Reinforcement learning of strategies for Settlers of Catan

Michael Pfeiffer
University of Zurich
42 PUBLICATIONS 323 CITATIONS

REINFORCEMENT LEARNING OF STRATEGIES FOR SETTLERS OF CATAN

Michael Pfeiffer
Institute for Theoretical Computer Science
Graz University of Technology
A 8010, Graz
Austria
E-mail: pfeiffer@igi.tugraz.at

KEYWORDS
Reinforcement Learning, Self-play, Model Trees, Settlers of Catan

ABSTRACT

In this paper we study the application of machine learning methods in complex computer games. A combination of hierarchical reinforcement learning and simple heuristics is used to learn strategies for the game Settlers of Catan (© 1995 by Kosmos Verlag, Stuttgart) via self-play. Since existing algorithms for function approximation are not well-suited for problems of this size and complexity, we present a novel use of model trees for state-action value prediction in a sophisticated computer game. Furthermore we demonstrate how a-priori knowledge about the game can reduce the learning time and improve the performance of learning virtual agents. We compare several different learning approaches, and it turns out that, despite the simplicity of the architecture, a combination of learning and built-in knowledge yields strategies that are able to challenge and even beat human players in a complex game like this.

INTRODUCTION

Developing programs that play games has always been a major challenge for artificial intelligence research. In many classical board or card games computers have reached the level of human grandmasters, or even outperformed them. On the other hand, the computer game industry has often ignored current developments in AI. Instead they rely mainly on heuristic rules, which require a lot of a-priori knowledge by the AI designer, and are also very inflexible. Machine learning provides a variety of tools to tackle these problems, and since Samuel's pioneering work (Samuel 1959) in the 1950's, researchers have used learning very successfully for games like backgammon, chess or poker (see (Fürnkranz 2001) for an overview). There are, however, only very few commercial computer games that make use of machine learning, and only few people are trying to close the gap between research and industry.

One of the most impressive applications of machine learning in game playing is Tesauro's TD-Gammon (Tesauro 1995), which used *reinforcement learning* to master the game of backgammon through self-play. This means the program uses only very little knowledge from its designer, but rather learns to approximate an evaluation function with an artificial neural network from the outcome of thousands of training games against a copy of itself. After more than one million training games and the inclusion of some backgammon specific knowledge, TD-Gammon reached a playing strength that surpasses most human grandmasters. In this paper we demonstrate how Tesauro's approach can be modified and applied to even more complex games, which are more related to commercial strategy games.

For this purpose we decided to study the game *Settlers of Catan*. Due to the monumental size of the problem, we need to think about ways to shorten the required learning time. We do this by using a hierarchical learning architecture, and by incorporating simple a-priori knowledge. For learning and representing the state-action value function we use model trees (Quinlan 1992), which are better suited for representing discontinuities and local dependencies than neural networks, as used by Tesauro.

REINFORCEMENT LEARNING

Reinforcement learning (*RL*) is used to learn strategies for an agent through interaction with the environment. RL problems are usually formulated as *Markov Decision Processes* (*MDP*s), where at every time step t the agent perceives the state of the environment s, chooses and executes an action a, receives a *reward signal* $r(s, a)$, and finds itself in a new state $s' = \delta(s, a)$. The task of the agent is to find a *policy* $\pi(s, a)$, that is, a mapping from states to actions, so as to maximize the *cumulative* (*discounted*) *reward* over time. The *discount factor* $\gamma \in [0, 1]$ thereby describes the present value of future rewards.

The quality of a policy is measured by a *value function* $V^{\pi}(s)$, which is defined as the expected discounted return if we start from state s and follow policy π (Sutton and Barto 1998). An *optimal* policy π^* is then defined as any policy satisfying $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all policies π and all states s. It is also useful to define the *Q-function* $Q^{\pi}(s, a)$ as the expected return if we take action a in state s, and thereafter follow policy π. Thus, knowing $Q^* = Q^{\pi^*}$ is sufficient to find an optimal policy, because then we can simply choose $\pi(s) = \arg\max_a Q^*(s)$ at any state s.

