Reinforcement Learning In Real-Time Strategy Games

António Gusmão and Tapani Raiko
Aalto School of Science

Abstract

We consider the problem of effective and automated decision-making in modern real-time strategy (RTS) games through the use of reinforcement learning techniques. RTS games constitute environments with large, high-dimensional and continuous state and action spaces with temporally-extended actions. To operate under such environments we propose Exlos, a stable, model-based Monte-Carlo method. Contrary to existing model-based algorithms, Exlos assumes models are imperfect, reducing their influence in the decision-making process. Its effectiveness is further improved by including a novel online search procedure in the control policy. Experimental results in a testing environment show the superiority of Exlos in discrete state spaces when compared to traditional reinforcement learning methods such as Q-learning and Sarsa. Furthermore, Exlos is shown to be effective and efficient on an environment with a large continuous state and action space. This work is a summary of [Gusmao 2011].

Keywords: reinforcement learning, real-time strategy, games, artificial intelligence, UCT, planning, continuous reinforcement learning

1 Introduction

In the last decades, the video and computer game industry has been growing at an amazing rate, already being a multi-billion dollar industry, one of the biggest in the entertainment sector. In 2008, the computer and video game sales grew to 11.7 billion dollars in the United States alone, a growing trend consolidated over a 12 year-period [Entertainment Software Association 2009]. This work will focus on one game genre, real-time strategy (RTS) games, but the algorithms discussed herein have a much broader applicability, including other game genres and various real-world problems. RTS games are most usually related to warlike simulations, involving both economic development and military tactics. Opposing teams must collect resources in order to build armies and defeat their opponents. Players are faced with many tactical decisions which must be taken quickly and, in most cases, under uncertainty since enemy location and units are unknown to the player. Recent RTS games have rich worlds that can involve thousands of units deployed in massive game scenarios. Game designers stand much to gain from incorporating state-of-the-art AI techniques into their games. Perhaps more importantly, academic AI researchers have in video-games an important step between purely theoretical work and solving complex real-world problems. This is not a novel concept (see e.g. [Buro 2004]) but has yet to catch the necessary attention from the AI academic community.

RTS games are essentially complex simulators with the power to model a large diversity of real-world problems. They force researchers to come up with new, robust and efficient algorithms that can be tested in a virtual environment without requiring expensive hardware (e.g. robots) or cryptic, unintuitive simulators. With the increase in computation power in the last decades, machine-learning methodologies become valid candidates to tackle the complexity of RTS games. Several authors have proposed learning systems for RTS games [Hsieh and Sun 2008; Weber et al. 2010; Molineaux et al. 2009] but most are either not effective or operate under very limited conditions. From available techniques, reinforcement learning (RL) stands out as a method that seamlessly combines planning and learning in a framework with a wide scope of applicability. In addition, reinforcement learning is an established field with significant theoretical grounding and is currently a popular and active research topic, stimulated by recent discoveries of convergent algorithms for RL with function approximation [Maei et al. 2009; Maei et al. 2010].

Currently, there are few success cases of RL in high-dimensional and continuous state or action spaces, such as the ones encountered in RTS games. This article introduces a novel method which combines ideas from several existing algorithms, resulting in a robust learning system for stochastic environments with temporally-extended actions and continuous state and action spaces.

2 Background

2.1 Reinforcement Learning

In reinforcement learning (RL), a positive reward is awarded to a decision maker if something good happens and negative reinforcement is given if something bad happens. The decision maker attempts to adjust his decisions in order to receive increased positive rewards. A RL agent must know how to act at each state it might visit by discovering which of its current available actions lead to the largest long-term rewards, essentially learning a mapping between states and actions. The optimal actions are discovered from interaction with the environment, may that interactions be real or simulated. For an extensive introduction to reinforcement learning the reader is referred to [Sutton and Barto 1998].

2.2 UCT Algorithm

UCT [Kocsis and Szepesvári] stands for upper confidence trees and is a Monte-Carlo tree search (MCTS) method that sets up a multi-armed bandit problem for each state.

