Reinforcement Sailing

Philip Sterne
Department of Computer Science
Rhodes University

Gillian Hayes
Institue of Perception, Action and Behaviour
School of Informatics
University of Edinburgh

October 25, 2006

Abstract

This article examines applying reinforcement learning to sailing. We give a model of a simple sailing boat. Standard tabular reinforcement learning is shown to be ineffective in controlling this naturally continuous model. We examine a method [Smith, 2001b] which adaptively quantises both the state and action spaces and show that it has similar performance to the tabular case but requires only a fraction of the resources. Finally we examine the continuous method of wire-fitting [Baird and Klopf, 1993] combined with advantage learning [Baird, 1995].

Our findings suggest that for tasks which require smooth actions in response to slowly-changing states (such as sailing) the best results are obtained by using a continuous method. Wire-fitting was found to have a slower convergence rate, but might form a good starting point for a yacht's autopilot.

Keywords: [Reinforcement Learning, Sailing, Wire-fitting, Self-organizing Feature Maps]

1 Introduction

Sailing is a complex interaction of forces generated by moving through both wind and water. Machine learning is the automated process of trying to see patterns inherent in data. This article looks at using sailing as a test problem and applying several reinforcement learning algorithms to it and is a shortened version of [Sterne, 2004].

The sailing is modelled using a computer simulation, and it is not claimed that the final solutions found would be applicable to a real-world yacht. Moreover current sailing regulations forbid the use of automated systems in setting the sails, however there is still scope in creating an autopilot for a yacht.

In [Adriaans, 2003] a description of such an autopilot is given. This system has a hybrid architecture in which various sailing rules are provided by an expert. The system then learns the optimal parameters for these rules. Adriaans claimed that the task of sailing is a difficult one, with a vast array of sensor information making convergence of learning algorithms very slow.

This article argues that Adriaans discarded reinforcement learning prematurely. Instead of approaching the problem as a monolithic reinforcement learning problem, it should have been subdivided into behaviours corresponding to Adriaans' agents, several for each different time-scale. Reinforcement learning could then be applied to those behaviours which would benefit from such an approach[1]. It would have then been possible to learn each problem in a reasonable amount of time. Moreover a computer simulation could provide a good first approximation of the policy to be learnt.

This article aims to explore one such layer in the architecture - the basic sailing layer. In this layer, we avoid complications such as obstacle avoidance, and obtain a starting point for adding other layers onto this architecture.

However sailing is a task which naturally requires continuous actions, and reinforcement learning for continuous states and actions only has weak guarantees for convergence. Moreover the rate of convergence is normally much slower than tabular reinforcement learning. This article explores the trade-offs in several methods while gradually increasing the complexity of the solution.

1.1 Structure of this article

The following section provides some of the related work in the field of continuous state and action reinforcement learning, as well as examining Adriaans' system in slightly more detail. Section 3 provides a brief introduction to sailing and presents a model of a sailing boat, along with a hand-coded strategy. Section 4 examines the performance of the standard discrete methods of reinforcement learning when applied to the model and discusses their limitations. In section 5 we present a method of adaptively quantising the state and action space using two separate self-organising feature maps. In section 6 the final method of using a neural network to predict the reinforcement received. Finally in section 7 we provide a conclusion and discuss the findings of this research.

[1] We are not claiming that all such behaviours are appropriate candidates for reinforcement learning, for example the rules for determining right of way would be better encoded in a symbolic form, this would give one more confidence that all possible variations would be correctly obeyed.

2 Related Work

In this section we review the relevant work done on the RoboSail project which looks at applying different machine learning techniques to sailing. This article assumes the reader is familiar with Reinforcement Learning, if not [Sutton and Barto, 1998] or [Kaelbling et al., 1996] are both excellent introductions to the subject.

2.1 The RoboSail project

Machine learning techniques have been applied in the domain of sailing already. In [Adriaans, 2003] the application of reinforcement learning to sailing was examined, however they reported very slow training times in [van Aartrijk et al., 2002]. Adriaans also argued that different aspects of sailing required different time scales of evaluation. As an example: while steering the boat requires sub-second reaction times, other tasks such as navigation only need to be evaluated hourly. This discouraged the authors and they instead pursued a hybrid approach.

In their approach, basic knowledge of sailing was used to form rule sets, each operating on a different time-scale. The rules were of the form:

> If the apparent wind angle[2] is between x and y then the sail should be set at z.

where x, y and z were defined by fuzzy rules. The initial estimates for x, y and z were supplied by expert knowledge and then fine-tuned through experience which took the form of a large database containing actual sailing episodes.

[Adriaans, 2003] split the task into four main agents, each operating on a different time-scale. They are as follows:

- *Skipper* - This agent is responsible for tactical decisions, it examines weather maps and tidal information to establish a goal way point. (This process is only partly automated, and requires input from the human skipper as well.) The *Skipper* makes decisions roughly every 3-6 hours.
- *Navigator* - This agent takes the goal way point set by the *Skipper* and the boat's current position, it then decides on a compass heading that is to be followed to best obtain this way point. Decisions at this level are made every 15-30 minutes.
- *Watchman* - The *Watchman* uses the compass heading it is given as well as the current heading to determine a rudder target. This decision also takes into account other factors such as the current wind speed, and sailtrim[3]. At this level a decision is made every second.
- *Helmsman* - The *Helmsman* accepts the rudder target and the current speed of the boat. It outputs the force to be exerted by the motors. To achieve smooth control this is run ten times a second.

This article explores using reinforcement learning in what would be the *Watchman* layer. We use only a simple state description under the assumption

[2] The apparent wind is the difference between the true wind and the boat's velocity and is the wind experienced on the moving boat.

[3] The sailtrim refers to how the sails are set

that other layers could be added in a subsumption-style manner [Mahadevan and Connell, 1992] to modify the original behaviour when necessary. In this manner it is possible to use reinforcement-learning in an incremental fashion.

3 The Sailing model

In this section we briefly introduce the basics of sailing. We go on to describe in detail the model of a sailing boat which is used in the reinforcement learning algorithms. We describe the state representation and the performance of a hand-coded controller.