The idea behind most RL approaches is *policy iteration*: starting with an arbitrary policy, the agent first evaluates the Q-function of the current policy, and then improves the policy by selecting new actions that are greedy with respect to the current estimation of Q. All this is done while the agent interacts with the environment and receives numerical rewards or punishments. The best-known RL algorithms are *Q-learning* and *SARSA* (Sutton and Barto 1998).

Obviously the policy to which the algorithm converges is highly dependent on the reward signal. For games, the most natural choice of rewards is to give the agent zero reward for intermediate moves, a positive reward (e.g. +1) for winning the game and a negative (e.g. –1) or zero reward for losing. The resulting state value function then approximates the probability of winning from this state, and can be used like an evaluation function for heuristic search methods. By choosing different reward models, the designer may bias the resulting policies, e.g. by also rewarding important subgoals. This may speed up the learning process, but also bears the risk of learning policies that are not optimal in the original reward model.

Self-play is used for learning strong policies in adversarial domains. The agent learns while playing against itself, and therefore has to face increasingly stronger opponents. The major drawback of this method is that without sufficient *exploration* of the state and strategy space, the players only learn to counter a very small set of policies. This problem is particularly severe for deterministic games like chess or Go, while e.g. the dynamics of backgammon appear to be perfect for this co-evolutionary approach (Pollack and Blair 1998). Since Settlers of Catan is also a highly stochastic game, the use of self-play seems justified.

APPROXIMATION WITH MODEL TREES

In the above formulation of RL, a value for $Q(s, a)$ has to be learned for every possible state-action pair, which is impossible for very large, potentially even infinite, state spaces. One solution to overcome this problem is to define the current state using a finite number of *features*, and to approximate the Q-function as a function of a finite-dimensional parameter vector. Linear and neural network approximators, which are trained via gradient descent, are most frequently used.

Even though these methods have been successfully applied for RL tasks, we found that they have certain drawbacks that make them less suitable for complex game domains. The discrete nature of board games produces many local discontinuities in the value functions. On the other hand, linear or neural network approximators tend to smooth the value function globally. It is also often the case that the importance of certain state features changes in different situations, which is e.g. impossible to represent with linear approximators. As TD-Gammon (Tesauro 1995) has shown, neural networks can cope with all these difficulties, which of course also exist in a game like backgammon. However, the price to pay is an undesirably high number of training games which is needed before reasonable results can be obtained. Therefore it is justified to look for alternatives which are faster at learning local models and discontinuities.

In (Sridharan and Tesauro 2000) it was shown for a smaller scenario, that tree-based approximators have all the desired properties and require less training time than neural networks. So we decided to use *model trees* for function approximation in this experiment. Model trees are similar to decision trees, but instead of predicting discrete classes, they predict real valued functions. Model trees recursively partition the feature-space into regions, by choosing one attribute as a split criterion at every level of the tree. In the leaves of the trees, regression models are trained to predict a numerical value from the features of an instance. Most model tree algorithms use linear models, but in principle any function approximator can be used.

In this paper we used a variant of Quinlan's *M5* algorithm (Quinlan 1992) to learn model trees. This algorithm first grows a tree by selecting splitting criteria so as to minimize the target variable's variance, and then builds linear regression models in the nodes. Finally the tree is pruned, which means that sub-trees are replaced with leaves, as long as the prediction error of the resulting tree does not exceed a certain threshold.

In this context a separate model tree was trained for every action, and the target variable was the Q-value of this action for the current state. The main disadvantage in using model trees for value function approximation is that there is currently no algorithm for online training. To refine the predictions of a model tree, we must therefore rebuild it from scratch, using not only the new training examples, but also the old ones, that were used for the previous model trees.

