Interactive Verification of Game Design and Playing Strategies

Dimitris Kalles[1,2], Eirini Ntoutsi[2,3]
AHEAD Relationship Mediators SA, Patras, GR
Department of Computer Engineering and Informatics, University of Patras, GR
Computer Technology Institute, Patras, GR
Contact: dkalles@acm.org, ntoutsi@ceid.upatras.gr

Abstract

Reinforcement learning is considered as one of the most suitable and prominent methods for solving game problems due to its capability to discover good strategies by extended self-training and limited initial knowledge. In this paper we elaborate on using reinforcement learning for verifying game designs and playing strategies. Specifically, we examine a new strategy game that has been trained on self-playing games and analyze the game performance after human interaction. We demonstrate, through selected game instances, the impact of human interference to the learning process, and eventually the game design.

1. Introduction

The game theory domain is widely regarded as appropriate for understanding the concepts of machine learning. Scientists usually focus on strategic games and make efforts to create "intelligent" programs that efficiently compete with human players. Such games are suitable because of their complexity and the opportunities they offer to explore winning strategies. Furthermore, evaluation criteria are typically known, whereas the game environment, the moves and the termination conditions can be simulated by software.

Scientists have long tried to create expert artificial players for strategy games. In 1950, Shannon began to study how computers could play chess and proposed the idea of using a value function to compete with human players [1]. In 1959, Samuel created a checkers program that tried to find "the highest point in multidimensional scoring space" [2]. Although the experiments of Samuel's research were impressive they did not exert significant influence (at that time). It was in 1988 when Sutton formulated the TD(λ) method for temporal difference learning [3]. Since then, more games such as Tetris, Blackjack, Othello [4], chess [5], backgammon [6-7] were analysed by applying TD(λ) to improve their performance. During the 1990s, IBM made strenuous efforts to develop (first with Deep Thought, later with Deep Blue) a chess program comparable to the best human player. Whether it succeeded is still mainly a philosophical and technological question.

One of the most successful and promising applications of TD(λ) is TD-Gammon [6-7] for the game of backgammon. Using reinforcement learning (RL) techniques and after training with 1.5 million self-playing games, a performance comparable to that demonstrated by backgammon world champions was achieved.

The advantage of RL to other learning methods is that it requires little ad hoc programming effort for system training. Training is achieved by a system's interaction with its environment and it is the system itself that detects which actions to take via trial and error, with limited need for direct human involvement. RL comprehends changes of the learning environment without having to be re-programmed from scratch.

As far as strategy games are concerned, the most important and critical point of them is to select and implement the computer's strategy during the game. The term *strategy* stands for the selection of the computer's next move considering its current situation, the opponent's situation, consequences of that move and possible next moves of the opponent. RL helps solve this problem by formulating strategies in terms of policies.

In this paper we continue the research of Kalles and Kanellopoulos [8] on the application of RL to the design of a new strategy game (see section below, for a detailed game description). The research demonstrated that, when trained with self-playing games, both players had nearly equal opportunities to win and neither player enjoyed a

pole position advantage. In this paper, we aim to explore the extent to which this conclusion continues to stand for the case one of the opponents is human. Specifically, we will try to give answers to questions such as: (1) Are games played from a computer against itself enough to accomplish learning? (2) Which case is more suitable for learning, a computer playing against itself or a computer playing against a human player? (3) Does playing with human players improve the computer performance much more than playing against itself?

The rest of this paper is organised in six sections. The next section presents the details of the game. It includes the basic components of the game, rules for legal pawn movements, special characteristics and playability issues. The third section refers to the game analysis; which methods are used and how they could lead towards learning. The fourth section describes training issues and experimental results. The fifth section refers to the human factor and how this affects the learning procedure. Finally, we put all the details together and discuss lines of future research that have been deemed worthy of following.

2. A description of the game

We now describe a new game, designed by one of the authors, first presented in [8]. It is a deterministic game, which seems easier than chess and backgammon to analyze. Its main difficulty arises from the fact that, during analysis, moves must be stochastically generated to ensure sufficient exploration of the state space, unlike in case of the backgammon where moves are due to dices. The challenge of designing a game with sufficient depth, and exploring its playability in respect of strategies, were the main reasons why we turned to a new game rather than use some of the already well known ones.

