Inferring dependencies in Embodiment-based modular reinforcement learning

David Jacob, Daniel Polani, Chrystopher L. Nehaniv
Adaptive Systems Research Group, University of Hertfordshire
College Lane, Hatfield, Herts AL10 9AB, UK
D.Jacob, D.Polani, C.L.Nehaniv@herts.ac.uk

Abstract

The state-spaces needed to describe realistic physical embodied agents are extremely large, which presents a serious challenge to classical reinforcement learning schemes. In previous work (Jacob et al., 2005a, Jacob et al., 2005b) we introduced our EMBER (for EMbodiment-Based modulaR) reinforcement learning system, which describes a novel method for decomposing agents into modules based on the agent's embodiment. This modular decomposition factorises the state-space and dramatically improves performance in unknown and dynamic environments. However, while there are great advantages to be gained from a factorised state-space, the question of dependencies cannot be ignored. We present a development of the work reported in (Jacob et al., 2004) which shows, in a simple example, how dependencies may be identified using a heuristic approach. Results show that the system is able quickly to discover and act upon dependencies, even where they are neither simple nor deterministic.

1. Introduction

The state-spaces needed to describe realistic physical embodied agents are extremely large. This presents a serious challenge to classical reinforcement learning schemes, which model the agent and environment as a monolithic entity with every variable defining the state represented as a separate dimension in the state-space. In previous work (Jacob et al., 2005a, Jacob et al., 2005b) we introduced our EMBER (for EMbodiment-Based modulaR) reinforcement learning system, which describes a novel method for decomposing agents into modules based on the agent's embodiment. The modules are able to sense, act and generate reinforcement reward locally, and to learn from this reward.

The factorisation of the state-space which this decomposition induces has several advantages:

1. the agent is now represented independently of the environment, so the combined state space is $|\mathcal{S}_A| + |\mathcal{S}_E|$ rather than $|\mathcal{S}_A| \times |\mathcal{S}_E|$, where $\mathcal{S}_A$ and $\mathcal{S}_E$ are the states needed to represent the agent and the environment respectively
2. the agent's own state-space is similarly the sum of the state-spaces of its modules, which greatly reduces the agent representation size
3. rather than learn only a single task, as in classical reinforcement learning, the agent also learns about the world at the same time from its interactions with the environment: this knowledge becomes incorporated in the agent and is available to help with the learning of new tasks as required. By contrast, classical reinforcement learning learns every task *tabula rasa*, and no transferable knowledge is acquired: everything has to be learned from scratch every time.
4. as a result, performance in dynamic environments and on new tasks is greatly improved: the modular agent is able to generalise to previously unvisited states from locally-similar states already experienced. Thus it is able to avoid actions it suspects to be bad without having to try them first, which gives advantages both in quality of performance and in learning speed.

However, while there are great advantages to be gained from a factorised state-space, the question of dependencies cannot be ignored. It is possible to an extent to treat the modules as independent agents, selecting actions on the basis that similar local observations require similar local actions irrespective of global state; but it is clear that where the modules do not sense and act orthogonally with respect to one another, this assumption must be modified. It remains, however, a central tenet of the EMBER framework that dependencies are generally low-dimensional, and that even allowing for the complexity of identifying them and taking them into account, the advantages of the modular approach remain substantial.

We present a development of the work reported in (Jacob et al., 2004) which shows, in a simple example, how dependencies may be identified and acted upon. To avoid having to perform unguided search through potentially very large agent state-spaces, which would be both unscalable and against the spirit of the modular decomposition, a heuristic approach is taken. Results show that the system is able quickly to discover and act upon dependencies, even where they are neither simple nor deterministic.

2. Related Work

2.1 Reducing the size of the state-space

Because the sheer size of the state space in classical RL formulations makes learning slow, many methods have been proposed for reducing the size of this state-space. This has often been done by exploiting task structure: for example, McCallum's *utile distinctions* (McCallum, 1993) distinguishes between states on the basis of their utility in the context of the current task, and to that extent generalises between spatially-distinct states with the same utility. The process used however is computationally expensive and does not make use of any intrinsic properties of the problem under consideration. It starts with no distinction between states and its early learning is therefore entirely random. As a consequence it is too general in application to be particularly suited to embodied systems; further, being explicitly task-based, it cannot help us when the task is changed.

