued variable and can be used to derive clusters from such a distribution. Approximation is needed for two reasons. With continuous variables the number of distinct values can become large and it is necessary to store an approximate distribution in order to provide an upper bound for memory use. Additionally, since with such variables most values are never sampled, there is a need to generalise over unseen values based on those seen.

A compressible distribution d is a tuple $\langle entries(d), count(d)\rangle$, containing a sorted set of non-overlapping *entries* and the total number of samples inserted into it. Each entry e is a tuple $\langle lower(e), upper(e), count(e)\rangle$ describing a real interval and the number of samples seen in this interval. An entry e is said to be *single* if $lower(e) = upper(e)$. If an entry e is single, it encompasses the single-valued set $\{lower(e)\}$, otherwise it encompasses the interval $[lower(e), upper(e))$. We denote the interval encompassed by an entry e by $interval(e)$. Also let $height(e)$ be equal to $count(e)$ if the entry is single and $\frac{count(e)}{upper(e) - lower(e)}$ otherwise. Every distribution can contain a maximum number of entries, in our case set to 64.

1) Adding a Sample: When a sample s is added to distribution d, if there exists an entry e, s.t. $s \in interval(e)$, then $count(e)$ is incremented. Otherwise, if there is space for more entries, a new entry $\langle s, s, 1\rangle$ is added to the set of entries. If there is no space, then some two entries are merged and the procedure is restarted. Finally, $count(d)$ is incremented.

2) Merging: Merging is performed in order to reduce the number of entries in the distribution. When merging, two consecutive entries e_1, e_2 are selected, with e_2 being non-single, so as to minimise $score2(e_1, e_2) = |height(e_1) - height(e_2)|$. These entries are removed and a new entry $\langle lower(e_1), upper(e_2), count(e_1) + count(e_2)\rangle$ is added. If two such entries are not found, then three consecutive entries e_1, e_2, e_3 are selected, so as to minimise:

$$score3(e_1, e_2, e_3) = \begin{cases} score2(e_1, e_2) & \text{if } e_3 \text{ single} \\ score2(e_1, e_2) + score2(e_2, e_3) & \text{otherwise} \end{cases}.$$

Then e_1 and e_2 are removed and a new entry $\langle lower(e_1), lower(e_3), count(e_1) + count(e_2)\rangle$ is added. e_3 is only used to donate the upper bound.

3) Compressing: The merging process can be used both to decrement the number of entries and to derive clusters of regions of similar probability. Given a probability threshold t, compressing the distribution consists of iteratively performing the merge operation while the absolute probability difference between the penultimate and last entry selected in a merge is less than t, with the absolute probability difference of entries a, b in distribution d defined as $\frac{|height(a) - height(b)|}{count(d)}$. We set $t = 0.05$. After compressing is complete, the remaining entries describe clusters, where for each cluster its values are considered to have similar probability.

C. Searching for Intervals

With a state and model representation as presented above, the problem becomes a matter of finding the right variables and intervals for rules. We describe two possible methods for achieving this: a novel method based on learning clusters on generalised variable probability distributions, and a method based on landmark-finding algorithms in [2]. In Section III compare the results obtained using the two methods.

D. Cluster-based Search

With the cluster-based search, we first find intervals to use as postconditions of rules and use them to create rules with empty preconditions. Then we iteratively specialise these rules by making copies with added preconditions. The process ends after no further improvement is possible. The algorithm works in several stages:

1) Finding Possible Causal Relations Between Variables: As the first step, the algorithm looks for variables that change close together in time, at the same time or one after the other, which gives it a partial order over variables that is assumed to only include those pairs of variables between which there is a causal relationship. Let variable $v \in \mathcal{V}$ and let $changed_v$ be a predicate such that:

$$changed_v(i) = \begin{cases} true & \text{if } v_{i-1} \neq v_i \\ false & \text{otherwise} \end{cases}.$$

Then, two variables v, v' are related if:

$$\mathrm{P}(within_{\vartheta, changed_{v'}} \mid changed_v) - \mathrm{P}(within_{\vartheta, changed_{v'}}) > \theta,$$

where ϑ is a deadline parameter (0.11s) and θ is a probability difference threshold parameter (0.05). Denote this relation as $related(v, v')$. During this step, the algorithm computes this for all pairs of distinct variables.