3.1 Introduction to sailing

For our model only two parts of a boat are relevant: the sail, and the keel (see Figure 3.1). Both the sail and the keel can be viewed as wings in different media. Due to wind (or tides) these two media are moving relative to each other and this creates flow over both of the wings. If controlled correctly this enables a boat to be maneuverable and travel in a given direction, as long as the direction is not directly into the wind. For a longer description, as well as details of another sailing model please see [Sterne, 2004].

Figure 1: A symbolic view of a boat from above.

Sailing is a non-holonomic control problem, as we are unable to turn on the spot. This makes it a difficult problem to learn how to approach a target. In this article we ignore the problem of approaching a target and instead concentrate on sailing in a given direction.[4]

3.1.1 Continuous states and actions?

As a control problem, sailing warrants continuous states and actions. There are several reasons for this:

- In an ideal solution the rudder is to be used as little as possible; every time the rudder is used, it acts as a brake, slowing the boat down.
- The lift generated by the sail is very sensitive to the angle of attack. Changes of less than 5° can result in a significant loss of lift (see figure 3(a)), and the boat will sail a lot slower. If the variables were to be discrete and have sufficient resolution the space requirements would be huge.

While both the rudder and sail require fairly sensitive control, we can still hope to learn the correct actions in a given context. This is because the performance varies smoothly; as we approach the optimal action the reward increases in a fairly linear manner. However we will still have to deal with the problem of delayed reward; for example in both of our models if we are currently sailing slowly away from the target then the best action is to build up speed so that the rudder becomes effective, and we are then able to turn and sail directly towards the target. A short term solution would rather use the rudder as much as possible to slow down the boat, so that we sail away from the target as slowly as possible.

[4] It is later shown, if the controller is sufficiently good at sailing in a target direction, then it is able to sail a course, which involves approaching several targets.

3.2 Timin's Model

The model was designed by Mitchell Timin for use in research into evolving a neural network controller[Timin, 2006]. Originally the sailing model had to learn to sail around a circular island, however we modified the task so that it had to sail in a given direction. The AnnEvolve group was able to evolve a controller which sailed around the island in 90% of the trials. The original model also had a shifting wind, which made the task more complex. We chose to use a steady wind model, since we can rotate the boat and target direction until the wind is aligned to north.

Figure 2: The AnnEvolve Model. α_s and α_k stand for the angle of attack in the sail and rudder respectively.

As shown in Figure 2 the heading of the boat is derived from the direction of the velocity of the boat (i.e. the boat's drection is found by adding the velocity vector to the keel angle). The boat changes its heading through the use of of the sails and keel, which both generate forces parallel and perpendicular to the path of the boat. The forces change the velocity, which in turn changes the heading of the boat. An unrealistic side-effect of this is that the boat is easily maneuvered at low speeds (since the direction of the velocity can change quite drastically if there is only a small amount of momentum). The state and action variables are as follows (with the ranges given in brackets) :

State		Action	
Speed	$[0, 0.5]$	Rudder	$[-\frac{\pi}{4}, \frac{\pi}{4}]$
Relative wind	$[-\pi, \pi]$	Sail	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
Directional error	$[-\pi, \pi]$		

The Speed is the euclidean norm of the velocity. The Relative wind is the difference in angles between the target direction and the relative wind measured in radians. The directional error is the difference between the heading and the target direction. The state representation is realistic in the sense that this information would be readily available on a sailing boat.

The Rudder is measured as the angle of attack (i.e. an angle relative to the moving water). The sail's angle of attack is specified with respect to the apparent wind, however with a larger range.

The angles of attack are converted into forces parallel (drag) and perpendicular (lift) to the wind and water (see figure 3(a). The drag and lift coefficients are calculated using the polar diagram shown in 3(b) and are multiplied by the sail area (or rudder area). The force is then:

$$F = c \times m \times V^2 \times A$$

where:

- c is the coefficient (lift or drag)
- m is a coefficient representing the density of the medium through which the sail/rudder travels. For air this is 1.226 kg/m^3, water is 1000 kg/m^3.
- V is the velocity.

(a) As the wind is diverted by the sail, lift and drag are generated.

(b) The coefficients of lift and drag as dependent on the angle of attack (α)

Figure 3: Calculating the resultant forces.

- A represents the area of the sail or rudder.

(from [Marchaj, 1964] page 70).

To transform the angles of attack into actual forces estimates of the boat's measurements were needed. We modified Timin's original figures to give a model which was slightly more maneuvrable (by lowering the mass). The measurements were as follows:

Measure	Measurement
Sail Area	100m^2
Rudder Area	6m^2
Boat Mass	$10,000\text{kg}$

Initially we had hoped to be able to learn the tacking behaviour[5] required for sailing into the wind, by simply specifying a direction which was too close to the wind. Unfortunately none of the learning methods surveyed could handle this problem. All would be able to head somewhat into the wind, however the drag of the wind would continually slow them down, until their velocity was away from the target. Since heading is dependent on velocity, the boats would end up sailing away from the target. In some cases they might recover, and head towards the wind again however they would be losing ground, rather than gaining ground. To overcome this we ensured that the required headings would never be pointing less than 45° into the wind.

[5]When the target is almost directly into the wind then a boat cannot sail to the target directly. Instead it sails roughly 45° to one side of the wind and then turns (known as 'tacking') to sail on the other side of the wind.

3.3 Mathematical updates of the dynamics and reward signal

The update is based on a second order Adams-Bashforth update (see page 295 in [Burden and Faires, 1997]). If trying to find a solution to the problem:

$$\dot{v}(x) = f(x)$$

An approximate solution can be given in the form:

$$v(x_t) = v(x_{t-\Delta t}) + \Delta t(1.5f(x_t) - 0.5f(x_{t-\Delta t}))$$

Since force is proportional to acceleration we can use this formula to calculate the velocity. We can also use this formula to calculate the updated position from the velocity. The update equations are as follows:

$$\begin{aligned} x_{i+1} &= x_i + (1.5\dot{x}_i - 0.5\dot{x}_{i-1})\Delta t \\ \dot{x}_{i+1} &= \dot{x}_i + (1.5\ddot{x}_i - 0.5\ddot{x}_{i-1})\Delta t \end{aligned}$$

with x_i, $\dot{x}_i$ and $\ddot{x}_i$ representing the position, velocity and acceleration respectively. To calculate the acceleration the force vectors are simply summed ($\ddot{x} = \frac{1}{m}\sum_j f_j$).