We followed the ideas of (Sridharan and Tesauro 2000), in which regression trees (model trees with constant predictions in the leaves) performed very well in a much simpler environment. The goal was to develop an offline RL algorithm, suitable for learning via self-play. First a number of training matches is played, using the current model trees to define the policy. The stored game traces (state, actions and rewards) and the current estimation of the Q-functions are used to calculate the approximate Q-values of the new training examples. Also, SARSA-like updates of the older training examples are calculated, to reach convergence in the self-play process. A new model tree approximation of the Q-functions is built from the whole updated training set, and these trees are then used to play the next set of training games. Even though there is not much experience with using model trees in reinforcement learning, and no convergence results exist, we found that this algorithm yielded promising results for a complex game like this.

SETTLERS OF CATAN

Klaus Teuber's *Settlers of Catan* is probably the most popular modern board game in the German-speaking area. The island of Catan consists of 19 hexagonal land fields of different types, which are randomly placed at the beginning of the game. A typical arrangement of the board is shown in Figure 1.

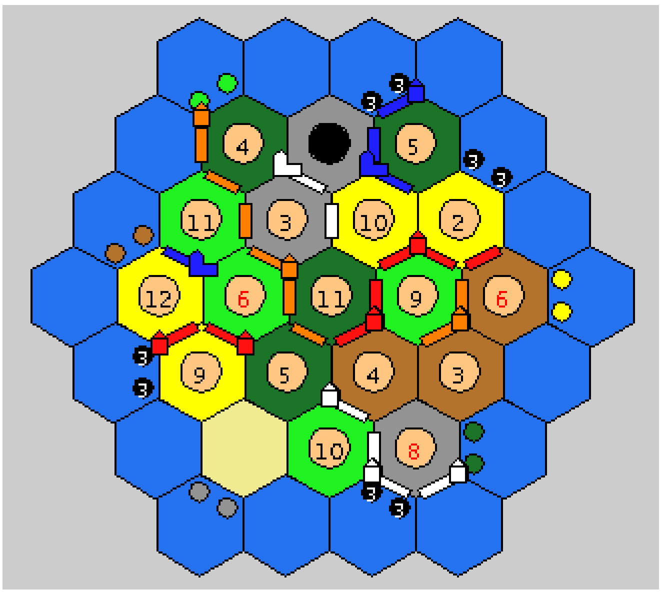

Figure 1: A typical situation in a Settlers of Catan game (screenshot from the Java simulation)

Each field is characterized by the resource it produces and the production probabilities of the field. The island is surrounded by an ocean and nine ports, which do not produce anything, but can be used for trading. Four players are colonizing the island by building roads, settlements and cities on the corners and edges of the hexagonal fields.

The players' settlements produce resources, which can be used to further expand the colony. Since only certain combinations of resources can be used for building, and players usually do not possess all of them, they have to negotiate with their opponents. The players are awarded victory points for their buildings and other special achievements. The first player to reach ten victory points wins the game.

LEARNING STRATEGIES FOR SETTLERS

The complexity of the rules and the dynamics of Settlers of Catan raise questions, that usually do not appear in classical board games. Here e.g. we have to deal with 3 opponents, and each of them can execute an arbitrary number of actions, which makes MiniMax approaches almost impossible. There is an element of chance for resource production, and interaction with the opponents is required for negotiation. Players can choose from a large action-set, and they have to balance long-term and short-term decisions, always depending on the performance of their opponents. All this places Settlers of Catan among the most complex games for which learning a full game strategy via self-play RL has ever been tried.

To make this task feasible, we used a *hierarchical RL* approach, in which the whole strategy was divided into smaller *behaviors*, for which independent policies were learned. The high-level policy first selects a behavior, and this behavior's policy chooses one of several low-level actions. Each policy (high- or low-level) is defined by a set of model trees, which approximates the action values for any given state. Given these approximate action values a controller would choose the next action with some mix of exploitation (i.e., choosing the highest valued action) and exploration.