2) Computing Approximate Variable Distributions: Independently, the algorithm computes an approximation of the probability distribution of each variable using a compressible distribution data structure (see Section II-B). The end product is an approximation of the distribution $\Delta_v : \mathbb{R} \to [0, 1]$, defined as $\Delta_v(x) = \mathrm{P}(v_i = x)$.

3) Creating Rules with Empty Preconditions: After variable distributions are computed, each variable's distribution is *compressed* (see Section II-B3) to produce clusters in the form of real-number intervals. For each variable v and each of its clusters c, the rule $\langle \emptyset, v_i \in c, 0 \rangle$ is created and added to the set of all rules.

4) Specialising Rules: For the purposes of specialising rules, the set of rules with empty preconditions becomes the first *generation.* From each generation, a subsequent generation is produced. New rules arise from rules from the previous generation with one precondition predicate added, so that all rules in the nth generation have $n - 1$ predicates in the precondition. Once a generation with no rules occurs, the process is stopped. Rules from every generation are added to the set of all rules.

In a given iteration of the algorithm, a specialisation procedure is invoked for every rule r in the current generation (see Algorithm 1). This procedure first computes, for each variable $related$ to $post(r)$, an approximate probability distribution $D_{v,r}(x) = \mathrm{P}(v_i = x \mid post(r))$, which is stored in a compressible distribution data structure. As the next step it creates clusters from this distribution and, for each cluster c, proceeds to create and evaluate a new rule, derived from the current rule but with an added precondition predicate of $v_i \in c$. At this point, the new rule r' is evaluated (see Algorithm 2). If there are at least ω occurrences and $\mathrm{P}(r') - \mathrm{P}(r) \geq \theta$, the evaluation continues, otherwise r' is discarded.

As a final condition, there cannot be a better rule for making the same prediction, which in this context is taken to be another rule with the same deadline, the same postcondition, a subset of the precondition predicates of r' and exactly the same probability as r'.

The necessity for this last test is justified by the following example: suppose there exists a rule $r_1 = \langle \{k\}, z, 0 \rangle$ with predicate k selecting a small number of states. Then, in the next generation, another rule $r_2 = \langle \{l, m\}, z, 0 \rangle$ arises. A generation afterwards r_2 might lead to $r_3 = \langle \{k, l, m\}, z, 0 \rangle$. If l and m together select a superset of the states selected by k then the precondition of r_3 will select the same states as that of r_1 and their probabilities will be the same. In this case, clearly, l and m are not needed, which makes r_3 redundant.

All of the above-mentioned evaluation steps are performed for two variants of r': one with a deadline of 0 and one with a deadline of ϑ: it is possible for zero, one or two new rules to be created from the starting rule r, variable v and interval c, depending on the outcome of the evaluation steps. Each newly-created rule will be more reliable than its predecessor, in terms of its success probability.

Algorithm 1 SPECIALISE-RULE(r)

```
Parameters: r - rule
for all v ∈ V do
    if pre(r) does not refer to v and related(v, post(r)) then
        {compute conditional distribution:}
        D_{v,r}(x) ← P(v_i = x | post(r))
        {compute clusters from distribution:}
        C_v ← COMPRESS(D_{v,r})
        for all c ∈ C_v do
            SPECIALISE-RULE-WITH-PREDICATE(r, v, c)
        end for
    end if
end for
```

Algorithm 2 SPECIALISE-RULE-WITH-PREDICATE(r, v, c)

```
Parameters: r: rule, v: variable, c: interval
References: N: next generation of rules, R: set of all rules
p ← v_i ∈ c {make new predicate}
r' ← ⟨pre(r) ∪ p, post(r), 0⟩ {make new rule}
{enforce minimum number of occurrences:}
if |{i ∈ S s.t. (pre(r))(i) = true}| < ω then
    return
end if
{evaluate rule with no deadline:}
if P(r') − P(r) ≥ θ and not EXISTS-BETTER-RULE(r') then
    R ← R ∪ r'
    N ← N ∪ r'
end if
{evaluate rule with deadline ϑ:}
r' ← ⟨pre(r'), post(r'), ϑ⟩
if P(r') − P(r) ≥ θ and not EXISTS-BETTER-RULE(r') then
    R ← R ∪ r'
    N ← N ∪ r'
end if
```

E. Landmark-based search

This interval search method is derived from the landmark-search algorithms by Mugan [2], but adapted to work with the world and model representation described in this paper for the purpose of comparing the results of both methods.

