Non-Linear Monte-Carlo Search in Civilization II

Bernd Nötscher

Frankfurt University of Applied Science
frankfurt-university.de

Abstract. This paper presents an innovative Monte-Carlo search algorithm, published in 2011[1], usable for very large sequential decision-making problems. The key innovation is to use a non-linear value function approximation (VFA), locally in time, for the Q function of reinforcement learning which guides the action selection of the Monte-Carlo search. This value function generalizes between related states and actions, and can therefore provide more accurate evaluations after fewer simulations. Additionally, this algorithm is extended to automatically make use of domain knowledge to improve the learning rate. This algorithm is applied to the game FreeCiv[1] (an open-source clone of Civilization II) which is very demanding due to large state space and around 10^{21} joint actions[2]. For building the Q function, a neural network, extended by domain knowledge that is created automatically from the Civilation II game manual, is applied. This non-linear Monte-Carlo search wins over 78% of games against the built-in AI of FreeCiv.

Keywords: monte-carlo search, non-linear regression, neural networks, domain knowledge, reinforcement learning

1 Introduction

Monte-Carlo search is a well-known simulation-based search paradigm, which has been successfully used to challenging games such as Poker, Go, and real-time strategy games. The idea is to get the values of states and actions via simulations also called roll-outs in order to select the best strategy. Each simulation starts with a different candidate action from the current game state and tries to find the best solution. Each simulation has n-simulations steps. At each simulation step an action is selected depending on the current state which will lead to a different state. After completion of each simulation, the game score at the end of the simulation is determined. What follows is comparing all game scores and the first played action of the simulation with the best game score is actually played. The actions played during the simulation have no effect for the new state of the

[1] http://freeciv.wikia.com. Game version 2.2

[2] These measures of the state and action spaces are empirical estimates by the experimental setup. They are based on the average number of units, available unit actions, state attributes and the observed state attribute values.

real game. The success of this approach depends on the performance of action and state processing, which is highly influenced by the used action-state selection algorithm and of course of the state space and possible actions within the game. In this case, FreeCiv is used which comes with such an enormous state space and possible actions, that without optimization Monte-Carlo search would be too demanding and therefore impractical.

A common approach to overcome this problem is value generalization by value function approximation (VFA), in which the values of all states and actions are approximated using a smaller number of parameters (regression is a synonym for function approximation). Monte-Carlo search can use VFA to estimate state and action values from the simulation scores.

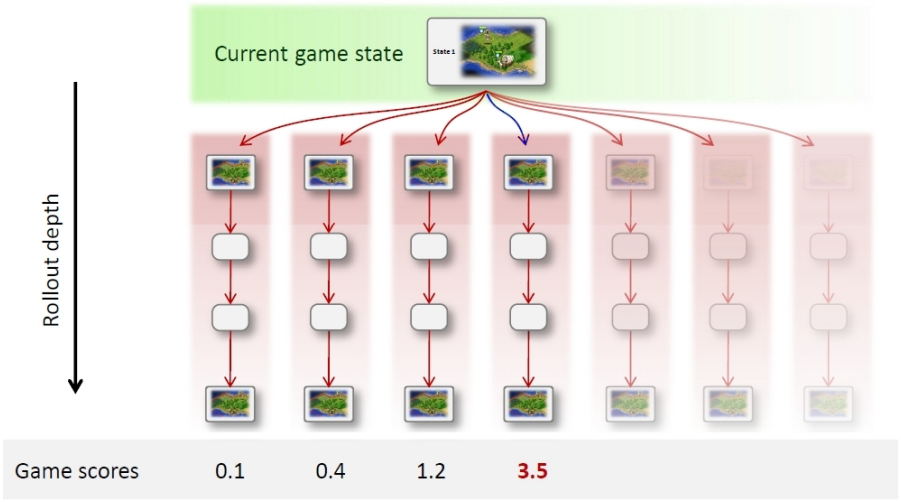

Fig. 1. Monte-carlo search: Select actions via simulations, try many candidate actions from current state and see how well they perform. The red value indicates the highest value, which means the first action of its simulation (the blue arrow) will be actually played to reach the next game state.