2.2 Multiple sources of reinforcement reward

The principle of combining multiple sources of reinforcement reward is treated in (Shelton, 2000), but this differs from the current work in that the sources are themselves considered to be agents, competing to influence the outcomes of an overall policy. Shelton's work therefore fits better within the established framework of multi-agent RL, for example (Hu and Wellman, 1998).

2.3 Reward shaping through multiple rewards

Multiple sources of reward are also used in the form of subsidiary rewards to bias system behaviour, so-called 'reward shaping' (Ng et al., 1999). However, the authors here are aiming more towards the introduction of heuristic reward to provide dense reward functions, which although difficult to construct may give performance advantages (Smart and Kaelbling, 2002). Although superficially similar to some aspects of the current work, there is no direct intervention in the action selection process of the learning algorithm, and thus no a priori generalisation between unvisited global states.

2.4 Hierarchical task decomposition

Hierarchical RL is another area which has attracted much research. The term "Modular Reinforcement Learning" is often used in this context: a large overall task is decomposed into smaller tasks which can be individually learned, for example (Dietterich, 2000), the module-based RL of (Kalmár et al., 1998), and the options framework introduced by (Sutton et al., 1999). This approach can give benefits in speed of learning and the availability of training examples for sub-tasks which occur multiple times. Unfortunately the decomposition is hard to achieve autonomously, although some success has been attained by state occupation frequency analysis (McGovern and Barto, 2001) and, in certain situations, by dimensional discrimination (Hengst, 2002). In general, however, it appears that domain or world knowledge may be required: the Hierarchy of Autonomous Machines (Parr and Russell, 1997) is an example. Also notable are sequential decompositions (Morimoto and Doya, 1998), where intermediate sub-goals assist in the performance of larger tasks by effectively limiting divergence from a desired trajectory.

EMBER, the framework for reinforcement learning of which this work forms a part, differs from much of the above in that it is explicitly designed for embodied agents acting in the physical world, or models thereof. The factoring out of the agent from the world, and the modules in turn from the agent, attacks the problem of the combinatorial explosion of state spaces by maintaining a separation between learning a particular task and learning basic competences which facilitate any task. In addition, by using the tools of reinforcement learning but augmenting them with logical inference and manipulation of the learning process using the novel algorithms Factor-Q (Jacob et al., 2004) and Hard Constraint Modular Learning (HCML) introduced here, it becomes possible to make use of heuristics and to predict outcomes through generalisation. This substantially reduces the quantity of examples required, leading to faster and more efficient learning.

3. Description and purpose of experiments

The experiments were based on an eight-connected gridworld, a small section of which is shown in figure 1*(a)*. The agent is octagonal, and can move in eight directions: its orientation is constant.

The agent is decomposed into eight modules, whose internal structure is shown in figure 1*(b)*, together with *(c)* their configuration within the agent. Each module is equipped with a pair of sensors: a proximity sensor, which outputs 1 if an obstacle (or the wall) occupies the adjacent cell, otherwise 0, and an impact sensor, which outputs 1 only when hit. The actuator within the module, when fired, attempts to move the agent one cell in the direction of the arrow. (The sensors may be implemented within one physical sensor, and actuators may not be physically separate, but for the purposes of explanation it is helpful to think of them as described).

Figure 2 illustrates which actions succeed or fail. Referring to the diagram, action 1 fails; the impact sensor