At first glance the simplest and most intuitive reward function would be to give a reward proportional to the distance covered in the target direction. However in the above formulas the position is updated using the old velocities. Thus the reward received would be the same for all actions in a given state and only the next state would receive different rewards. To avoid this we chose to use the directional velocity as a reward signal which still gives us a useful reward signal, with the advantage of having immediate feedback.

3.3.1 Hand-coding a controller

In order to provide a baseline performance we hand-coded a controller. This proved quite time-consuming as changing the policy often had unintended consequences. This was in part due to the simplifications the model made which made the task less intuitively like sailing. Nevertheless we obtained a reasonably simple controller which had good performance.

The actions of the controller are calculated as:

$$\begin{aligned} \alpha_s &\Leftarrow -0.4 \times \texttt{RW} \\ \alpha_r &\Leftarrow \begin{cases} 0 & \text{if } \|\texttt{DE}\| < \frac{pi}{9} \\ \text{sign}(\texttt{DE})\frac{\texttt{DE}^2}{2.3} & \text{otherwise.} \end{cases} \end{aligned}$$

where DE stands for Directional error, and RW stands for the relative wind. By using the rudder only when the directional error was large we were able to sail quite fast, and still steer the boat using the sail. While this controller is not optimal (it does not use speed information) it appears to be close to optimal.

3.3.2 Evaluating performance

To test the performance we sailed a simple triangular course with it, see figure 4(a). In this course sailing from buoy 1 to buoy 2 is directly into the wind, which requires tacking. While our controller is capable of doing this, it does not do it particularly well if one notices how far away from the wind the boat has to sail. Also in moving from buoy 2 to 3 the controller does not sail a straight line.

However the boat is able to maintain a good speed, while sailing the course. This is shown in figure 4(b). To obtain this graph we randomly started the boat in 100 random positions far away from a target, the reward for 100 simulation steps was recorded. The average distance covered towards the target direction in each simulation step is plotted.

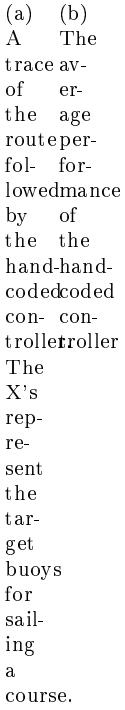

Figure 4: The hand-coded performance for Timin's model.

Figure 5: Three different types of eligibility traces, from left to right: a spatial trace, a temporal trace and a spatio-temporal trace. Eligibility is represented through shading; darker states are more eligible (after [Thompson, 2002])

4 Discrete Methods

In this section the normal reinforcement learning method of dividing up the state space is examined. We also test the spatio-temporal eligibility trace. It is shown that these naïve approaches result in an extremely slow learning rate and are rather wasteful. The trace does improve the performance noticeably but is still a poor method for this task.

4.1 Spatio-temporal traces

Here we examine the use of a spatio-temporal eligibility trace [Thompson, 2002]. The standard temporal trace [Singh and Sutton, 1996] awards eligibility to states which have recently been experienced, while a spatial trace awards eligibility to states that are similar to the experienced state. A spatio-temporal trace simply awards eligibility to states similar to those recently experienced (see figure 5). In this way we can hope to gather a broad overview of the task, early on, without having to experience every state. As the trial progresses however we will want to narrow the spatial aspect of the trace to enable a higher resolution of the task.

To implement the spatio-temporal trace we included the idea of a neighbourhood in the update equation of the standard eligibility trace to:

$$elig(i,j) \Leftarrow \max(neighbour(i,j), \lambda\gamma elig(i,j)) \tag{1}$$

We implemented the eligibility trace in this manner as we felt this was faithful to the original theory of replacing traces [Sutton and Barto, 1998]. Our method has the advantage of degenerating into a standard replacing trace as the spatial eligibilty shrinks. Our *neighbour* function was a Gaussian curve centred on the closest discrete state (with a maximum value of 1).

4.2 The Naïve Method

Due to the additional complexity involved in a continuous reinforcement learning framework when researchers are faced with a naturally continuous problem many simply convert it to a discrete problem. However the performance can be severely hampered as a result of many things:

- *Too coarse a discretisation.* If the continuous variables are sectioned too coarsely then it is possible that many conceptually distinct states which require separate actions are now mapped to the same discrete state. This will result in poor learning as there will be a large variation in the rewards received for each action.
- *Too fine a quantisation.* However if the variables are sectioned too finely then there is a problem with the 'Curse of Dimensionality'. Moreover it is highly unlikely that each state will be experienced a sufficient number of times as the size of the state space explodes exponentially.

- *The wrong offset.* Even if the discretisation has partitioned the state space into correctly sized units there is no guarantee that these states will be nicely aligned with significant changes in the reward function. If the boundary condition of conceptually different states lie in the middle of a discrete state then that state would receive conflicting reinforcement.

While we could use expert knowledge to quantize the states in an intelligent manner to help lessen these problems, we chose to first examine *tabula rasa* learning's performance. As a further research direction it would be interesting to see how much better the methods surveyed here would perform with the incorporation of expert knowledge.

4.2.1 Experimental Setup

To learn the Annevolve task the state-action space was sectioned as follows:

- 12 states for the relative wind equally spaced in $[-\pi, \pi]$.
- 12 states to encode the directional error equally spaced in $[-\pi, \pi]$.
- 5 states for the speed ($[0.05, 0.5]$).
- 5 states for the rudder action ($[-\frac{\pi}{4}, \frac{\pi}{4}]$).
- 5 states for the sail action in ($[-\frac{\pi}{2}, \frac{\pi}{2}]$).