Consider a reinforcement learning agent. Denote $Q(s,a)$ as the value of action a in state s and $\mathcal{A}_s$ as the set of available action at state s. Let $\beta(s,a)$ be a real-valued bias factor. At each state s UCT takes the action with largest:

$$Q_{UCT}(s,a) = Q(s,a) + \beta(s,a) \tag{1}$$

The bias factor is:

$$\beta(s,a) = \sqrt{\frac{2b^2 \ln(\sum_{a'} N(s,a'))}{N(s,a)}}$$

where $N(s,a)$ is the number of times action a was taken in state s.

Essentially, UCT adds a bonus to each action value, the bonus being determined by the bias factor $\beta(s,a)$. Taking action a at state s results in a decrease of the bonus given to a and an increase to the one given to all other actions that could have been taken, i.e. $\beta(s,a)$ is decreased and $\beta(s,a')$ is increased for all $a' \neq a, a' \in \mathcal{A}_s$. UCT will never stop exploring. It follows directly from Equation (1) that, in an infinite number of action selections at state s, any action will have its value increased to infinity if it is not picked infinitely often.

3 Exlos

3.1 Speeding Up Learning: Simulated Experience

In our problem setting we assume action models, $P(s'|s,a)$, are available but inaccurate. Simulating experience is a way of compensating for lack of real experience, but it can become a nuisance if real experience is not lacking. A simple way to solve this issue is to combine simulated experience and real experience in a way that the former becomes less important as the latter is accumulated. The resulting algorithm possesses the advantages of model-based learning and no significant drawback. A general purpose manner of doing this is to define the value of a state as a weighted average of a simulation-based value and a value based on real experience, enforcing the weight of the simulation-value to drop to zero as the number of episodes experienced goes to infinity. Let $V_{sim}(s)$ represent the value of state s obtained from simulated experience, henceforth known as simulated value. Let $V_{real}(s)$ denote the value obtained from real experience, henceforth known as real value. Then value $V(s)$ for state s is:

$$V(s) = \frac{w_{real}V_{real}(s) + w_{sim}V_{sim}(s)}{w_{real} + w_{sim}} \quad (2)$$

where w_{real}, w_{sim} are the weights of real value and simulated value, respectively. If action models or the learning process of V_{sim} are not reliable, weights should be updated so that $V(s)$ approximates V_{real} as the number of experienced episodes goes to infinity.

For a state s_t, we compute the value $V_{sim}(s_t)$ as the average return obtained by starting at state s_t, and simulating partial episodes, $\{s_t, s_{t+1}, ..., s_{t+d}\}$, until either a terminal state is found or a threshold episode length is reached. At the final simulated state, s_{t+d}, the value of $V_{real}(s_{t+d})$ is taken as the reward for that episode and is propagated through the simulated episode back to s_t, the state where the simulation started. The simulation value function is dependent on the real value function. Hence, it makes sense to update the former only when the latter changes. Exlos updates simulated values after experiencing an episode and updating the real value function.

In essence, the simulation value function is a way to propagate real experience to larger regions of state-space, effectively extracting more information from each experienced episode. Smooth function approximators perform a similar generalization but loosely based on the assumption that state proximity corresponds to a proximity in values. However, this is not necessarily a good assumption. The simulation value function, however, generalizes based on the structure of the reinforcement learning problem, considering how actions affect the environment.

3.2 Incorporating Heuristic Estimates

In large state-spaces that result in lengthy episodes, a prior state-value function is essential. When no experience has been collected, algorithms can do no better than to select actions at random. Agents acting randomly can take a prohibitively long time to reach terminal states. In addition, it might take many episodes for a reasonable region of state space to be evaluated. Another problem comes from function approximation, which when lacking experience, generates erroneous, non-zero values for unvisited regions of state-space. A prior state-value function guides the agent in the initial stages of learning and effectively reduces the influence of poor generalization due to lack of experience. Consequently, Exlos supports the inclusion of a prior state-value function which is combined with both real and simulated value of a state according to:

$$V(s) = \frac{w_{real}V_{real}(s) + w_{sim}V_{sim}(s) + w_{prior}V_{prior}(s)}{w_{real} + w_{sim} + w_{prior}} \quad (3)$$

The weight w_{prior} may be a function of state but for simplicity, we consider it a constant. The contribution of V_{prior} to $V(s)$ should approach zero as experience is collected. This can be achieved by having w_{real} represent the number of visits to state s. In that situation a constant w_{prior} represents the experience contained in the the prior value function.