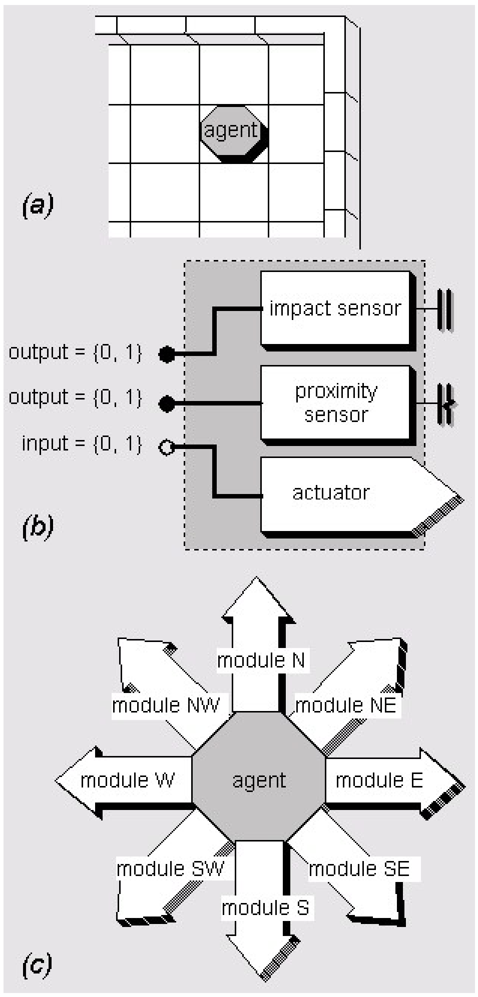

Figure 1: Part of the gridworld, one module, and the layout of the modules within the agent

in module NW registers a hit. Action 2 succeeds; a clear cell along a row or column of the grid can always be moved into. Action 4 fails; only an obstacle where B is will cause this action to fail. 5 fails, but for a different reason from 1; obstacles in the same places as B and D together will always prevent a diagonal action such as this one from succeeding, *irrespective of whether obstacle C is present.* In these circumstances, exactly one of module E and module S will register a hit with equal probability: module SE is not hit. By contrast, action 3 succeeds since there is only one cell occupied (by obstacle B) next to the destination cell.

The agent therefore has to learn that "diagonal" actions, that is, actions by modules NE, SE, SW and NW, will fail in cases 1 and 5 (and 5 also where obstacle C is not present) but will succeed in case 3, whether or not B is present. Further, in case 1 module NW sustains the hit; in case 5, where the local observation is the same, module SE is not hit (although the action fails).

We need a system which is capable of learning dependencies of this kind, quickly and using a minimum of data, since every positive example costs one collision, and we wish to minimise these.

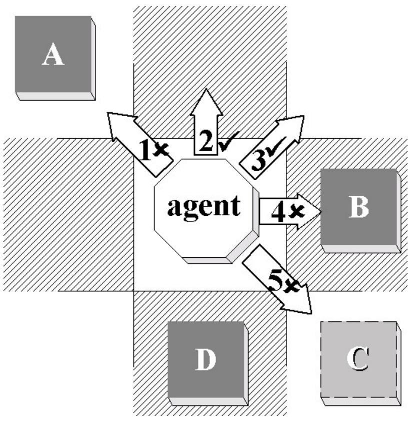

Figure 2: Showing the dependency rules which govern the gridworld. Action 5 fails whether or not obstacle C is present. See text for details.

We will describe the principle of operation of the modules ignoring inter-module dependencies before explaining how the system is extended to allow dependencies to be identified and acted upon.

4. Modular learning using HCML

We make the standard assumptions that states, actions and time are discrete, and that the sets of states and actions are finite.

Modules learn to predict the local effect of their actions as follows. Each module m in the set of modules $\mathcal{M}$ maintains a table of predicted immediate local reward $Er_m(s_m, a)$ being the expected immediate reward for m under agent's action a. If a is not an action associated with m, this reward is defined as zero. Here, s_m is the internal state of module $m \in \mathcal{M}$ corresponding to the current modular observation o_m, and "local" rewards are those generated internally by the agent, in this case from the impact sensors. $\vec{\rho}$ is the vector of projected immediate local reward summed over the modules such that

$$\rho_a = \sum_{m \in \mathcal{M}} Er_m(s_m, a). \qquad (1)$$

At the same time, the world external to the agent is represented by a set of states $\mathcal{S}$. The agent, which for current purposes we assume knows which state it is in, maintains a table of utility $q(s, a)$ for each action a in each state $s \in \mathcal{S}$. Here, the rewards r relate only to the fulfilment of the current task and are administered externally - they constitute the means by which the agent is directed, exactly as in conventional reinforcement learning.

The vector $\vec{Q}(s_\circ)$ of combined expected future discounted reward on which the agent will base its action selection is given by:

$$\vec{Q}(s_\circ) = F\vec{q}(s) + G\vec{\rho} \tag{2}$$

where $F \in \{0,1\}$ and G is a normalisation factor.