This resulted in a total of 18000 state-action pairs (experiments were carried out with finer quantization, but the performance did not improve significantly). Standard temporal difference learning (TD(λ)) was implemented with the parameters as follows:

Parameter	Value
λ	0.9
γ	0.9
ϵ	$0.9 \times 0.9995^t + 0.05$
α	$0.5 \times 0.9995^t + 0.3$

where ϵ represented the probability that an exploratory action would be taken, and α represented the weighting that was given to new experience. These parameters have been shown to be fairly robust in terms of performance so we did not attempt to optimise them extensively. The decaying formulas used for α and ϵ were found to be useful in transitioning gracefully from an initial value to a final value.

4.2.2 Results

We tested the standard tabular method as well as the method involving spatio-temporal traces on Timin's model. We first evaluated the learning speed for both methods. This was done by training each method for a total of 30000 steps but evaluating the performance after every 1000.[6] (The performance evaluation was

[6] This experience was broken up into shorter episodes lasting 50 simulation steps, which ensured that most of the state space was experienced. Thus performance was evaluated after every 20 episodes.

the discounted reward from 50 episodes; each 50 steps long with no exploratory moves.) This was repeated 15 times and the results are plotted in figure 6(a). From this we can see that the spatio-temporal trace has increased the learning rate greatly; the performance plateaus after roughly 15000 steps. [Smith, 2001a] reports that neighbourhood learning increased the rate of convergence by a factor of six, from this diagram it seems that spatio-temporal learning has an even faster rate; the performance after 1000 steps is roughly equivalent to the final performance of the standard method after 30000. However if the experiment is run for a sufficiently long time then the standard method eventually increases its performance to the same level.

The final performance was also examined in more detail. When evaluating the performance of both methods exploratory moves were turned off and the simulation was run for 100 steps. This was repeated 100 times and the average reinforcement was recorded. The results are plotted in figure 6(b). From this we can see that using a spatio-temporal trace performs best with an average reward consistently greater than 0.2.

Qualitatively the controller was able to sail reasonably efficiently away from the wind. The controller was able to marginally sail into the wind, however was unable to tack when the course changed direction. This would result in the boat drifting slowly away from the target.

However we may improve things by only focusing on the regions which are likely to occur during normal sailing. This leads to adaptive quantisation.

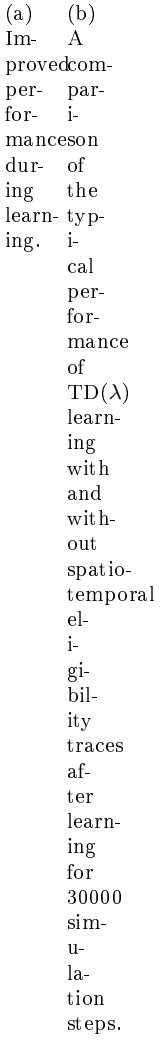

Figure 6: Discrete performance

5 Adaptive discrete reinforcment learning

In the previous section static quantisation of the state-action space was explored. In this section we present a method of dynamic quantisation taken from [Smith, 2001b]. We also explore whether a change of state representation affects Smith's method.

5.1 Self-organizing feature maps

Self-organizing feature maps (SOFM's) are an unsupervised form of learning[Kohonen, 1997]. They were developed by Teuvo Kohonen. They have been used in many areas (A recent bibliography of the applications of Kohonen nets lists 5000 articles [Kaski et al., 1998] [Oja et al., 2003]).

SOFM's are used to discover properties of an input distribution. For this paper we will use two SOFM's: one to discover the frequently visited states in the normal course of sailing, and another to find useful actions.

A SOFM consists of several nodes joined together in a specified topology. When an input is presented to the SOFM the winning node is identified. This node is then moved closer to the input, all of the winning nodes neighbours are also moved (but not quite as much). This simple algorithm allows a simple representation (the specified topology) of a possibly complex input distribution. This property of SOFM's is exploited in the following section on Smith's method.

5.2 Smith's method

In [Smith, 2001b] the tasks attempted needed only a one-step return, such as controlling a Khepara robot and shows that this method is able to learn an adequate controller for such a task. The sailing task is considerably more complex, and represents a good test of the scaling properties of the algorithm. However since this task is more complex the spatio-temporal trace from section 4 is used. In this way neighbouring state-action pairs can share the reward.

The algorithm uses two Kohonen Self Organising Feature Maps (SOFM's). The first SOFM is used to map actions and the second is inputs. A Q-table is maintained as a link between states and actions. The algorithm is as follows (taken from [Smith, 2001a]):

1. Initialise the *input map* and *action map* to small random values.
2. Present the *input map* with the current state vector and identify the winning unit in the *input map* (s_j).
3. Identify a unit in the *action map* as follows:

$$action = \begin{cases} \arg\max_a(Q(s_j, a)) & \text{with } \Pr(1-\epsilon) \\ \text{Random Action} & \text{with } \Pr(\epsilon) \end{cases}$$

4. The action chosen is calculated by adding some exploratory noise to the weights of the winning node of the *action map*.
5. Receive reinforcement r and next state s' from the environment.

6. If $r + \gamma \max_i Q(s'_j, a_i) > Q(s_j, a_j)$, then the perturbed action appears to be an improvement over the proposed action, so update the *action map* towards the perturbed action.

7. Update *all* Q-values towards the corrected return proportionallly to the Q-learning rate and the product of the two neighbourhoods (of the two SOFM's) :

$$elig(s,a) \leftarrow \max(N_{state}(s)N_{action}(a), \gamma\lambda elig(s,a)) \tag{2}$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha(t)elig(s,a)\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

8. Update the *input map* towards the state just seen according to the usual SOFM update rule.

9. Goto instruction 2

5.2.1 Adaptive Discretisation

Smith's method is attractive for several reasons:

- *Adaptivity* The most important reason is that the discretisation is adaptive in both the state and the action spaces. This is important in learning to control a sailing model as it is not clear which states will be experienced, in using the model. It also avoids many of the difficulties discussed in the beginning of this section.

- *Simplicity* The second reason for choosing such a method is for its conceptual simplicity. It is easy to understand and implement.