The game is played on a square board of size n, by two players: black and white. Two square bases of size a are located on opposite board corners. At game kick-off each player possesses β pawns. For the rest of the paper we adopt the assumption that the white player is the owner of the lower-left base (and of white pawns) while the black player is the owner of the upper-right base (and of black pawns). Each player's goal is to move a pawn into its opponent's base; the first player who achieves this is the winner.

A pawn can move to an empty square that is vertically or horizontally adjacent, provided that the maximum distance from its base is not decreased (so, backward moves are not allowed). A pawn that has no legal moves is lost (more than one pawn may be lost in one round). If some player runs out of pawns the opponent is considered to be the winner.

Legal moves can be categorized in moves of leaving the base and of moving from one square to another. The base is considered as a single square and not as a set of squares, therefore every pawn of the base can move at one step to any of the adjacent to the base free squares.

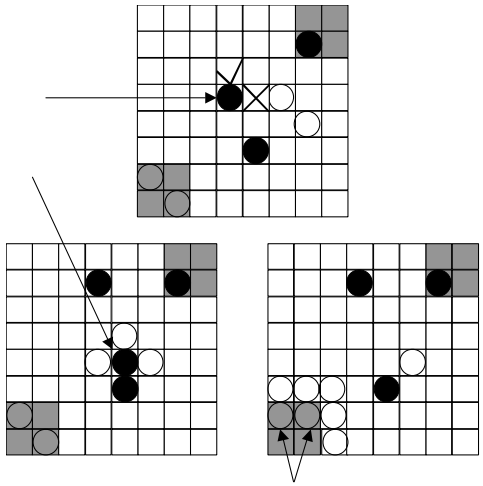

Figure 1. Examples and counterexamples of moves.

Examples and counterexamples of moves are shown in Figure 1. The upper board demonstrates a legal and an illegal move for the pawn pointed by the arrow (this pawn can move "backward" only to the ✓ square but not to the × square because its distance from base is decreased). The lower boards demonstrate moves that cause losses of pawns (with arrows showing pawn casualties). Such (loss incurring) moves bring about the direct adjustment of the moving pawn with some pawn of the opponent. In such cases the "trapped" pawn automatically draws away from the game. As a by-product of this rule, when there is no free square next to the base, the rest of the pawns of the base are lost as is shown in the bottom right game board.

3. A brief analysis of the game

The initial challenge was to design and implement a system that learns how to play through a number of self – playing games. Reinforcement Learning is ideal for this purpose since it is characterized as learning that takes place via continuous interaction of the learning agent with its environment. The agent itself detects which actions to take via trial and error learning.

The game is a discrete Markov procedure, since there are finite states and moves, and since each episode does terminate. The *a priori* knowledge of the system consists of the rules only. The agent's goal is to learn a policy π: $S \rightarrow A$ (S is the state space and A is the space of legal moves), that will maximize the expected sum of rewards

in a specific time; this is called an optimal policy. A policy determines which action should be taken next given the current state of the environment.

The move selection is critical and affects the whole learning procedure. The agent has to decide whether to choose an action that will straightforward maximize its reward or to try a new action for which it does not know anything but *may* prove to be better (the first case is known as *exploitation*, whereas the second is known as *exploration*). The answer to the above question is (in our case, too) both. Specifically, the system uses an ε-greedy policy, with $\varepsilon=0.9$, which means that in 90% of the cases the system chooses the best-valued action, while in the rest 10% it chooses a random one.

The agent estimates whether it is good for it to be in a specific position using the $V^{\pi}(s)$ value function. According to $V^{\pi}(s)$, the value of the state s of the strategy π equals to the sum of the expected rewards starting from state s and following the strategy π. Specifically, the agent is interesting in discovering the optimal strategy (the strategy that will maximize the expected sum of rewards) and for this it uses the optimal value function $V^{\pi}(s)$. Learning comes from the experience in playing or training from samples of positions taken from the game. Because of the high dimensionality and large state space of this computation we use neural networks as a generalization technique, to interpolate between game board situations.