2 Related Work

There exists different approaches to optimize Monte-Carlo search. Most related is the Monte-Carlo search combined with global non-linear VFA[3]. In contrast the examined algorithm uses a local value function which focuses on the dynamics of the current subgame only, which may be quite specialized. It can considerably easier approximate than the full global value function which models the entire game.

Furthermore, Monte-Carlo search has been used with linear VFA[4]. Other different approaches focused on smaller, simplified or limited components of the game. Examples are reinforcement learning to dynamically select between two hard-coded strategies in the game Civilization IV, focus on city building in Civilization IV, addressing the combat aspect of strategy games only, learning high-level strategies and using hard-coded algorithms for low-level control, and focusing only on a tactical assault planning task[1].

In contrast, the aim of the algorithm of this paper is to learn to control all aspects of the game FreeCiv, and to play successfully against the hand-engineered, built-in AI player.

3 Non-linear Monte-Carlo Search

A Markov Decision Process (MDP) is used to represent the game, which comes with states, actions, transition distribution and rewards.

Each state represents a complete configuration of the game world including units, cities, landscape and their attributes. Each action represents a joint assignment of actions to each city and each unit. A game step means the execution of an action leading from one state to a new state done by the transition distribution (which includes the AI movement and game rules). This is done by using the original FreeCiv code as a black-box simulator. Each state has a reward function which is provided by the game code represented by the ratio of the game scores between the two players. For action selection, a policy is used to find the best action and explore new actions. Balancing exploring and exploitation of the policy π is done by a ϵ-greedy algorithm. The action-value function Q^{π} returns the expected final reward after executing an action in a state and is used for predicting the expected value of the candidate action. For complex games like FreeCiv, it is not possible to have action-values for every state and action combination. A common solution is to represent the action-values by function approximation.

procedure SimulateGame (s_t)

for $u = t \dots \tau$ **do**

Compute Q function approximation
$Q(s,a) = \vec{w} \cdot \vec{f}(s,a)$

Sample action from action-value function in ϵ-greedy fashion:

$$a_u \sim \begin{cases} \text{uniform}(a \in A) \text{ with probability } \epsilon \\ \arg\max_a Q(s,a) \text{ otherwise} \end{cases}$$

Execute selected action in game:
$s_{u+1} \leftarrow T(s'|s_u, a_u)$

if game is won or lost **break**

end

Update parameters $\vec{w}$ of $Q(s,a)$

Return action and observed utility:
return $a_t, R(s_\tau)$

Fig. 2. The general Monte-Carlo algorithm to simulate a game as pseudo-code.

3.1 Key features

The Q function gets adjusted online from the simulations. Therefore the approach learns by actively playing the game. This also applies to the text interpretation which learns from simulation feedback. Additional, the Q function updates in respect to the local game context making it a local value function focusing on the dynamics of the current simulation.

3.2 Advantages

Applying non-linear VFA gives two fundemental advantages. The first one is the better representational power from smaller number of simulations. Complex games such as FreeCiv require lots of computation power for Monte-Carlo search simulations which limits the possible count of simulations for each game step making it impractical to find the best action. The second advantage, is the possibility to use a multi-layered representation such as a neural network, which can automatically learn features of the game state. This also allows to use domain knowledge automatically extracted from the game manual to train and learn faster giving an overall better performance.

4 Reading manuals

Incorporating the game manual into Monte-Carlo search is done by identifing the most relevant text segment for each state. After the relevant sentence is known, the words of that sentence need to be classified into actions and states (see Figure 3). Those information is then incorporated to find the best action for the current state[2].

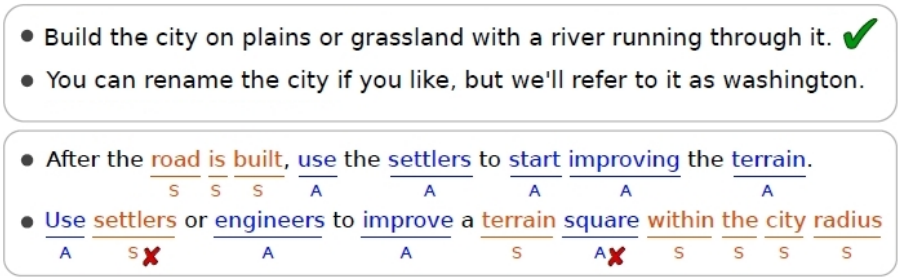

Fig. 3. Box above: An example for sentence relevance decision. The algoritmh tries to select the best action for a "Settler" unit located near a city and the first sentence is the most important. The box below shows an example for predicate labeling of a sentence: words labeled as action description are marked with an "A", and words labeled as state are marked with a "S". The remaining background words are ignored. Mistakes are marked with crosses.