- *Distribution sensitive* The SOFM honours the input distribution, this allows a higher resolution in state-space regions which are experienced more often. While there is no guarantee that this is where the resolution is needed it does seem reasonable to apportion resources this way.

Comparing the update equation 2 (on page 15) with equation 1 (on page 10)one notices how the product of the neighbourhood functions of the state SOFM and the action SOFM, forms the spatio-temporal trace. In [Smith, 2001a] this is called neighbourhood Q-learning, however the temporal aspect of the trace is not included in that article.

5.3 Representation of the state

An advantage of using self organising maps is their dimensionality reduction. They automatically map onto any manifold present in the state space, which allows us to use a better representation of the state space. Rather than using the angles directly we are now able to take the sine and cosine of the angles. Using this representation angles slightly less than π appear close to angles slightly greater than $-\pi$, if one uses the plain angle then these angles appear far apart. While the size of the the state representation has increased, there is no change in the intrinsic dimensionality. Respecting any manifold present in the state space could speed up learning significantly.

Figure 7: A comparison of different state representations on the learning performance (Timin's model)

Unfortunately Smith's method requires the setting of many parameters. The interested reader is advised to consult [Sterne, 2004] for a full description of the parameters (fortunately the performace of Smith's algorithm appears robust for most of the parameters).

In order to ensure the SOFM's covered the entire state space they were trained in short episodes (50 steps long) and we randomly reinitialised the model afterwards. This ensured that large parts of the SOFM did not converge to cover a single episode, which would occur if many similar states were shown to the SOFM. The experiment was repeated 15 times and the averaged results are shown in figure 7.

It is impressive that the method has learnt reasonable control of Timin's model with one fifth of the number of states as in the discrete case. We did experiment with using larger state space maps, however the performance did not improve significantly and the training times were noticeably longer.

From figure 7 we can see that there is no increase in the performance when using the sine and cosine of the angles rather than the angles themselves. This is due to the fact that the best place to introduce a discontinuity would be between π and $-\pi$ since this is the angle at which the best direction to turn changes. By breaking the angle in the best place there is nothing to be gained from using a sine and cosine representation, but it is reassuring that there is nothing to be lost either.

Qualitatively the performance is identical for both representations. The boat has learnt to sail downwind very well, however it battles to sail even slightly above the wind. The controller's actions are somewhat jerky and this definitely reduces its effectiveness (since it slows down the boat considerably). While there are ways to interpolate outputs in a SOFM, [Aupetit et al., 2000], it is not clear how to apply reinforcement learning to such interpolated results. It is quite easy to think of scenarios where such interpolation could lead to disastrous results. For example if one is trying to learn obstacle avoidance in a mobile robot then averaging a left turn and a right turn could result in a collision. This suggests a naturally continuous method might be best.

6 Continuous-state Reinforcement learning

In this section we present the final method examined for sailing control. Wire-fitting [Baird and Klopf, 1993], uses a function approximation scheme and an interpolation function to approximate the value function.

One of the main problems in extending Reinforcement Learning theory to continuous states is the problem of divergence. For problems with continuous states some sort of generalization is required. However many of the theoretical guarantees rely strongly on each state being experienced many times, not merely similar states. Thus many of the guarantees on convergence fall away, and in some cases divergence has been observed.

Divergence in some cases hasn't stopped researchers from exploring these methods. Many researchers have tried using a function approximator to estimate the value of the current state. The best known success is Tesauro's Backgammon player (TD-Gammon) which has learnt to play at grandmaster level. [7] This was achieved through repeated self-play with the information used to update a three layer back-propagation network. See [Tesauro, 1995] for more details.

However in [Boyan and Moore, 1995] several straightforward examples are shown which reliably diverge using a variety of different function approximation schemes. The simplest case of proven divergence is Baird's counterexample [Sutton and Barto, 1998]. In this case the estimated value can diverge, even though the linear function could represent the true value exactly. The lack of convergence for this simple case, is worrying and there are other cases where divergence occurs [Baird, 1999].

The problem of divergence also partly occurs as a result of bootstrapping. This refers to the process of learning a new estimate of a state based on the estimates of other states. In [Thrun and Schwartz, 1993] reasons are given why bootstrapping can lead to divergence. In this paper they consider a reinforcement learning problem where reward is given only at the end of an episodic task. As a result the values of the initial states don't differ by much, as the decayed future reward is small. Taking a non-optimal move will result in only a small penalty. If the function approximation scheme introduces random, unbiased noise, then under certain conditions we can expect Q-learning to fail.

This is as result of the max operator which introduces bias into the unbiased noise. However this biased noise can easily dominate the useful information. In this case the network is expected to fail, and will not improve beyond a random controller.

6.1 Advantage updating

The problem of overestimation increases when the values of the possible actions don't differ by much. In this case even discrete Q-learning suffers from long training times. In [Baird, 1993] Baird argues that as a control problem approaches continuous time (i.e. the time between action selection decreases) then the cost of choosing a suboptimal action approaches zero. This is because the change in states becomes vanishingly small for any given action (in continuous time). As it does so the time required to train a discrete controller

[7] While Backgammon is not a problem involving continuous states or actions the size of the state-action space is prohibitively large, thus requiring the generalisation needed for continuous reinforcement learning.

increases exponentially. In [Baird, 1993] a solution is proposed where one learns 'advantages', which in essence represents the derivative.

Mathematically, as

$$\lim_{\Delta t \to 0} Q(s,a) \to \max_{a'} Q(s,a')$$

which implies that $\forall_{a,a' \in A} Q(s,a) - Q(s,a') \to 0$, differentiating between the optimal and worst possible action becomes impossible. However if we define an advantage as:

$$A(s,a) = \lim_{\Delta t \to 0} \frac{Q(s,a) - \max_{a'} Q(s,a')}{\Delta t} \tag{3}$$

where Δt is the time step then it does not converge to zero for all actions in the state. Advantage updating has been shown to have constant convergence time in simulations where the time step approaches zero [Baird, 1993]. Moreover since the advantage function does not approach zero for all actions there is more chance of representing it reasonably accurately with a function approximator (although divergence is still possible).