3.3 Online Search

In computer chess, the minimax planning algorithm assumes that the value function is not optimal, and therefore creates a search process at each decision state, attempting to improve its action-value estimates. In reinforcement learning the focus is on learning the value function and, in most cases, no online search is performed. This is acceptable if one assumes the value function converges to the target value function, which is the optimal value function for off-policy learning, and may or may not be for on-policy learning. However, value functions represented by function approximators converge to an approximation of the target value function. Often, the estimated value function exhibits artifacts related to the functional family of the approximator. For example, multi-layer perceptrons with logistic neurons typically exhibit long and narrow ridges, whereas radial-basis function (RBF) networks generate bumpy surfaces. An online search process can look past those artifacts and correct local inaccuracies, greatly alleviating the need for a locally accurate function approximator. In turn this allows the use of smoother function approximators that are likely to generalize better. Hence, online search should be an essential part of any RL algorithm that relies on function approximation.
Searching is a non-trivial process in MDPs and increasingly so in continuous environments. The approach we propose considers each search as a reinforcement learning problem in a local environment extracted from the original environment. At decision state s, Exlos starts a reinforcement learning problem with starting state s and with modified terminal conditions. For the map environment defined in the beginning of this chapter Exlos creates a smaller squared map, centered in respect to the current position of the RL agent. The set of terminal positions for this new environment includes all terminal positions of the original environment that lie inside the squared region, plus all positions that lie outside the squared region. In other words, a simulated episode is terminated if the agent either reaches a terminal position or reaches a position outside the squared region. For positions outside the squared region, the experience value V_{real} is taken as the reward for that position.

Computational efficiency is improved by limiting the situations where the online search procedure is executed. Instead of determining an ephemeral local value function at each decision step, the value function is stored and queried in future decisions. For simplicity, the validity of the local value function is considered to be the state-space where it was computed. When the agent leaves the local environment, the local value function is discarded and will not be available even if the agent returns to states belonging to the local environment. Whereas traditional planning methods output sequences of actions, Exlos's online search outputs a policy in the form of a local value function. Therefore, Exlos is able to adapt to stochastic events whereas traditional planning is bound to follow the sequence of actions in the plan, irrespective of the actual state transitions that occur. In addition to reusing the value function, Exlos only performs online search where it is most beneficial - when the algorithm reaches a local maxima of the value function.

Since hill-climbing based on one-step lookups of the global value function is ineffective at escaping local maxima, Exlos invokes the search procedure, broadening the region of state-space effectively considered when deciding the next action to take.

3.4 Exploration

It is common for reinforcement learning agents to use ϵ-greedy or a softmax distribution to enforce exploration of the state-action space. We argue that such policies lack an adequate system that promotes exploration of rarely-visited states. Policies which rely on random behavior to explore the state-space are not appropriate for problems where the environment has a tendency to transition to low-reward states. This is the case in the map environment and in competitive games in general, where opponents constantly exploit sub-optimal decisions made by the reinforcement learning agent. We have analysed an algorithm which encourages exploration through a different mechanism - UCT. UCT explores state-action space by acting greedily in respect to an action-value function that increases the value of actions which are not taken regularly. This is a more effective method of exploration because it forces the algorithm to direct itself to states and actions which are rarely taken. We propose a mechanism inspired by UCT which encourages exploration by increasing or decreasing the value of states, resulting in a corresponding increase or decrease in action values.

Exlos reduces the values of visited states and increases the value of non-visited states. The bias, $\beta_{offline}(s)$, is determined by:

$$\beta_{offline}(s) = c\sqrt{\frac{\ln(N)}{T(s)}} \tag{4}$$

where N is the number of episodes experienced and $T(s)$ is the number of episodes in which state s was visited. The value of a state becomes:

$$V_{explor}(s) = V(s) + \beta_{offline}(s) \tag{5}$$

The final ingredient that completes Exlos is an additional bias value that increases exploration within an episode (which is already guaranteed by the stochastic policy). Denote the bias as β_{online}. The value for a state s becomes:

$$V_{explor}(s) = V(s) + \beta_{offline}(s) + \beta_{online}(s) \tag{6}$$

β_{online} is an ephemeral function which is reset at the end of each episode. It is the sum of two components, one that incites exploration by devaluing visited states during the episode and one that devalues local maxima found.