The main feature of this approach is that it maintains, for each variable, a partition of the real numbers into a set of non-intersecting intervals separated by landmarks. Intervals and landmarks together form a variable's *qualitative space*. This adaptation implements landmarks for variable v with a set of real numbers L_v. The qualitative space is derived from the landmarks by including all intervals $[l, l]$ (also denoted $\{l\}$) containing only landmarks, together with all open

intervals between each two consecutive landmarks and the open intervals $(-\infty, s)$ and $(g, +\infty)$, where s and g are the least and greatest landmarks, respectively. For example, with a landmark set of $L_v = \{l, m\}$, the qualitative space for v is $Q_v = \{(-\infty, l), \{l\}, (l, m), \{m\}, (m, +\infty)\}$.

Intervals in rule predicates can only be taken from the qualitative space. Since the latter is fully derived from the set of landmarks, the problem is transformed into one of finding the right landmarks for a variable.

Like the former method, this approach works in several stages:

1) Finding Possible Causal Relations Between Variables: This stage is identical to the one described in Section II-D1. It finds pairs of variables between which there is a possible causal relation and marks such pairs v, v' as $related(v, v')$.

2) Setting Initial Landmark Values: All variables are given the same initial landmark set containing only zero: $L = \{0\}$.

3) Iterative Landmark Search: This stage attempts to find new landmarks for each variable, based on existing landmarks belonging to another variable. If variable v is related to v' and v' is in a qualitative interval c, it is possible that the probability distribution of v given that $v' \in c$ will be noticeably different from the distribution of v. This stage makes note of such differences and introduces new landmarks at points where this difference is sufficiently large.

For each variable v' and interval $c \in Q_{v'}$, for each other related variable v, this part of the algorithm computes the distribution of v given that $v' \in c$. This distribution is stored discretely in an array made up of bins, where the bin size is two times the average absolute change of v. A change is assumed to take place only when $|v_{i+1} - v_i| > 0.001$. Then it computes the distribution of v and stores it in another array with exactly the same partition into bins. Finally, the corresponding bins are compared to expose possible differences between the two distributions. Each bin b that meets the condition that:

$$\mathrm{P}(v \in b \mid v' \in c) - \mathrm{P}(v \in b) > \theta_E$$

(where $\theta_E = 0.2$) is considered as a candidate for creating new landmarks. Of these bins, the one with the highest score is selected. Its lower and upper bounds are added as new landmarks, provided that they are at least one average absolute change away from any existing landmark.

Once this procedure is completed for all v', it is iteratively repeated on all variables with their updated landmark sets. Iterations are ceased when no new landmarks can be added.

4) Creating Rules with Empty Preconditions: This stage is analogous to that described in Section II-D3, except that the intervals in postconditons on variable v are taken from its qualitative space Q_v instead of from clusters on approximate distributions. The outcome is one new rule per every variable per every element of its qualitative space.

5) Specialising Rules: Analogously to the stage described in Section II-D4, rules are specialised generation by generation. However, similar to the previous stage, the intervals for preconditions are not clusters on approximate distributions but are taken from a variable's qualitative space. New generations contain rules presenting an improvement in reliability over their predecessors and the procedure is finished when no improved rules can be constructed.

6) Narrowing Down Intervals: As the last step, for every rule r, for every predicate in $p \in pre(r) \cup post(r)$, the algorithm, using the method of Fayyad and Irani [3], attempts to find a cut-point, which splits the predicate's interval into two parts, of which only the better is preserved in the rule. This results in a new rule replacing the old rule, provided that such a cut-point can be found.