However one consequence of equation 3 is that the maximum advantage in any state is zero. This makes it hard to determine which states are more desirable than others. An estimate of the next state's value is also required to update the advantage estimate. Originally Baird proposed to learn the value function as well as the advantage function. However it is possible to modify the rule so that the maximum value is unchanged. We discuss this form of advantage learning in more depth in section 6.4.

6.2 Residual Methods

Residual methods are promising as they are only slightly different formulation of the problem, and they have convergence proofs. We briefly sketch a derivation of the formula below, by considering the Q-learning update (the interested reader should consult [Baird, 1999] and [Baird, 1995] for a more in-depth derivation).

Assume we have a parameterisation of the Q-values as $Q(x,u,w)$ where w is a vector of weights. A direct method attempts to minimise the temporal difference error directly:

$$\Delta w = \alpha \left(R + \gamma \max_{u'} Q(x',u') - Q(x,u) \right) \frac{\partial Q(x,u)}{\partial w} \tag{4}$$

however this can lead to instability and divergence as mentioned earlier. Rather we try to minimise the mean squared Bellman residual which is defined as:

$$E = \frac{1}{n} \Sigma_x \left[\langle R + \gamma \max_{u'} Q(x',u') \rangle - Q(x,u) \right]^2$$

This gives rise to the residual update:

$$\Delta w = \alpha \left[R + \gamma \max_{u'} Q(x',u') - Q(x,u) \right] \left[\frac{\partial}{\partial w} \gamma \max_{u'} Q(x',u') - \frac{\partial}{\partial w} Q(x,u) \right] \tag{5}$$

For this update convergence can be guaranteed to a local minimum of the Bellman residual. However the convergence can be very slow, to overcome this Baird

Figure 8: The wire-fitting architecture (after [Baird and Klopf, 1993]).

proposes a weighted average of 4 and 5. If one is careful with the weighting factor then one can guarantee that the update is never away from the direction suggested by eqation 5. Thus the guarantee of convergence still holds.

This learning rule can be applied to other forms of learning such as advantage learning. In the next section we use a neural network trained on targets which have been modified using a form of the above update, but modified to work on advantages rather than Q-learning.

6.3 Wire-fitting

Wire-fitting[Baird and Klopf, 1993] has been succesfuly used on a similiar task : controlling an autonomous underwater vehicle [Gaskett et al., 1999b]. Wire-fitting requires a function approximator (such as a neural network) to estimate several control actions(u_i's) as well as control values(y_i's). A control action is a potential action for a given state, in this article the actions consist of setting the sail and setting the rudder. Each control value is an estimate of the reward received for taking the corresponding control action. The neural network maps the given state into estimated control values and control actions. These control actions and control values are fed into the interpolation function:

$$f(u) = \lim_{\epsilon \to 0} \frac{\sum_i y_i \times [\|u - u_i\| + c(\max_k y_k - y_i) + \epsilon]^{-1}}{\sum_i [\|u - u_i\| + c(\max_k y_k - y_i) + \epsilon]^{-1}} \tag{6}$$

by using this interpolation function we are able to estimate the reward received from actions other than the control actions. Equation 6 has some useful properties for reinforcement learning:

- *No wild predictions.* If the point encountered is far away from known points then the expected value is simply the average of the control values. This prevents any unrealistic extrapolation.
- *Analytical Maximum.* Since any point away from the control actions is a weighted average of the control values, the maximum of the function must occur at one of the actions. Moreover the form of the interpolation guarantees that that maximum will pass through the maximum control value. This means that finding the maximum of the function does not require any evaluations of the function, one can simply use the maximum control value.
- *Differentiability.* This function is differentiable, which allows the gradient of the error to be calculated and passed back to the function approximation scheme. This gradient can then be used to calculate new targets to train the function approximation scheme on. Since the gradient of equation 6 has a complicated form it is not presented here. It is not technically difficult to derive however and the interested reader is advised to read [Gaskett et al., 1999a] for more details.

Since the control values and actions are the output of a function approximator they are smooth functions of the state. Since the policy is simply the action

with the maximum control value we can see that the policy action is also a smooth function of the state. However taking the maximum introduces discontinuities into the policy action at decision boundaries where the action with the maximum value changes. This is a good class of policies, as the crisp decision boundaries avoid problems with averaging intermediate values, for example in a collision avoidance task either a left turn or a right turn might be acceptable but averaging them results in moving forwards which could lead to a collision.

A potential disadvantage is that the formula relies on the norm. For high dimensional spaces the norm behaves non-intuitively [Bishop, 1995] [8]. This would have the result that the interpolated function would be near the mean for most input values, rendering the interpolated function useless for very high-dimensional surfaces.

Added flexibility comes from the fact that architecture does not place any restrictions on the type of function approximation scheme. For this task we used a back-propagation neural network [Bishop, 1995], but other types are possible such as lazy learning or radial basis functions.

An outline of the algorithm we followed:

1. Initialise the neural network weights to small random values.
2. Calculate control actions and control values by presenting the current state to the neural network.
3. Get the action

$$action = \begin{cases} u_i \text{ where } i = \arg\max_j(y_j) & \text{with } \Pr(1-\epsilon) \\ \text{Random Action} & \text{with } \Pr(\epsilon) \end{cases}$$

4. Add some exploratory noise to the action.
5. Take action and observe reward r and next state s'.
6. Calculate the reward predicted by equation 6.
7. Calculate the gradient of the error and use this to update the network weights.
8. Train the network on the updated control actions and values.
9. Goto step 2.

We also found it necessary to bound the targets for the control actions and action or else they would sometimes diverge. This is in part due to the gradient calculation updating all the control actions when it really only needs to update the nearest few. As an example if the sampled action has a lower reward than predicted it will push all the control actions which are above it (i.e. have a greater control value) away and lower them slightly. If they are already at the bounds of the action space it makes sense to restrict them, since they will not contribute to the surface otherwise.

[8] Bishop gives an exercise showing that the proportion of volume that a d-dimensional sphere contained in a d-dimensional cube tends to zero as d increases. As a control action only has a spherical region of influence (from the norm) this shows that it too suffers from problems of dimensionality although to a far lesser extent than most other methods.