In fact, two neural networks were used, one for each player, because each player has a unique state space. Back-propagation was used, setting the RL parameters to $\gamma=0.95$ and $\lambda=0.5$.The input layer nodes are the board positions for the next possible move, totalling n^2-2a^2+10. The hidden layer consists of half as many hidden nodes, whereas the output node has only one node, which can be regarded as the probability of winning beginning from a specific game-board configuration and then taking on a specific move.

At the beginning all states have the same value except for the final states. After each move the values are updated through the temporal-difference learning rule. The algorithm in use is TD(λ), where λ determines the reduction degree of assigning credit to some action. Using λ only, the eligible states (eligibility traces can be seen as a temporary record of the occurrence of an event, e.g. visiting a state) or actions are assigned credit or blame when a TD error occurs. We used the technique of replacing eligibility traces instead of accumulating them, because the latter approach has been known to inhibit learning, when a repeated wrong action generates a large bad trace.

4. Training issues in self-playing games

Initial experiments [8] had suggested that both computer players have nearly equal opportunities to win. However, when we tested the game performance against a human player we realized that the human player was winning almost always, independently of the moves the black player was following. Obviously the network training was not enough. Tesauro [6-7] reached a high level performance in his TD-Gammon after playing a huge number (*1,500,000*) of self-playing games. And, as Sutton and Barto [9] point out, in the case of the first *300,000* games, TD-Gammon performance was poor, games lasted hundreds or thousands of moves before one side or the other won, almost by accident.

The above symptoms arose in our game as we noticed that the initial games lasted hundred of moves with the majority of moves being cyclical between two squares. So, we kept on the training procedure and in order to speed up learning we changed the way of assigning reward. In the initial experiments, each action-move is given reward –1, unless the resulting state is a final one; then the reward is +50 for the winner's last move and –50 for the loser's last move. The new reward assignment procedure was more explicit; each action-move is assigned reward not only in final states but also during the learning procedure when it loses some pawn or when it is next to the opponent's base. Our intention was to lead the agent to learn quickly that these movements are poor and to avoid them in the future.

The new training results showed a clear improvement in computer playing even in the case it had to compete with a human player. We identified four obvious points of improvement towards the agent's goal to establish an advantage in winning the game.

First, the computer player now clearly attempts to protect its base by covering the next-to-base squares in case an opponent's pawn approaches them. This is a clear sign that the computer player has learned to protect itself against specific attacks.

Second, the back-n-forth moves were significantly decreased. Currently, the average number of moves per game has been nearly halved.

Third, the area covered by the computer player during the game has been significantly expanded. The computer player does not stop short at squares lying near its base but expands its moves so as to cover distant squares too. This is another sign that the computer player has begun to understand its goal to possess the opponent's base.

Fourth, the computer player protects its pawns. More specifically, it moves carefully so as to avoid adjacency with opponent's pawns, which might cause their loss. Towards this direction the computer player does not

occupy all next to (its) base squares; note that due to game rules, when all next-to-base squares are occupied, the remaining base pawns are lost. In previous experiments, the computer player played usually with four pawns only, as it lost all the others when it covered all next-to-base squares (see also Figure 1 for such an example).

5. The human factor in learning acceleration

The above results were all signs of game performance improvement. Aiming at greater improvement and to speed up learning we decided to examine the impact of human interference to the learning procedure. Our question was: how can a human player improve the game performance and speed up the learning procedure? And, we ask, does this have the same influence as adding handcrafted features (note that the latter has been shown to accelerate learning [6-7]).

In the proposed game we use two ways to disturb regularity: exploration and the human factor.

Training the system by self-playing games restricts the exploration to very narrow portions of the state space, due to the absence of some strong "regularity disturbance" factor. In the case of backgammon, dice play such a role and this is believed to be vital in the success of TD-Gammon. Dice produce a high degree of variability in the positions seen during training and, as a result, the learner explores more of the state space, leading to the discovery of improved evaluations and new strategies.

Through exploration the computer chooses some moves randomly. The main drawback of exploration is that it is very slow and there is no orderly exposure to skilled opponents, therefore the danger of over-fitting (in rather unimportant areas) is significant.

The human player gives the computer opportunities to explore a large state space different from what it has seen by playing against itself. A human opponent can create long-term viewed playing sequences that help a computer player to follow a **loosely guided unexplored path.** Furthermore the human player may play experimenting with a wealth of strategies and tactics, therefore reducing the risk of over-fitting during learning.