5 Implementation

The following section describes the approach to non-linear VFA which can be applied effectively in the context of Monte-Carlo search. The action-value function is represented by the output of a multi-layer neural network. Non-linear is achieved by using at least one additional function for processing the values of the neural network getting more accurate values. Furthermore, this network uses a linguistic model that automatically extracts and incorporate domain knowledge from the game manual to positively guide action selection.

5.1 Four Layer Neural Network

Each layer represents a different stage of analysis. The input layer consists of deterministic, real-valued features of the current game state, candidate action, and document. The second layer is responsible of encoding the sentence-relevance and predicate-labeling decisions. The third layer combines the first two layers. Each unit in this layer corresponds to a fixed feature function. The last layer, the output layer encodes the action-value function Q which also depends on the document, as a weighted linear combination of the units in the feature layer. Furthermore, reinforcement learning is used to update the neural network through trial-and error interactions.

5.2 Parameter Estimation

Learning is performed during playing by observing the outcome of the simulations, and updating the parameters of the non-linear action-value function Q.

While applying reinforcement learning, a solution is needed to minimize mean square error between predicted value (Q function) and observed value (reward), which is done by this algorithm using gradient descent, i.e. backpropagation.

$$Q(s_t, a_t, d) = \vec{w} \cdot \vec{f}$$

$$\vec{w} \leftarrow \vec{w} + \alpha_w \left[Q - R(s_\tau)\right] \vec{f}(s, a, d, y_i, z_j)$$

Fig. 4. Box above: Here $\vec{w}$ is the weight vector. Below shows the parameter updates for the Q function.

5.3 Features

Three sets of predefined feature functions exist to convert the attributes of the game state, actions and text into real valued inputs to layers of the neural network. Figure 7 shows some examples. This algorithm results in 473,200 features, from taking the Cartesian product of all observed game attributes, all unique words in the game manual, and all possible predicate labels. That means that the algorithm makes use of a very large neural network.

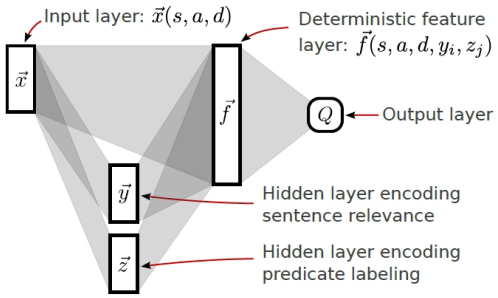

Fig. 5. The architecture of the network. Each rectangle contains a collection of units in a layer. The shaded trapezoids represents the connections between layers. A fixed, real-valued feature function $\vec{x}$(s, a, d) transforms the game state s, action a, and strategy document d into the input vector $\vec{x}$. The first hidden layer is related to the game manual incorporation containing two separated sets of units y and z. The units of the second hidden layer $\vec{f}$(s, a, d, y_i, z_i) are a set of fixed real valued feature functions on s, a, d and the active units y_i and z_i of $\vec{y}$ and $\vec{z}$ respectively.

$$x_1(s,a,d) = \begin{cases} 1 \text{ if action=build-city} \\ \quad \& \text{ tile-has-river=true} \\ \quad \& \text{ text-has-word=\{build\}} \\ 0 \text{ otherwise} \end{cases}$$

$$f_1(s,a,d,y_i,z_j) = \begin{cases} 1 \text{ if action=build-mine} \\ \quad \& \text{ tile-has-coal=true} \\ \quad \& \text{ action-words=\{mine\}} \\ \quad \& \text{ state-words=\{coal,hill\}} \\ 0 \text{ otherwise} \end{cases}$$

Fig. 6. Some example features guiding text incorparating: One task is to find the sentence most relevant to the current combination of state and candidate action. x_1 is an example of such a feature and would be active if a = "build-city", the tile in s where the action is being considered has a river, and the candidate sentence comes with the word "build". f_1 is a combination of state, action, and predicate information of the relevant sentence as input to the final layer.