The high-level policy receives only positive rewards for winning the game, i.e., at every time step the reward for choosing a particular behavior is zero, only at the end of the game the winning agent receives a reward of +1. The other agents also receive a reward at the end of a match, which is scaled between 0 and 1 according to their victory points. The low-level policies are only rewarded for reaching the sub-goal of the behavior, so there is only an indirect connection between high-level and low-level rewards. The collected data (states, actions and rewards) from the training games is used to train the high-level policy and all low-level policies independently. We used *a-priori* knowledge to map *low-level* actions to *primitive* actions, which are the ones that can actually be executed on the board. E.g. the low-level action `build-settlement` is mapped to a specific board position by a heuristic algorithm. However, the design of this heuristics could be simplified significantly, because the outcome of a move could be evaluated with respect to the learned value functions. This was especially useful for assessing trades with the opponents, because the profitability could naturally be estimated by the resulting change in the value function, without relying on an economic model.

Since good state-representation can significantly improve the performance of RL controllers with value-function approximation, we mainly used *high-level features*, which are calculated to summarize important facts about the current board situation, rather than feeding raw board positions into the learning algorithm.

Four different approaches of hierarchical RL were used in order to learn Settlers of Catan strategies. The *feudal* approach is inspired by the concept of feudal learning (Dayan and Hinton 1993). It lets the high-level policy select a new behavior at every time-step, and tries to learn both the high-level policy and the behaviors simultaneously. In the *module-based* approach the high-level policy changes only when the sub-goal of the last selected behavior is reached. The *heuristic* approach used a hand-coded high-level policy, but learned the low-level behaviors via self-play. And finally the *guided* approach used the same heuristic high-level strategy during learning, but then devised its own high-level policy from this experience.

EXPERIMENTS AND RESULTS

First, 1000 random games were played to provide initial training examples for the model tree learning algorithm described above. Then we ran between 3000 and 8000 training games for each approach, updating the policy after every 1000 games. The policies were evaluated in games against random players, previously learned strategies, and human opponents. Since there is no real benchmark program for Settlers of Catan, testing against a human was the only way to assess the real playing strength of the learned policies.

The learning and testing environment was written entirely in Java (Pfeiffer 2003). Due to the complexity of the game and the huge amount of training data (about 1 GB per 1000 games), the learning time was very high. Playing 1000 games in our simulation environment took about one day on a 1.5 GHz dual-processor machine. Another 20 to 24 hours were needed to train the model trees on the same computer, so the number of training matches that could be played was limited.

The results of the *feudal* approach were not satisfactory, mainly because the agent could not learn to stick to one high-level strategy when it was selected. Overfitting in the learning of model trees also occurred, which resulted in huge trees and even caused a decrease of performance for larger training sets.

The *module-based* approach outperformed the feudal strategies, but still it was not competitive in games against humans. We found that the low-level behaviors made almost the same choices, that a human would have made in the same situation with that goal in mind. The high-level strategy however was still very poor. We also could not avoid overfitting, even though this time the trees were pruned more aggressively.

Using a simple *heuristic* high-level policy in combination with learned behaviors, the virtual agents for the first time were able to beat a human opponent. This *heuristic* method was a significant improvement over the previous approaches. The amount of prior knowledge used for this method is still very small, compared to "classical" rule-based solutions. The high-level behavior plays a role in the selection of target positions on the board, e.g. for placing settlements and roads. This knowledge alone made the heuristics outperform most of the previous approaches, even with random low-level action selection, which can be seen in Figure 2.

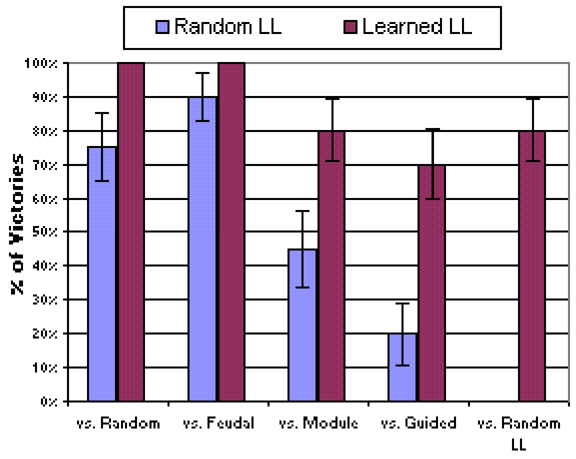

Figure 2: Performance of heuristic high-level strategies, using random or learned action selection in the low-level (LL), against other approaches (during 20 games)

Figure 2 also clearly indicates that learning is responsible for the greatest improvements in playing strength, because the trained policy wins most of the games against players who use random low-level policies, as well as against all previous approaches.