F and G are calculated as follows:
We first find the lowest world q-value $q_{\min}$:

$$q_{\min} = \min_a q(s,a).$$

Similarly, we find the lowest projected local reward $\rho_{\min}$:

$$\rho_{\min} = \min_a \rho_a.$$

Now, to detect unvisited states whilst allowing for the effects of noise,

$$\text{if} \quad q_{\min} > -\eta, \quad F = 0$$

.
In this case we use only the locally-predicted reward to select an action, setting G to 1. Similarly,

$$\text{if} \quad \rho_{\min} > -\eta, \quad G = 0$$

.
Here, only the world q-values are used in action selection, with $F = 1$. In the case where $F = 0$ and $G = 0$, no information about future rewards is available and an action is selected at random.
In all other situations,

$$F = 1 \quad \text{and} \quad G = \frac{(q_{\min} - \eta)}{\rho_{\min}}. \tag{3}$$

A value of $\eta = 0.1$ has been found by experiment to work well, but the algorithm does not appear sensitive to the particular value chosen, so long as it is large enough to confer reasonable immunity from noise by ignoring small value fluctuations.

After action we update the world reward table, but this time using the same combined reward calculation as is used for action selection:

$$\begin{aligned} q(s,a) \quad \leftarrow \quad & (1-\alpha)q(s,a) + \\ & \alpha\left[r + \gamma \max_{a'}\left(F'.q(s',a') + G'\rho_{a'}\right)\right] \end{aligned} \tag{4}$$

The modules' reward tables too are updated:

$$\forall m \in \mathcal{M}, Er(s_m,a_m) \leftarrow (1-\alpha_m)Er(s_m,a_m) + \alpha_m r_m \tag{5}$$

where α_m is the module's learning rate, a_m is the modular action corresponding to a, the action taken (if there is no corresponding action, m's NULL action is updated), s_m was the module's internal state *prior* to action, and r_m is the immediate local reward received.

In this way, HCML is able to base its action selection on both the world (task-based) and the learned modular reward functions.

5. Discovering dependencies

To avoid having to perform an exhaustive factorisation of the observation vector, an operation whose time complexity increases exponentially with the vector's length, we adopt a heuristic approach. There are several stages to this, each of which may yield a partial or total solution, depending on the problem. The heuristics are as follows, assuming no sensor and actuator noise:

1. Any sensor which registers a reading prior to a successful action cannot *on its own* be responsible for the failure of that action on other occasions

2. Observations collectively predicting failed actions are likely to occur a similar number of times prior to action failure.

3. The more frequently an observation occurs before action failure, the more likely it is to be a factor in predicting that failure. Observation vectors collected under action failure may be analysed for co-occurrence of these high-frequency observations to assess whether this is the case. In a search for factors, ordering observations by frequency can greatly reduce mean search times by allowing combinations of the most frequent observations to be checked first.

4. The local reward vector identifies the source of punishment: punishments received from different modules may represent different dependencies.

Together these heuristics should narrow the search for the relevant dependencies. Of course other statistical methods could be used, but where heuristics are available, but not a lot of data, such methods may not be the best choice. Where fast learning is required, heuristics are indispensable to reduce the search space.

5.1 Dependencies and Modularity

Modularity in itself constitutes a means of reducing the amount of search which has to be done to identify dependencies.

First, the modules may be assembled into a system with a logical topology which matches the physical layout of the agent. In our example we visualise the whole agent's observation vector as a ring, to reflect the layout of the modules. Each module is "plugged into" this observation vector, so that for each, its own sensor reading is in position 0, the reading of the sensor to its right is in position -1 and to its left is in position +1, and so forth. In this way the fact that for our agent there is no external privileged "front" direction or any other distinction made between modules is reflected in the internal logical layout. By representing the sensors "as they are", such logical contiguities can help to represent the world "as it is". This reduction in the arbitrariness of the representation in itself constitutes a powerful heuristic.

Second, identifying similarities and symmetries among modules may lead to a reduction in the amount of learning required by each. In the current experiments, four modules have identical dependencies. By determining that each behaves like the others, we increase the number of examples available to each; further, we can infer that all should have the same reward function, which gives immunity from spurious asymmetries introduced by noise.

Third, it is likely that different actions will have different dependencies. By assembling the data separately for each action we reduce search complexity. To the extent that there is overlap in dependencies, this may increase space complexity, but by pre-sorting the data it should reduce the dimensionality of the space to be searched. In a search for factors in a list of data, the search time is exponential in the dimensionality of the data, so it is necessary to keep this as small as possible.