6 Application to FreeCiv

The non-linear Monte-Carlo search algorithm is applied to the game of FreeCiv, but uses the official game manual of Civilization II as the source of domain knowledge. This game manual contains 2083 sentences and has a large vocabulary of 3638 words. FreeCiv is a turn-based empire-building strategy game for many players inspired by the history of human civilization. The player has to make decisions about where to build new cities, which improvements or units e.g. workers to build. The game can be won by conquering all other civilizations. The algorithm is tested on the a small size two-player game, which is set in a 1000 square map playing against the built-in AI player of FreeCiv.

7 Evaluation

For each actual game step, 500 simulations were executed from current game state, with each simulation lasting 20 simulated steps. Experiments were run on typical desktop PCs with single Intel Core i7 CPUs.

7.1 Win Rates

The algorithm is compared against the following four baselines. The percentage of victories in 100-step games while playing against the built-in game AI are in brackets: MC Tree Search (0%), Linear MC (17.3%), Non-linear MC (26.1%), Random Text (40.3%), Non-Linear Text MC (53.7%). In 50 full games played against the built-in AI, the full model won 78.8% with standard error of 5.8%.

To summarize, in both the 100-step game and full game evaluations, Non-Linear Text MC significantly outperforms all baselines. To validate the hypothesis that information extracted from manuals is useful, the baseline, Random Text, was created (being the same as the full model except the words of the used manual has been randomly rearranged) showing that using correct domain knowledge actually improves the win rate.

7.2 Computation Time

Non-linear models need more computation for action-value function estimation. To explore this possible disadvantage, a further experiment has been done. Figure 8 shows the results of this evaluation for 100, 200 and 500 rollouts.

8 Conclusions

The language analysis is not a full-scale analysis, but it is good enough to improve the learning rate. During the first steps of a game, this algorithm benifits heavily from the linguistic feature[2]. The experiments show that the best win rate can be expected when the linguistic architecture is applied. According to

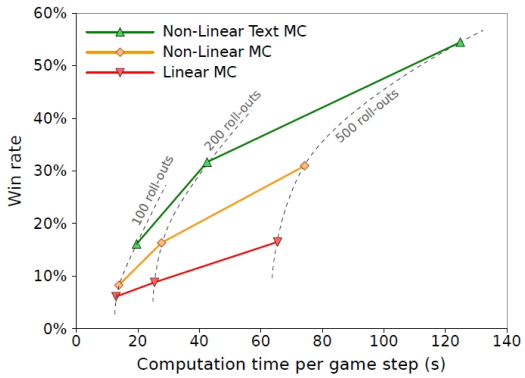

Fig. 7. Win rate as a function of computation time per game step.

the authors, the algorithm makes the strong assumption that units in the game operate independently of each other, which also limits the type of strategies that can be learned and also limits unit coordination ability. Changing it, to overcome those disadvantages, would greatly strengthen the ability to learn new type of strategies. The authors also state that human knowledge encoded in natural language can be automatically leveraged to improve control applications and that environment feedback is a powerful supervision signal for language analysis. In my opinion there are several issues not addressed in the original paper. First, the authors claim to use Civilization II, but actually use FreeCiv. There might be good reasons, not to use a more modern Civilization as testing environment, e.g. IV, but nevertheless it looks a bit lame. Second, they do not use the challenging cheating built-in AI of FreeCiv. Third, the documentation is insufficient especially source codes are not adequate documented. But on other hand, automatically incorporating domain knowledge looks very promising for future studies.

References

1. S.R.K Branavan, David Silver, and Regina Barzilay, "Non-Linear Monte-Carlo Search in Civilization II", 2011
2. S.R.K Branavan, David Silver, and Regina Barzilay, "Learning to win by reading manuals in a monte-carlo framework", 2011
3. G. Tesauro and G. Galperin, "On-line policy improvement using Monte-Carlo search.", 1996
4. D. Silver, R. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories.", 2008
5. Code, data & experimental framework, http://groups.csail.mit.edu/rbg/code/civ