3.5 Overview of Exlos

Figure 1 summarizes the structure of Exlos.

Let us go through the decision and learning process of Exlos. A weighted average (3) combines a prior value function, V_{prior}, an experience-based value function, V_{real} and a simulation-based value function, V_{sim}:

$$V_1(s) = \frac{w_{real}V_{real}(s) + w_{sim}V_{sim}(s) + w_{prior}V_{prior}(s)}{w_{real} + w_{sim} + w_{prior}}$$

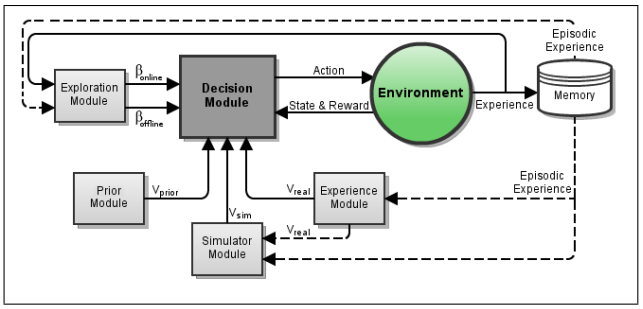

Figure 1: *Exlos structure. Dashed lines represent offline connections that are not active during execution of an episode.*

The simulation value training algorithm updates V_{sim} from values of V_{real} and both V_{real} and V_{sim} are estimated using function approximators. Visit counters are used to determined w_{real}. The exploration module encourages exploration across episodes through an added bias signal, $\beta_{offline}$ and exploration within an episode through β_{online}:

$$V(s) = V_1(s) + \beta_{offline}(s) + \beta_{online}(s)$$

Given a state-value function and knowledge of action models, action-values are computed:

$$Q(s,a) = E[V(s') \mid s,a] = \sum_{s'} P(s'|s,a)V(s')$$

With action-values calculated, the decision module picks actions following a softmax policy. In addition, an online search procedure is invoked whenever a (local) maximum of the state-value function is found. The search solves a reinforcement learning problem in a simulated local environment with $V_{real}(s)$ as the source of rewards as well as a prior for the local value function. The value function outputted by the search is used as replacement for $V_{real}(s)$.

4 Testing Environment

We implemented a set of two-dimensional rectangular maps, each map being an independent environment. The environments are episodic with fixed terminal states and each state is a position $\{x, y\}$ of the map. There are two versions of the map environments. A discrete version (a tile-based map) where x and y must be integers, and a continuous version where x and y are real-valued. Certain states cannot be visited and no action will generate them. They are represented as black squares in the map. For all purposes, they are not part of the state-space. Denote these states *walls*. Rewards are zero in all states except terminal states.There are two types of terminal states. Winning states and losing states. In a winning state the reward given is $+1$. Conversely, in a losing state the reward is -1. At each state there are four actions: move up, move down, move left and move right. The moves are not deterministic. With a certain probability a move in any direction will result in moving one tile in the direction of the closest losing state. This mimics an adversarial environment where the opponent leads the reinforcement learning agent to a losing state. In the case of continuous state maps, the set of actions is augmented with temporally-extended actions that perform movements across greater distances than the base four actions. These actions are sampled during the execution of each episode and consist of movements to locations sampled randomly from a circular area centered at the state at which the actions are available. The reinforcement learning agent solves a discounted reward problem.

5 Results

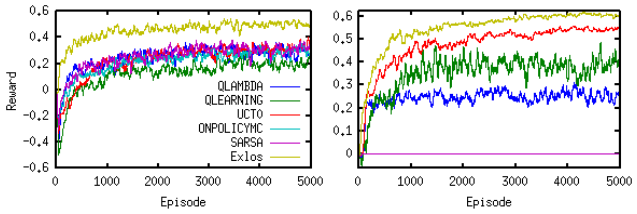

Figure 2: *Rewards obtained by popular reinforcement learning algorithms on two different, discrete map environments of small dimensions.*