Given r and p, the algorithm first takes the variable v contained in the predicate p and computes the distribution of v, given that $pre(r)$ holds, using a compressible distribution structure. Then, for each of its entries, it evaluates the cut points at the lower and upper bounds of the entry. Given a cut-point k, it partitions the set of states i selected by $pre(r)$ into those where $v_i < k$ and those where $k \leq v_i$. In other words, $S = \{i \in \mathcal{S}\colon (pre(r))(i) = true\}$, $S^- = \{i \in S\colon v_i < k\}$ and $S^+ = \{i \in S\colon k \leq v_i\}$. Using these sets, we define the information gain of putting a cut-point at k as:

$$G_k = \mathrm{H}(S) - \frac{|S^-|}{|S|}\mathrm{H}(S^-) - \frac{|S^+|}{|S|}\mathrm{H}(S^+),$$

where $\mathrm{H}(X) = -\sum_{x \in X} \mathrm{P}(x) \log_2(\mathrm{P}(x))$ is defined as the entropy of set X. The cut-point k^* with the highest G_k is selected, provided that $G_{k^*} > \theta_G$ and $2G_{k^*}\,\mathrm{P}(v_i \in S) > \theta_G$, where θ_G is a threshold parameter (0.25). If so, two candidate rules r_1 and r_2 are considered, in which the interval of p is replaced with S^- and S^+, respectively. If $\mathrm{P}(r_1) - \mathrm{P}(r) > \theta$ or $\mathrm{P}(r_2) - \mathrm{P}(r) > \theta$, then k^* is added as a new landmark to L_v and r is replaced with the more successful of r_1 and r_2.

III. Evaluation

We evaluated our work by recording state sequences in the program OpenArena [4]. We then used the recorded logs as input for the cluster-based and landmark-based algorithms described in Section II-D and Section II-E. We then compared the results by inspecting the resultant rules and landmarks.

Our test domain is the first-person-perspective three-dimensional shooter game OpenArena, chosen because it is a complete and challenging environment to learn rules in. In the game, the player controls a character who can walk, run and jump in a three-dimensional level and interact with objects and other players. For the purposes of this evaluation, we aimed to provide data that would allow the learning of a set of rules that can sufficiently describe the player's walking behaviour. We modified the source code of OpenArena 0.8.1 to log pertinent state information at the beginning of each frame displayed on the screen. We used this capability to record the player moving forward and backward repeatedly along a straight line while varying the key presses so that many possible combinations of the variables would be present in the log. The total number

of states in the log was 3610, with a running time of about one minute.

After capturing this data, we modified it to include an acceleration variable, derived from changes in velocity. As it turned out, the game only updates its physics once every five states (or under 100ms), so in order to avoid sharp spikes the in acceleration, we implemented a running average over five states, which reduced the spikes but introduced a slight delay in the response of the acceleration to the velocity. Figure 1 depicts a plot of the test data, with keypress variables (in_forward, in_back) multiplied for clarity to be in the sets {0,80} and {0,100} instead of {0,1}.

In order to walk forward in OpenArena, the player needs to press the forward button. Until the player's speed is at maximum (320), it will cause acceleration to be greater than zero. Acceleration becomes zero either when the player is not pressing any button or if the player is moving at maximum velocity (or walking backwards at maximum negative velocity (-320)). We expected the learning algorithms to develop rules providing a similar description.

A. Results

Table I summarises the results obtained from each of the two algorithms. The results show that both algorithms can create reliable rules in a domain such as the test domain. The cluster-based algorithm created about half as many rules as the landmark-based algorithm. On average, the rules created by the cluster-based algorithm had a higher success probability, but among rules with probability above 0.75, the average success probabilities became close. This is because most of the rules created by the landmark-based algorithm (372 out of 510 or 73%) had a probability of success lower than 0.75, compared to 135 out of 250, or 54%, in the case of the cluster-based algorithm. However, in absolute terms, the landmark-based algorithm created a greater number of reliable rules.

These results can be explained using the fact that the landmark-based algorithm performed an iterative narrowing of elements of the qualitative spaces, while the cluster-based algorithm computed broader clusters without refining them.

Figure 2 contains example rules learnt by each of the algorithms. What is apparent is that the cluster-based algorithm places interval boundaries at more natural locations, while the boundaries of the landmark-based algorithm can be placed at more arbitrary locations, due to the coarseness of the bins during the search for new landmarks. In particular see the rule marked # which describes the case where velocity has hit its maximum and so acceleration will soon be zero. See also the rule marked $\star$ which describes the case where no key is pressed and so the player will decelerate. The cluster method creates rules which provide a clearer general model, while the landmark method makes a fine-grained model more suitable for planning. We speculate that the cluster method should be more useful if it is to form a basis for further learning of higher order rules, because the regions seem to be getting closer to the underlying rules which generate the simulated data.