We also conducted a last experiment, in which the agents were *guided* by the heuristic high-level policy during learning. As expected, these agents did not reach the level of the heuristic agents, but their performance against humans is slightly better than that of the module-based strategies. The sub-optimal high-level strategy, caused by a lack of negative training examples (i.e., wrong choices which cause punishment instead of reward), is probably the main reason for these results.

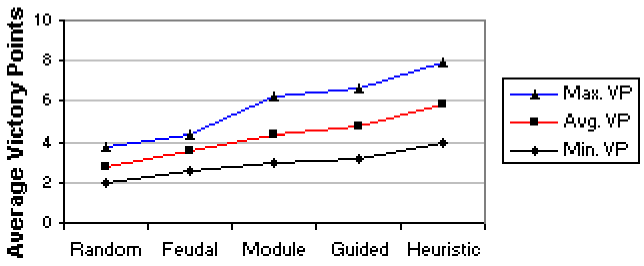

Figure 3: Performance of different strategies against a human opponent (during 10 games)

To obtain a rough estimate of the real playing strength of the different strategies, the author, who considers himself a Settlers of Catan expert, played 10 matches against the best strategies from each approach. The results of the matches are shown in Figure 3. There were always three computer players playing against the human, so the three lines show the average number of victory points of the worst, average and best artificial player. Note that a score of 10 would mean a win in every game, so the maximum of 8 points, achieved by the best *heuristic* player, indicates a pretty good performance. Actually the *heuristic* player managed to win 2 out of 10 games against the author, and at least one of the agents came close to winning in every match. Demonstration matches against other human players of various playing strengths have confirmed the competitiveness of the best learned strategies, but these results have not been included in the above statistics.

Summarizing, we can say that although the agents did not learn to play at a grandmaster-level, like in TD-Gammon, the results are encouraging, considering the complexity of this game.

CONCLUSION

In this paper we have demonstrated how reinforcement learning can be used to learn strategies for a large-scale board game. A combination of new and well-tried learning methods was used to make this problem feasible. We see this as a first step to apply advanced machine learning techniques for even more complex computer games. This research has shown that the combination of learning with prior knowledge can be a promising way to improve the performance, and ultimately also the human-like adaptiveness of agents in computer games.

ACKNOWLEDGEMENTS

The work was partially supported by the Austrian Science Fund FWF, project # P17229-N04, and PASCAL, project # IST2002-506778, of the EU.

REFERENCES

Dayan, P. and Hinton, G.E. 1993. "Feudal Reinforcement Learning" In *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, San Mateo, CA, 271-278

Fürnkranz, J. 2001. "Machine Learning in Games: A Survey." In *Machines that Learn to play Games*, J. Fürnkranz and M. Kubat, eds. Nova Scientific Pub., Huntington, N.Y., 11-59

Pfeiffer, M. 2003. "*Machine Learning Applications in Computer Games*." MSc Thesis, Graz University of Technology

Pollack, J.B. and Blair, A.D. 1998. "Co-Evolution in the successful Learning of Backgammon Strategy." *Machine Learning* 32, no. 1, 225-240.

Quinlan, J.R. 1992. "Learning with Continuous Classes." In *Proceedings of the 5th Australian Joint Conference on AI.* World Scientific, Singapore, 343-348.

Samuel, A.L. 1959. "Some Studies in Machine Learning using the Game of Checkers." *IBM Journal on Research and Development* 3, 211-229.

Sridharan, M. and Tesauro, G.J. 2000. "Multi-agent Q-Learning and Regression Trees for Automated Pricing Decisions." In *Proceedings of the 17th International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 927-934.

Sutton, R.S. and Barto, A.G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA

Tesauro, G.J. 1995. "Temporal Difference Learning and TD-Gammon." *Communications of the ACM* 38, 58-68.