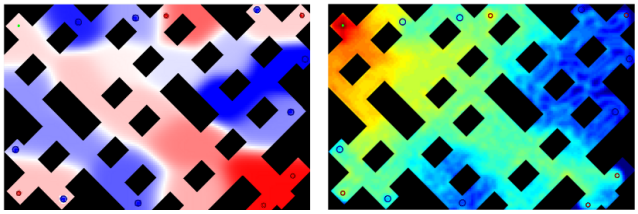

Figure 3: *A map with 1200x800 tiles. Black positions are walls, states that the agent cannot visit. The start location is in the top left. Small circles are winning locations, larger circles are losing locations. On the left the state-value function is shown, and on the right an heat map of the states visited by the agent during the learning process. Learning was performed for 200 episodes.*

6 Conclusions

We have motivated the research of real-time strategy game AI as a simulation environment for the development of novel and efficient learning algorithms that operate in complex, large-scale, stochastic environments with continuous state-action spaces and temporally-extended actions. In addition, we introduced Exlos, an effective on-policy Monte-Carlo algorithm. Exlos is a model-based algorithm that learns a state-value function. It encourages exploration of state-space by having two exploration signals, one driving exploration across episodes and the other enforcing exploration within an episode. The former is a a state-space version of the exploration bias found in UCT whereas the latter aids the algorithm to escape (local) maxima of the value function. To extract the most knowledge possible from each experienced episode, an additional state-value function is learned, the simulation value-function. It is determined by simulating short-duration partial episodes that bootstrap from the actual value-function learned from real experience. Exlos decision making is extended by an online search process. Online search is adapted to continuous state-action spaces by considering it as a reinforcement learning problem defined for a local environment analogous to the original environment. Thus, any algorithm that operates in the original environment can potentially be used for online search. Online search is computationally expensive and must be seldom relied upon. In Exlos, computational efficiency is improved by invoking online search only when it is most crucial - when a (local) maxima at a non-terminal state.

Exlos operating over a tabular representation of the state-value function was compared to Q-learning, Q(λ), on-policy Monte-Carlo and Sarsa(λ). In small-scale maps, Exlos learned approximately optimal value functions faster than the remaining algorithms. Exlos was shown to learn effectively on a large map with continuous state and action spaces and temporally-extended actions. In that map, near-optimal policies may produce plans with a thousand or more actions. No other algorithms were tested in that environment but, to our knowledge, very few algorithms in existence can cope with such complexity.

Exlos was not implemented in an actual RTS-game environment; this does not detract from the developed methods and tests performed since the testing environment mimics most of what is expected from a RTS game.

References

BURO, M. 2004. Call for AI research in RTS games. In *Proceedings of the AAAI Workshop on AI in Games*, AAAI Press, 139–141.

ENTERTAINMENT SOFTWARE ASSOCIATION, 2009. 2009 sales, demographics and usage data. Essential facts about the computer and video game industry.

GUSMAO, A. 2011. *Reinforcement Learning in Real-Time Strategy Games*. Master's thesis, Aalto School of Science, Department of Information and Computer Science.

HSIEH, J.-L., AND SUN, C.-T. 2008. Building a player strategy model by analyzing replays of real-time strategy games. In *Proceedings of IEEE International Joint Conference on Neural Networks*, 3106–3111.

KOCSIS, L., AND SZEPESVÁRI, C. Bandit based Monte-Carlo planning. In *Proceedings of ECML 2006*, vol. 4212 of *LNCS*.

MAEI, H., SZEPESVÁRI, C., BHATNAGAR, S., SILVER, D., PRECUP, D., AND SUTTON, R. 2009. Convergent temporal-difference learning with arbitrary smooth function approximation. In *NIPS*, 1204–1212.

MAEI, H., SZEPESVÁRI, C., BHATNAGAR, S., AND SUTTON, R. 2010. Toward off-policy learning control with function approximation. In *ICML*, Omnipress, J. Fürnkranz and T. Joachims, Eds., 719–726.

MOLINEAUX, M., AHA, D. W., AND MOORE, P. 2009. Learning continuous action models in a real-time strategy environment. In *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference (FLAIRS 2009)*.

SUTTON, R. S., AND BARTO, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

WEBER, B., MAWHORTER, P., MATEAS, M., AND JHALA, A. 2010. Reactive planning idioms for multi-scale game AI. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, 115–122.