	Cluster-based	Landmark-based
rules	250	510
avg. probability	0.67	0.55
rules above 0.25 probability	232	440
avg. probability above 0.25	0.71	0.61
rules above 0.5 probability	173	264
avg. probability above 0.5	0.82	0.77
rules above 0.75 probability	115	138
avg. probability above 0.75	0.92	0.90

Table I
SUMMARY OF RESULTANT RULE SETS

IV. Related Work

The closest work to ours is that by Mugan and Kuipers [1], [2]. They describe a comprehensive framework for learning rules over a qualitative space using landmarks, constructing an action hierarchy and using the actions for planning.

Drescher [5] proposed an approach to creating schemas as rules for modeling the world. Schemas are learnt by starting with bare schemas which contain an action, and are then improved by including effects and contexts. They do not tackle continuous variables however.

Jong and Stone [6] describe a method for learning a model of a continuous domain using averagers. Regions are encoded using a probability mass function which is utilised by the averager. However, states are treated as atomic multidimensional vectors and there are no explicit rules referencing the individual variables in a state.

Provost et al. [7] describe a technique for finding clusters in features of the state space using Self-Organising Maps [8] for solving robot navigation problems.

The technique proposed by Goodman et al. [9] builds a probabilistic model of the world using a discretisation of the state space into square regions.

Pasula et al. [10] describe a technique for learning a probabilistic model of a discrete domain with states described using relational predicates. Their algorithm involves a comprehensive set of operators that manipulate rules in order to improve the model. The rules are evaluated similarly as in our work. Another technique for learning planning operators in a probabilistic relational STRIPS-like domain is proposed by Safaei et al. [11]. Despite a simple language used to define rules, it shares similarities, such as finding superfluous predicates in rules.

Croonenborghs et al. [12] propose a technique for learning probabilistic models of relational domains with Dynamic Bayesian Networks, with conditional probability distributions modeled similarly to our work. For each atom, its conditional probability distribution is modeled using a relational probability tree, learnt using the tree learning algorithm TG.

V. Conclusions and Future Work

We have presented a novel approach to learning predicates for rules that works by learning clusters over an approximate probability distribution and demonstrated results in a practical domain. While this technique needs to be complemented by other rule manipulation techniques in order to produce a more

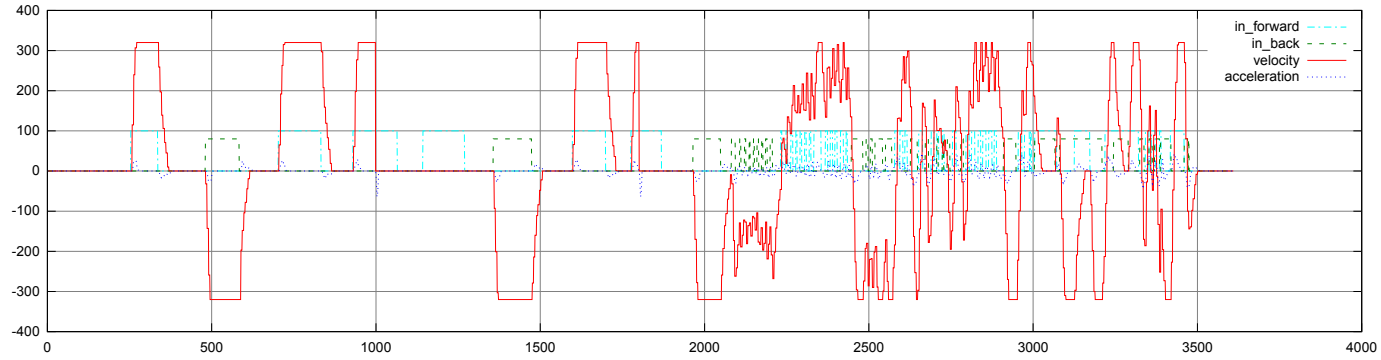

Figure 1. Test data

Cluster-based:

```
  0.772397: (<in(velocity,{-320})> => in(acceleration,{0})
  0.859649: (<in(in_back,{1}),in(acceleration,{0})> => in(velocity,{-320})
# 0.924282: (<in(in_forward,{1}),in(velocity,{320})> => within(0.11s):in(acceleration,{0})
* 0.996997: (<in(in_forward,{0}),in(velocity,[1,297)),in(in_back,{0})> =>
             within(0.11s):in(acceleration,[-64,0))
  1       : (<in(velocity,[1,297)),in(in_back,{1})> => within(0.11s):in(acceleration,[-64,0))
  1       : (<in(velocity,{0})> => within(0.11s):in(acceleration,{0})
```

