TRANSNAV International Journal on Marine Navigation and Safety of Sea Transportation
Volume 2 Number 2 June 2008

Reinforcement Learning in Ship Handling

M. Lacki
Gdynia Maritime University, Gdynia, Poland

ABSTRACT: This paper presents the idea of using machine learning techniques to simulate and demonstrate learning behaviour in ship manoeuvring. Simulated model of ship is treated as an agent, which through environmental sensing learns itself to navigate through restricted waters selecting an optimum trajectory. Learning phase of the task is to observe current state and choose one of the available actions. The agent gets positive reward for reaching destination and negative reward for hitting an obstacle. Few reinforcement learning algorithms are considered. Experimental results based on simulation program are presented for different layouts of possible routes within restricted area.

1 INTRODUCTION

Reinforcement Learning is actually a very actively researched topic in artificial intelligence. The main idea of reinforcement learning is based on agent interactions with environment (Fig 1.)

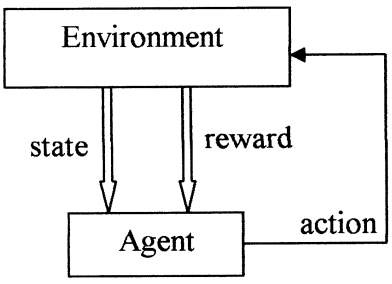

Fig. 1. General reinforcement learning model

The agent is a learning unit able to make decisions based on actual state and set of available actions. The outside element it interacts with is called the environment. In every time step agent choose one action and receives description of current situation from the environment. This situation is described by actual state and signal called reward. The agents goal is to maximize total amount of reward collected over time. In simplest case total accumulated reward is a sum of immediate rewards received in every step (Eq. 1).

$$R_t = r_{t+1} + r_{t+2} + ... + r_T \tag{1}$$

where T = terminal (final) state.

Some tasks have a continual character like process-control tasks thus there is no distinguished final state and $T \to \infty$.

Additional useful concept in this case is discounting (Sutton & Barto 1998) (Eq. 2.)

$$R_t = r_{t+1} + \gamma \cdot r_{t+2} + \gamma \cdot r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1} \tag{2}$$

where γ = discount rate and $0 \le \gamma \le 1$.

The discount rate determines importance of future rewards. Reward received k time steps later is worth γ^{k-1} times less what it would be worth when received immediately. If $\gamma<1$ then the infinite sum of rewards has a finite value. When γ is closer to one than agent takes future rewards into account more strongly thus it becomes more far-sighted.

Problems with delayed reinforcement are modeled as Markov Decision Processes (MDPs) (Kaelbling & Littman & Moore 1996).

An MDP consists of:

- a set of states S,
- a set of actions A,
- a reward function $R : S \times A \rightarrow R$
- a state transition function $T : S \times A \rightarrow P(S)$, where a member of P(S) is a probability distribution over the set S (i.e. it maps states to probabilities). We write T(s,a,s') for the probability of making a transition from state s to state s' using action a.

There is Markov Property that says that the model of environment is Markov if the state transitions are independent of any previous environment states or agent actions.

During learning process agent will choose an action according to some general rules called policy, denoted as π. Policy is a mapping from states *s* and actions *a* to the probability of π(s,a) which is taking action *a* in state *s*. Value of a state under policy π, denoted as $V^{\pi}(s)$, is the expected return when agent starts in state *s* and follows policy π.

$$V^{\pi}(s) = \mathrm{E}_{\pi}\{R_t \mid s_t = s\} = \mathrm{E}_{\pi}\{\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1} \mid s_t = s\} \quad (3)$$

Some detailed description of basic reinforcement learning algorithms is presented in the next chapter.

2 ALGORITHMS

2.1 *V-Learning*

Reinforcement learning algorithms tries to estimate value functions – values of states that say how good it is for an agent to be in given state.

In V-Learning algorithm agent learns value of visited states. The policy is created with one step state prediction for each action.

$$V(s) \leftarrow V(s) + \alpha \cdot (r + \gamma \cdot V(s') - V(s)) \quad (4)$$

where V(s) = value of state *s*; V(s') = value of next state *s'*; α = learning rate.

2.2 *Q-Learning*

Q-Learning algorithm (Sutton & Barto 1998) calculates values of state-action pairs. It tries to find an optimal state-action value function Q* independent of the policy being followed. This is off-policy temporal difference algorithm.

In every step actual Q(s,a) value is updated with δ value calculated from gained reward and maximum possible value of next state-action value function.

$$\delta \leftarrow r + \gamma \cdot \max Q(s', a') - Q(s, a) \quad (5)$$

where r = immediate reward.