Finally, a modular dependency structure in conjunction with a reward vector also reduces search effort in a high-dimensional space, by identifying factors which may be relevant in a given situation. It is likely, for instance, that rewards received from different modules may result from different dependencies (although this will not necessarily be the case, as in these experiments).

Although these heuristics may appear somewhat vague, a comprehensive analysis of dependencies in any realistically-sized agent is intractable, and anything which gives us an indication of where answers may lie must be preserved and explored. Moreover, since EMBER agents hold their learning within themselves, and learn about the world irrespective of task, this learning can be a gradual process. It is unlikely ever to be possible to learn complex dependencies quickly and easily, but any method holding out the possibility of learning them at all would be an advance over the "monolithic agent" approach, let alone the global state descriptions of classical reinforcement learning.

6. Method

We present a method for finding dependencies and symmetries in and between modules, making use of the heuristics outlined above. Although the method works, it is not so much the details which are important, as the ideas which it implements. As already noted, any available statistical methods could be used to analyse dependencies, but the use of the heuristics is important in narrowing the search for factors. In a small system like the current one we could manage perfectly well without them, but the modular approach then becomes largely fictitious, since unguided global state analysis suffers the curse of dimensionality and does not scale.

For a given action a by module M_a, a short list L of up to 5 different examples of past observation vectors O_p is maintained. Each time a fails, the prior observation is recorded in this list, together with its frequency and the module M_s which provided the punishment. A second short list P contains similar data, already processed, for use in action selection. In addition, each module has two sets of bins, one bin in each set corresponding to each element of the agent's observation vector. In these experiments, the vectors have 8 elements, representing the 8 proximity sensors.

The sets of bins hold, respectively,

1. the number of times each observation element has appeared in vectors which cause a to fail (bins F)
2. the number of times each element has appeared in vectors where a has succeeded (bins S).

When a collision is sensed in module M_s following module M_a's action, it generates a reward r and the agent's observation vector $\vec{o}$ is recorded. Module M_a deals with it in the following way:

- the observation is added to bins F
- if the '1' bits in $\vec{o}$ are a superset of those in any vector O_p in P, O_p's count is incremented, and r is added to its reward.
- if not, $\vec{o}$ is entered in list L, together with its reward r and source module M_r.

When L contains 5 items, it is processed.

First, simple dependencies are identified. If an observation o from another single module can predict success or failure, this is a simple dependency. Every acting module M_a collects the prior observation in bins S following successful actions, so an empty bin (noise effects excluded) suggests a corresponding simple dependency. Any observations in L containing this element have their rewards and frequencies summed, and are replaced by a vector representing this dependency placed in P. If no simple dependencies have been identified, or if they do not account for all the rewards received, then any remaining dependencies must be complex, *i.e.* they must arise from *combinations* of other modules' observations. The simple dependency test is crucial because although observation o may be the only one to appear in every vector which predicts punishment, it may in fact form part of several multiple dependencies whose other members do not overlap, and this would be hard to detect otherwise.

The next easiest test is for universal multiple dependencies. Any two or more observations which form a subset of all the examples remaining in L constitute such a dependency. Since we bin all the relevant observations in F, these should (subject to noise effects) contain the same number of each of the components of the dependency. If this is found to be so, then it is recorded in P, together with the reward sum and frequency for each module M_s which has registered the dependency.

If there are still some examples to be processed, two heuristics remain. First, it is likely that the same dependencies give rise to reward from the same modules;

to put it the other way around, if we have more than one observation with the same M_r, the two may be related by a common dependency. Second, because the time taken to find common factors in a set of vectors scales exponentially with the length of the vectors, which in realistic agents might be quite large, a heuristic to reduce the mean time of this process is required. This again is obtained from the observation bins: the more frequently an observation occurs, the more likely it is to be a factor. Combinations of the most frequent observations can be checked first, which tends to yield results very quickly.

As each observation is processed, it is deleted from L. Similarly, all observations prior to successful actions are checked against the list, and identical vectors deleted. In this way, large numbers of examples do not build up, and search times and noise effects are minimised. It also allows learning to be continuously updated.