Landmark-based:

```
# 0.703777: (<in_interval(in_forward,(0,inf)),in_interval(velocity,(216.545,inf))> =>
             within(0.11s):in_interval(acceleration,{0})
  0.859649: (<in(in_back,(0,inf)),in(acceleration,{0})> => in(velocity,{-320})
* 0.900524: (<in(in_back,{0}),in(in_forward,{0}),in(velocity,(82.4091,216.545))> =>
             within(0.11s):in(acceleration,(-12.2004,-5.72373))
  0.992248: (<in(in_back,(0,inf)),in(velocity,{-320})> => within(0.11s):in(acceleration,{0})
  1       : (<in(velocity,{0})> => within(0.11s):in(acceleration,{0})
```

Figure 2. Example rules learnt by the rule-learning algorithms

finely-grained model, it is sufficient to produce a reliable broad model of a domain.

There are several aspects in which cluster-based rule learning could be improved. One important aspect is that the learning is performed off-line from a log of states. In a fully-developmental agent this should be performed on-line.

A general drawback of discretising continuous spaces is the difficulty of capturing linear relationships between variables in variables that are uniformly distributed. It seems inefficient to discretise such variables into small intervals: a predicate expressing a linear relationship would be more suitable for such a case. In ongoing work we are addressing this by explicitly representing relationships in rules, and so the work starts to become more like relational learning approaches [10].

When considered independently from the single-dimensional distribution structure, the cluster-based technique could be generalised to two- or three-dimensional vector variables. In the future we plan to implement distribution approximation and clustering supporting higher-dimensional values. Especially when considering vector variables, it would be useful to synthesise new variables by combining existing variables in certain ways: subtracting two vectors, taking the length of a vector or automatically calculating a derivative of a variable might improve autonomy.

REFERENCES

[1] J. Mugan and B. Kuipers, "Learning to predict the effects of actions: Synergy between rules and landmarks," in *In Proceedings of the 6th (IEEE) International Conference on Development and Learning*, 2007.

[2] J. Mugan, *Autonomous Qualitative Learning of Distinctions and Actions in a Developing Agent*. PhD thesis, University of Texas at Austin, 2010.

[3] U. Fayyad and K. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *International Joint Conference on Artificial Intelligence*, 1993.

[4] OpenArena, "OpenArena." http://openarena.ws/, 2011.

[5] G. L. Drescher, *Made-up minds: a constructivist approach to artificial intelligence*. Cambridge, MA, USA: MIT Press, 1991.

[6] N. K. Jong and P. Stone, "Model-based function approximation in reinforcement learning," in *AAMAS*, p. 95, 2007.

[7] J. Provost, B. J. Kuipers, and R. Miikkulainen, "Self-organizing distinctive-state abstraction learns perceptual features and actions," 2004.

[8] B. Fritzke, "A self-organizing network that can follow non-stationary distributions," 1997.

[9] N. D. Goodman, V. K. Mansinghka, and J. B. Tenenbaum, "Learning grounded causal models," in *In Proceedings of the Twenty-Ninth Annual Conference of the Cognitive Science Society*, 2007.

[10] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling, "Learning symbolic models of stochastic domains.," *J. Artif. Intell. Res. (JAIR)*, vol. 29, pp. 309–352, 2007.

[11] J. Safaei and G. Ghassem-Sani, "Incremental learning of planning operators in stochastic domains," in *SOFSEM '07, Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science*, pp. 644–655, Springer-Verlag, 2007.

[12] T. Croonenborghs, J. Ramon, H. Blockeel, and M. Bruynooghe, "Online learning and exploiting relational models in reinforcement learning," in *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 726–731, 2007.