Procedural form of Q-Learnig algorithm:

Initialize Q(s,a) arbitrarily

Repeat (for each episode):

Initialize *s*

Repeat (for each step of episode):

Choose *a* from *s* using policy π derived from Q

(e.g., ε-greedy)

Take action *a*, observe *r*, *s'*

$$\delta \leftarrow r + \gamma \cdot \max Q(s', a') - Q(s, a)$$
$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot \delta$$

s ← s'

until s is terminal.

In this case an agent trained using an off-policy method may end up learning tactics that it did not necessarily exhibit during the learning phase – action corresponding to maximum possible state-action value may not be chosen.

2.3 *SARSA*

SARSA is on-policy temporal difference algorithm. For each step of episode Q(s,a) value is updated with values of {s,a,r,s',a'} signals, hence the name of this algorithm.

Procedural form of SARSA algorithm:

Initialize Q(s,a) arbitrarily

Repeat (for each episode):

Initialize *s*

Choose *a* from *s* using policy derived from Q

(e.g., ε-greedy)

Repeat (for each step of episode):

Take action *a*, observe *r*, *s'*

Choose *a'* from *s'* using policy derived from Q (e.g., ε-greedy)

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot (r + \gamma \cdot Q(s', a') - Q(s, a)$$

s ← s'

a ← a'

until s is terminal.

SARSA algorithm learns during the episode that some policies are poor and switches to something else searching better positive reinforcements.

3 POLICY CONTROL

Major difference between reinforcement learning and supervised learning is that the agent must explicitly explore its environment. It is very important to make a good balance between intensive exploration of the environment and the exploitation of the learned policy to enhance the learning performance.

There are three common policies used for action selection (Eden & Knittel & Uffelen 2002):

- ε-greedy - most of the time the action with the highest estimated reward is chosen, called the greediest action but sometimes with a small probability of ε, a random action is selected uniformly, independent of the action-value estimates. This method ensures that each action will be tried many times, thus ensuring optimal actions are discovered.
- ε-soft - very similar to ε-greedy. The best action is selected with probability 1-ε and the rest of the time a random action is chosen uniformly.
- softmax – one drawback of ε-greedy and ε-soft is that they select random actions uniformly. The worst possible action is just as likely to be selected as the second best. Softmax remedies this by assigning a rank or weight to each of the actions, according to their action-value estimate. A random action is selected with regards to the weight associated with each action, meaning the worst actions are unlikely to be chosen. This is a good approach to take where the worst actions are very unfavourable.

For example in ε-soft policy one can control exploration vs. exploitation problem by decreasing value of ε accordingly to learning process.

There are also other useful solutions described in Kaelbling & Littman & Moore 1996.

4 SHIP HANDLING WITH RL

Main concept of this work is try to simulate with RL a situation of ship manoeuvring through a restricted coastal area (Fig. 2).

This task can be described in many ways. Most important is to define proper state vector from available data signals (Fig 3.), possible actions and rewards received by the agent.

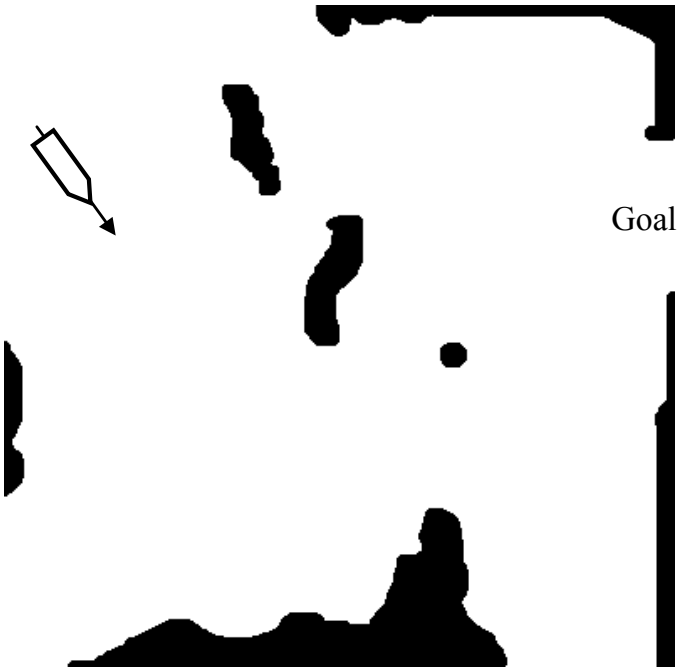

Fig. 2. Model of coastal environment

In this case the agent is the helmsman of the ship. He observes current state which can consist important signals like:

- position of ship in the area,
- ship's course (ψ),
- angular velocity (r),
- risk of grounding.

Environment is everything what is outside of the agent – in this case it is not only the restricted coast area but also a vessel steered by the helmsman.