6.4 Advantage learning

Since wire-fitting is designed for continuous state and action methods, it is natural to include advantage updating which can speed the convergence of near-continuous time learning by orders of magnitude [Baird, 1993]. If a problem requires continuous states and actions it is normally because fine-grained control is required. Advantage updating emphasizes the differences between states by a factor proportional to $1/\Delta t$ (see page 17 for more details).

In the original paper on advantage updating, it was proposed that one learn both a value function and an advantage function which is then normalised so that the maximum value for the advantage is zero in all states. However it is possible to modify the formula so that one does not need to model both the value and advantage functions [Baird, 1995]:

$$A(x,u) \leftarrow (1-\alpha)A(x,u)+$$

$$\alpha\left[\frac{1}{\Delta t}(R+\gamma^{\Delta t}\max_{u_{t+1}}A(x_{t+1},u_{t+1}))+(1-\frac{1}{\Delta t})\max_{u_t}A(x_t,u_t)\right] \qquad (7)$$

This is the advantage learning update (as opposed to advantage updating).

6.5 Experimental Setup

Since we are using a neural network for learning, we must be careful of forgetting previous knowledge. This interference can occur if we only focus on updating the network for the most recent experience. If we perform an update which is greedy with respect to the new experience we can change the network in other parts of the state-space for the worse. There are many ways to avoid interference; we take the simplest approach of performing training updates in batches. Other methods do exist, another method followed in [Gaskett et al., 1999a] is to maintain a buffer of experience which is constantly updated and retrained.

We found that increasing the size of the batches lead to increased performance. This is probably as a result of minimising the possible interference. However we found that initially only small batches were necessary to increase the performance. This led us to starting with a small batch size and incrementally increasing it after every batch update. We also found that the advantage learning update had slow convergence which meant we needed more than double the previous number of iterations of any other method to achieve maximum performance.

The parameters for both tasks are given:

Parameter	AnnEvolve
Initial Batch size	400
Increment	100
Number of updates	30
Number of control wires	25
Number of hidden units	12

The neural network had a single hidden layer of `tansig` units, with a linear output layer. We wanted to confirm the results of [Gaskett et al., 1999a] which showed a noticeable difference in performance between the standard temporal difference update and the update proposed by advantage learning. To test this we ran 15 trials of each update. The results are shown in figure 9(a). It is

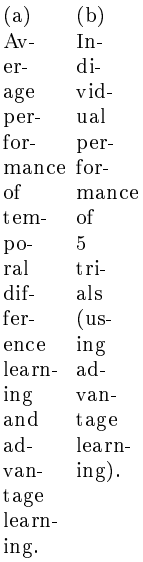

Figure 9: Wire-fitting for Timin's sailing model.

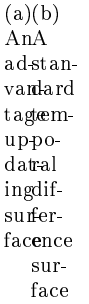

Figure 10: A comparison of the wire-fitting surfaces (note the different scales in the Z-axis)

interesting to note the differences in the surfaces predicted by both methods. We plot two typical surfaces in figure 10; note the increased scale in the z-axis for the advantage updating method, this increased scale gives the neural network an easier surface to fit, which accounts for the improved performance.

Advantage learning performs particularly well on learning Timin's model. In fact it found several solutions which were better than the hand-coded controller (see figure 4(a)). To quantify the difference in speed we timed a good solution found by wire-fitting with the hand-coded controller and found that the hand-coded controller required 2137 simulation steps to complete a fixed course, while the wire-fitting solution took only 1578. The means that the learned model is approximately 36% faster, yet on average receives less reward. This is due to the fact that hand-coded controller cannot sail close to the wind and has to cover much more distance as a result. This extra distance allows the learnt controller to win when sailing a course.

Figure 11: Two examples of solutions for Timin's model found using wire-fitting. (Compare these with figure 4(a))

We do find it rather strange that in none of the trials did wire-fitting find a strategy similar to the hand-coded's, since this seems to result in more positive reinforcement received. It is possible that this is due to the exploration noise and short episodes which would limit the maximum speed achievable. This encourages the agent to find the best direction when at this maximum speed.

However since the algorithm is only guaranteed to converge on a local minimum not all trials had equally good results. In figure 9(b) we plotted the performance of five individual trials. From this we can see that four of the trials converged to solutions which had better performance than the other methods and a single trial converged to a bad solution. Unfortunately most of the time the controllers were able to sail very efficiently into the wind on one side, but not efficiently on the other. Taking advantage of the left-right symmetry in this task would improve their performance further.

7 Discussion

This article originally had the intent of examining the claim made in [Adriaans, 2003] that reinforcement learning when applied to sailing had an incredibly slow convergence rate. While standard reinforcement techniques were slow there exist several techniques which are able to speed reinforcement learning, these include spatio-temporal traces and adaptive discretization.

However to find an improved controller we needed to use wire-fitting, which was comparatively slow. It should be pointed out that in [van Aartrijk and Samoocha, 2003] data-mining is performed on a database containing twelve million entries. Given this amount of experience we feel it would be possible to achieve good performance using reinforcement learning for several of the behaviours in Adriaans' agent based decomposition.

Tabular learning was unable to sail a course, due to the discontinuous actions it would take. This would both slow the boat and were not sufficiently fine-grained to ensure the boat was completely controllable. Smith's method was able to reproduce the performance of the tabular methods with only a fraction of the resources, yet this was still insufficient. Wire-fitting was able to find a solution which sailed a course faster than the hand-coded controller. It was able to do this by travelling more directly to the way-points. While it is a bit strange that the solutions found received less reward than the hand-coded controller, we feel this can be explained through the effects of the random exploratory noise. The noise limited the maximum speed the boat could travel at and the agent then focussed on sailing as directly as possible instead.

7.1 Future Research

In both Smith's method and wire-fitting there was a small amount of fine-tuning noise which was added to each action. In a real system one would not be able to simply add random noise, but one could systematically fine-tune the action with a coherent exploration strategy. This would enable the autopilot to slowly adapt to each boat, an advantage which is not present in the current architecture of Adriaans.