6.1 Action Selection

The learned dependencies are used in action selection by comparing the current observation vector with the vectors in P. If the current vector is a superset of any of the vectors, the projected reward for that action is the average reward of the relevant entries in P. This value then becomes an $Er_m(s_m, a)$ term in eq. (1), and action selection takes place as described in section 4. above.

7. Experimental Details

Experiments were carried out on a simulated 10×10 flat gridworld with a boundary wall. The start and goal positions were at opposite corners of this grid. 20 obstacles were placed on the grid. For the static world experiments, these were always set in the same cells, but for the dynamic world, their locations were changed randomly every 5 time steps. The only constraint on their positions was that the agent should not be unable to move as a result.

The experiments compared two versions of EMBER, with and without the dependency mechanism in place. For the purposes of a benchmark comparison, Q-learning (Watkins, 1989) was used, with $Q(s, a)$ initialized to zero for all s, a. Where all rewards are less than zero, this provides sufficient exploration without the need for occasional random moves (Brafman and Tennenholtz, 2001): accordingly, all action selection was greedy. For both Q-Learning and HCML, the discount factor γ was 0.9, and the learning rate $\alpha = \frac{1}{\kappa}$ where κ is the number of times a particular action has been taken at the time of the update. Where present, sensor and actuator noise were each modelled at 3%.

Rewards: For Q-Learning, the reward was -1 for every move except the move to the goal position, when it was 0. An additional reward of -1 was given for every collision, either with the wall or with an obstacle. In HCML, the same task rewards were used; the collision reward is internally generated by the modules as described above.

7.1 Performance Tests

Ten series of 100 episodes each were run for eight scenarios. These varied according to whether the world was static or dynamic, and whether sensor or actuator noise, neither, or both, were present. The world collision model was as shown in figure 2.

After each series of 100 episodes, the world reward tables were re-initialised. The EMBER agents' modular tables were not reset - it is a central tenet of the EMBER framework that the agent is able to use experience gained from one task to perform another, and that it need not therefore start from scratch every time. However, this was not the reason for the wide discrepancy in performance.

7.2 Discrimination Tests

After the tests above, the world model was altered so that the dependencies in the four diagonal sensors were different from one another. Moving NW, the rules were the same as before; to the SW, the agent was still blocked by obstacles to W and S, but the reward always came from the W module (instead of being stochastically distributed between S and W). To the SE, movement was blocked iff an obstacle stood to the E – here, punishment always came from the NE module. Finally, a more complex situation for the NE module: movement was blocked if *either* N and SE, *or* NW and E, were occupied. In this last case, punishments for both situations were received from the same module, making it even more difficult to discriminate between the two.

8. Results

8.1 Performance Tests

The results of the performance tests are summarised in figure 7.2. These show, above, the mean steps taken per 100 episodes in ten runs of 100 episodes each, and below, on a logarithmic scale, the total number of collisions made by the agent during the same period. The categories, from left to right, alternate between the static and dynamic worlds, and noise increases from no noise on the left through sensor noise and actuator noise separately, ending on the right with both types of noise simultaneously.

Figure 4 shows the mean episode length (averaged over ten runs) for a sample scenario, the static world with sensor noise, to show the respective convergence of Q-Learning and EMBER (with dependencies) learning methods.

8.2 Discrimination Tests

Printouts of the contents of the modules at the end of sample runs of the EMBER agent are shown in figure 5. The left example shows how the modules look when the