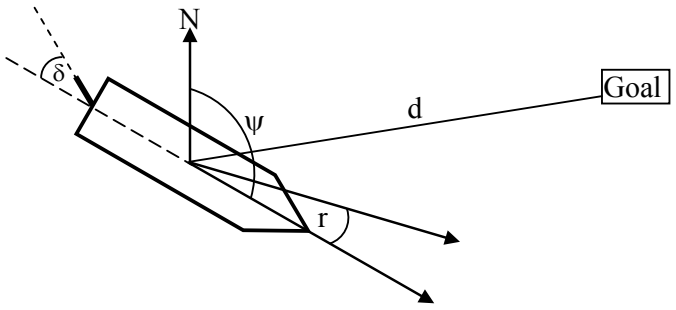

Fig. 3. Considered data signals of ship handling with RL.

Action available to take by the helmsman is one of the rudder angles (δ). The agent receives, i.e. –1 reward in every time step, –100 when ship hits an obstacle or run aground, +100 when ship reaches a

goal and –100 when she depart from the area in any other way.

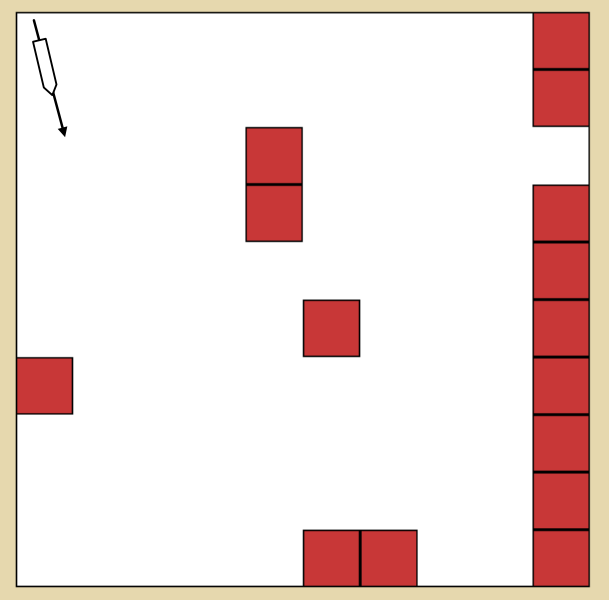

Fig. 4. Model of discretized world

There is of course many more useful signals e.g. distance to goal (d), penalty for frequent course change, negative reward for recede from goal. To simplify calculations we assume that speed of the ship is constant. Risk of grounding can be treated as multi-criteria problem which calculates a danger of getting stranded on shallow water. It can be estimated by function of ship's position, course and angular velocity.

More signals in state vector and reward function can improve projection of real coast situation to estimated state value function but also can increase computation complexity greatly. If one assume that state vector is described by 100 x 100 matrix of available position, 360 courses, 41 radial velocities and 71 rudder angles it will make more than 1mln of state-action pair real type values and it goes double with eligibility traces. One can deal with this problem by discretization of huge state space and estimate state-action pair values with common approximation methods.

In case of navigation task discretization of ship position, course and rudder angle can significantly improve learning rate with acceptable approximation of overall model to real situation fidelity.

An example of discretized state space is shown in figure 4. This is a part of application interface created and tested by the author. Experimental results showed that in simpler layouts of possible routes and few obstacles reinforcement learning SARSA algorithm was able to find proper although not optimal helmsman behavior after about 800-2000 epizodes.

There were other approaches containing weaker discretization of state space and a maps with detailed obstacles (i.e. shallow waters) like in figure 2. Additionally to improve value backups in episodic learning process an eligibility traces where used.

5 CONCLUSIONS

Experimental results with 1-step Q-Learning proves its slow learning rate which is very inconvenient in large state space problems. Eligibility traces, which bring learning closer to Monte Carlo methods, have improved learning speed. It was also very important to dynamically change the learning parameters during learning process.

SARSA algorithm uses longer but safer way during learning process accordingly to its value function update.

Using parameterised function approximation for generalization (Sutton, R. 1996) or artificial neural networks is the next step can improve reinforcement learning process in ship handling.

Some other advanced algorithms like prioritized sweeping can be taken into consideration in future work.

Furthermore splitting one agent to multi-agent environment could bring some new solutions to this problem.

REFERENCES

Eden, T. Knittel, A., Uffelen, R. 2002. Reinforcement Learning: Tutorial

Kaelbling, L.P. & Littman & Moore. 1996. Reinforcement Learning: A Survey

The Reinforcement Learning Repository, University of Massachusetts, Amherst

Sutton, R. 1996. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In Touretzky, D., Mozer, M., & Hasselmo, M. (Eds.), Neural Information Processing Systems 8.

Sutton, R. & Barto, A. 1998. Reinforcement Learning: An Introduction

Tesauro, G. 1995. Temporal Difference Learning and TD-Gammon, Communications of the Association for Computing Machinery, vol. 38, No. 3.