This forms the suggested direction of future research; to apply reinforcement learning to other behaviours in the behaviour-based decomposition of Adriaans in order to determine whether or not such a compositional approach to reinforcement learning is viable. However several other research directions arose which also appear promising. These are listed below.

7.1.1 Interpolation

We feel Smith's method performed extremely well considering its limited resources. It might have performed even better had there been an appropriate form of interpolation. One such heuristic could consider not just the winning node in the state SOFM, but also its nearest neighbour. If the actions recommended by both nodes are reasonably similar then the average of the actions could be chosen. If the actions are quite distinct then only one action should be chosen.

This would avoid trying to average a left turn and a right turn and end up moving forward into an obstacle etc. However this approach seems to introduce

even more parameters into Smith's method which already has too many. The most natural way of defining similar actions would be to determine the distance from each action node in terms of the number of nodes separating them. However this is left as a possible future research direction.

7.1.2 Symmetry

Since the task is symmetrical between left and right, learning could be sped up if one takes advantage of this symmetry. Unfortunately it is unlikely that the performance will improve as we ran all the learning methods until they showed no increase in performance.

7.1.3 Expert Knowledge

To obtain a fair comparison of the different reinforcement learning methods we used *tabula rasa* learning. However several of the methods could be considerably improved with the incorporation of expert knowledge. This knowledge could be included through an intelligent division of the state in the standard reinforcement learning. In Smith's method expert knowledge could be used to initialise the positions of the nodes of both the state and action SOFM's. It is not clear however how to incorporate expert knowledge into the wire-fitting method.

7.2 Acknowledgements

Thanks to George Wells and George Konidaris for providing motivation and a keen eye at key points during this research. This research was made possible through a Commonwealth scholarship (ref. ZACS-2003-335) which is also greatly appreciated.

References

[Adriaans, 2003] Adriaans, P. (2003). From knowledge-based to skill-based systems: Sailing as a machine-learning challenge. In *Machine Learning: ECML 2003, 14th European Conference on Machine Learning*, volume 2837 of *Lecture Notes in Computer Science*. Springer.

[Aupetit et al., 2000] Aupetit, M., Couturier, P., and Massotte, P. (2000). Function approximation with continuous self-organizing maps using neighboring influence interpolation. In Bothe, H. and Rojas, R., editors, *Proceedings of the ICSC Symposia on Neural Computation (NC'2000) May 23-26, 2000 in Berlin, Germany*. ICSC Academic Press.

[Baird, 1993] Baird, L. (1993). Advantage updating. Technical Report WL-TR-93-1146, Wright-Patterson Air Force Base Ohio: Wright Laboratory.

[Baird, 1999] Baird, L. (1999). *Reinforcement Learning Through Gradient Descent*. PhD thesis, Carnegie Mellon University.

[Baird and Klopf, 1993] Baird, L. and Klopf, A. (1993). Reinforcement learning with high-dimensional, continuous actions. Technical Report WL-TR-93-1147, Wright-Patterson Air Force Base Ohio: Wright Laboratory.

[Baird, 1995] Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, pages 30–37.

[Bishop, 1995] Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

[Boyan and Moore, 1995] Boyan, J. A. and Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 369–376, Cambridge, MA. The MIT Press.

[Burden and Faires, 1997] Burden, R. L. and Faires, J. D. (1997). *Numerical Analysis*. International Thompson Publishing, 6th edition.

[Gaskett et al., 1999a] Gaskett, C., Wettergreen, D., and Zelinsky, A. (1999a). Q-Learning in continuous state and action spaces. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*. Springer-Verlag.

[Gaskett et al., 1999b] Gaskett, C., Wettergreen, D., and Zelinsky, A. (1999b). Reinforcement learning applied to the control of an autonomous underwater vehicle. In *Proceedings of the Australian Conference on Robotics and Automation (AuCRA99)*.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. P. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

[Kaski et al., 1998] Kaski, S., Kangas, J., and Kohonen, T. (1998). Bibliography of self-organizing map (som) papers:1981-1997. *Neural Computing Surveys*, 1:102–350.

[Kohonen, 1997] Kohonen, T. (1997). *Self-organizing maps*, volume 30 of *Springer Series in Information Sciences*. Springer-Verlag New York.

[Mahadevan and Connell, 1992] Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artif. Intell.*, 55(2-3):311–365.

[Marchaj, 1964] Marchaj, C. A. (1964). *Sailing theory and practice*. Granada Publishing Limited.

[Oja et al., 2003] Oja, M., Kaski, S., and Kohonen, T. (2003). Bibliography of self-organizing map (som) papers:1998-2001 addendum. *Neural Computing Surveys*, 3:1–156.

[Singh and Sutton, 1996] Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3):123–158.

[Smith, 2001a] Smith, A. (2001a). Applications of the self-organising map to reinforcement learning. Technical report, University of Edinburgh.

[Smith, 2001b] Smith, A. (2001b). *Dynamic generalisation of Continuous Action Spaces in Reinforcement Learning: A Neurally Inspired Approach*. PhD thesis, University of Edinburgh.

[Sterne, 2004] Sterne, P. (2004). Reinforcement sailing. Technical Report EDI-INF-IM040206, University of Edinburgh.

[Sutton and Barto, 1998] Sutton, R. and Barto, A. (1998). *Reinforcement Learning : an introduction*. Adaptive Computation and machine learning. MIT Press.

[Tesauro, 1995] Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3).

[Thompson, 2002] Thompson, D. R. (2002). Scaling up spatial credit assignment through modularity. Master's thesis, University of Edinburgh.

[Thrun and Schwartz, 1993] Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*. Erlbaum Associates.

[Timin, 2006] Timin, M. (2006). AnnEvolve website : `http://www.annevolve.sourceforge.net`.

[van Aartrijk and Samoocha, 2003] van Aartrijk, M. and Samoocha, J. (2003). Learning to sail. In *Proceedings of European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems*.

[van Aartrijk et al., 2002] van Aartrijk, M., Tagliola, C., and Adriaans, P. (2002). AI on the ocean: the robosail project. In *Proceedings of the 15th European Conference on Artificial Intelligence, ECAI 2002*.