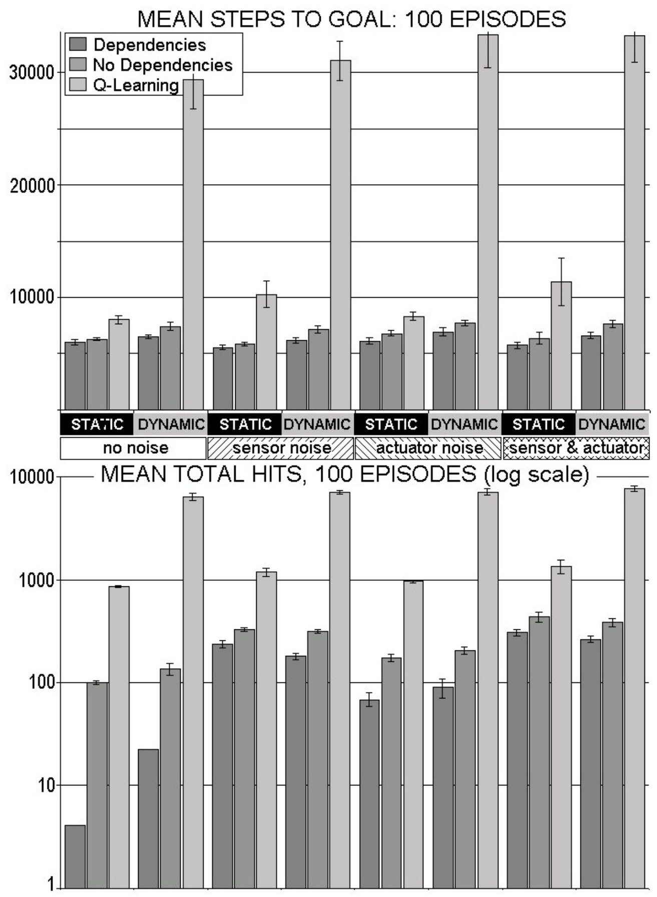

Figure 3: Results summary of the performance experiments, showing length of episode and hit statistics. Error bars are one standard deviation.

system is working correctly: the binary number shows the dependency as an observation vector, each module's observation being represented by a bit. For each module in the list, the rightmost bit (0) represents that module's own observation, and bits 1 and 7 the adjacent modules'. Other bits represent the other modules in order. The remaining numbers are the index of the module associated with this reward, followed by the reward sum and the number of instances recorded.

The similarities between the modules are clear to see, reflecting the symmetry of the world in this case. The modules N, E, S, W do not have dependency tables, although they possess the mechanism: there are no dependencies for these modules to learn.

The table in the centre is from a static world, where there are no open corners for the agent to learn from - it cannot learn satisfactorily without relevant examples, which are abundant in the dynamic world. These two example printouts are from the scenario where both sensor and actuator noise were present.

The third table shows a specimen printout obtained from a dynamic world, with sensor noise, for the different

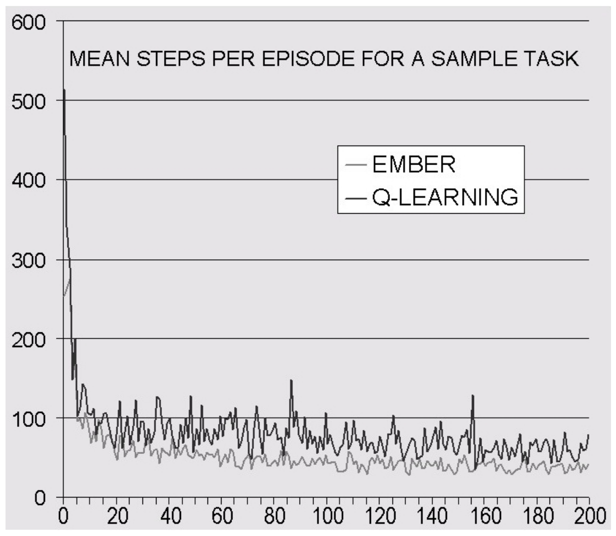

Figure 4: Mean episode lengths for a sample task to show convergence.

```
N:
NE:
10000010 1 -19 19
10000010 7 -16 16
E:
SE:
10000010 1 -28 28
10000010 7 -38 38
S:
SW:
10000010 1 -15 15
10000010 7 -10 10
W:
NW:
10000010 1 -12 12
10000010 7 -13 13
```

```
N:
NE:
10000011 1 -23 23
10000011 7 -26 26
E:
SE:
10000111 1 -46 46
10000111 7 -60 60
S:
SW:
11000011 1 -15 15
11000011 7 -22 22
10000111 7 -2 2
10000111 1 -2 2
W:
NW:
11010011 1 -3 3
11010011 7 -2 2
```

```
N:
NE:
10000100 6 -24 24
01000010 6 -14 14
E:
SE:
10000000 6 -49 49
S:
SW:
10000010 1 -19 19
W:
NW:
10000010 1 -19 19
10000010 7 -18 18
```

Figure 5: Contents of the modules at the end of sample runs of the EMBER agent: Left and Centre - original world rules, Right, asymmetric world rules

rules described in section 7.2 above. Here each module has detected the rule correctly, even the NE module, which had a difficult discrimination to make between overlapping dependencies.

9. Discussion and Conclusions

The differences between the two EMBER agents are significant. This is best shown in the hits score for the noise-free cases, where the effect is not masked by sensor noise. Without dependency detection, the agent has many more collisions when moving in diagonal directions. It therefore learns that it has a generally lower probability of success taking diagonal actions, so the frequency of these moves decreases. This in turn increases mean path lengths.

The better task-length performance of the EMBER agents in worlds where sensor noise is present is due to the increased variety of observation examples generated, which stimulates dependency detection. In realistic scenarios, where noise will be a significant factor, a method whose performance does not degrade at significant noise levels has obvious advantages. An increase in hit rate under noisy conditions is unavoidable, however, since any system must rely on sensor data to select appropriate actions. In these experiments, sensor noise is at 3%, a rate which implies that about one observation vector in every four will err in at least one element.

EMBER outperforms Q-Learning on every measure in every situation tried. The poor performance of Q-Learning, in dynamic environments in particular, is due to the extremely large state-spaces which are required to represent this simple problem, approximately 60 times the size of the EMBER representation. In the static worlds, the difference is accounted for only by the generalisation abilities of EMBER, and this is particularly reflected in the hit statistics. The combination of reinforcement learning in small state spaces with predictive generalisation and heuristically-guided logical strategies allows EMBER to learn quickly and with a minimum of examples.

In this paper we have presented an approach to the automatic determination of dependencies in a modular agent. We have shown that it is possible to learn such dependencies relatively quickly and efficiently in a small example, and we have tried to make use only of methods which would scale to more realistic scenarios. Results show that agents with this capability are able to learn and perform a simple task more quickly than by the use of conventional reinforcement learning.

More complex agents would undoubtedly require more flexible systems that that presented, and a larger concept of the structure of the body which perhaps divides it into regions connected by linkages such as limbs. It may be possible to generalise whole remote sections of the body, to reduce the representation burden. However, since an EMBER agent has only to learn once and can subsequently apply its acquired abilities in new tasks, a fairly gradual learning process would be acceptable.

These experiments present different arbitrary rules for an agent to learn. However, the ultimate aim is to have agents which can learn about themselves *in the real world*, where the rules for a given embodiment are largely fixed. Once an agent discovers how to act in the world, it will possess a physical intelligence, as a result of which many superficially different tasks may become very much easier to accomplish.

References

Brafman, R. I. and Tennenholtz, M. (2001). R-max – a general polynomial time algorithm for near-optimal reinforcement learning. In *International Joint Conference on Artificial Intelligence*, pages 953–958.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *Artificial Intelligence Research*, 13:227 – 303.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with hexq. In *Nineteenth International Conference on Machine Learning.*

Hu, J. and Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Fifteenth International Conference on Machine Learning*, pages 242 – 250.

Jacob, D., Polani, D., and Nehaniv, C. L. (2004). Improving learning for embodied agents in dynamic environments by state factorisation. In *TAROS 2004, Towards Autonomous Robotic Systems.*

Jacob, D., Polani, D., and Nehaniv, C. L. (2005a). Faster learning in embodied systems through characteristic attitudes. In *6th IEEE International Symposium on Computational Intelligence in Robotics and Automation, Helsinki, Finland.*

Jacob, D., Polani, D., and Nehaniv, C. L. (2005b). Legs that can walk: Embodiment-based modular reinforcement learning applied. In *6th IEEE International Symposium on Computational Intelligence in Robotics and Automation, Helsinki, Finland.*

Kalmár, Z., Szepesvári, C., and Lorincz, A. (1998). Module-based reinforcement learning: Experiments with a real robot. *Machine Learning*, 31:55 – 85.

McCallum, A. (1993). Overcoming incomplete perception with utile distinction memory. In *International Conference on Machine Learning*, pages 190–196.

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Eighteenth International Conference on Machine Learning*, pages 361–368.

Morimoto, J. and Doya, K. (1998). Reinforcement learning of dynamic motor sequence: Learning to stand up. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1721 – 1726.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Sixteenth International Conference on Machine Learning.*

Parr, R. and Russell, S. (1997). Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, volume 10.

Shelton, C. R. (2000). Balancing multiple sources of reward in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1082–1088.

Smart, W. D. and Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *International Conference on Robotics and Automation.*

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD thesis, Cambridge University.