For the experiments we used a game of dimensions 8x2x10 (8: the game board dimension, 2: the base dimension, 10: the number of pawns of each player at game kick-off). Since our game is a new one, we had to train it from scratch. We used two ways to do that, by generating self-playing games and by alternating self-playing and human-playing games, in line with the observations above.

The experimental results presented below reinforce our approach. After training the network with 119,000 self-playing games, we trained it by playing alternatively self-playing games and human-computer games. More specifically, we followed the training sequence shown in Figure 2 (where light-shaded squares correspond to human-computer games).

Figure 2. The training sequence of self-playing and human-computer games.

In all these human-computer games (the even-numbered steps in the training sequence of Figure 2) the human had a specific goal: to possess the opponent's base by capturing a particular next-to-base square. Our aim was to check whether the computer could learn from human attacks and how this would affect the learning procedure. The number of human-computer games used for training was comparably smaller than the number of computer games. Our intention was to give the computer the opportunity to face situations different from those it had explored.

After playing 160 human-computer games in combination with 18,160 self-playing games (totaling 137,160 games) the computer's performance has been rapidly improved, as it almost never allowed the human to enter its base through that particular square (see Figure 3 for such a game instance).

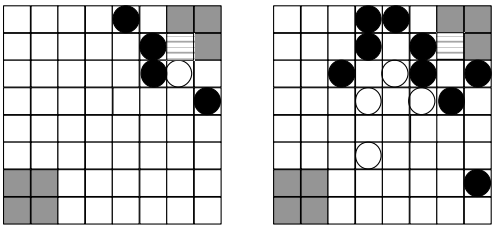

Figure 3. Performance improvement show case.

An example of game performance improvement is presented in Figure 3. In the left side we show the original performance of the game before human interference, while on the right we show the improved performance of the game after human interference. As we see, in the case of human-assisted learning, the computer

player has covered a much larger area during the game, effectively displaying a superior defensive behavior. Furthermore, the computer player has protected better its base from human-computer player attacks; note that the shaded base square before human interference –at the left- was uncovered while in the new setting it is protected. The last observation is a clear sign that the computer player has learned to protect itself against specific attacks.

To disambiguate the human impact in learning we also ran 137,160 **separate** self-playing games and we compared them with the above experiments. Experiments showed that in the second case the computer had **not** learned something **specific**. There was **little** improvement in its way of playing but this improvement was **general** and does not correspond to any specific strategy. This happens due to the slow speed of learning; the computer learns through self-playing games but such kind of learning can only be useful after a long number of self-playing games.

Our experiments suggest that learning can be accelerated through human interference, which acts as a way of exploration of new states. This result is extremely useful as RL is known to suffer from the problem of large training times, and we now demonstrate that we can aggressively fine-tune its rather-slow-guided search.

6. Experimental results

The findings are quite encouraging for learning acceleration. But does human interference contribute to the long-term game performance improvement or do we risk degrading the generality of computer playing? The latter would be surely achieved through self-playing games although the number of games required is extremely large. To explore this question we ran more experiments using the neural network weights. Specifically, we ran four sets of experiments, each set consisting of 1,000 computer-vs-computer games. Each set was based on a different training configuration though; see Table 1 for a list of configurations and related performance results.

Table 1. Cross-test of winning learning strategies.

White player (with)	Black player (with)
computer training 54.2%	computer training 45.8%
computer and human training 55.0%	computer and human training 45.0%
computer training 50.3%	computer and human training 49.7%
computer and human training 52.5%	computer training 47.5%

The term "white player with computer and human training" means that the white computer player bases its play on the knowledge received from the 137,160 compound human-computer games mentioned above (see training sequence in Figure 2), whereas the term "white player with computer training" means that the white computer player bases its play on the knowledge received from the 137,160 self-playing games mentioned above.

The above experiments show that human involvement should be carefully exercised to add value to computer performance. The human (white player) experience proved to be significantly helpful in the case of the black player; the percentage of the black player winning games has been increased from 45.8% to 49.7%. The opposite happens with the white player, whose initial goal was to train the black player with a particular defending strategy. Towards this aim, the white player was rather risky by not exploring new states, and, instead, following the minimal path that would ensure it the black's base possession. Performance percentage was decreased from 54.2% of winning games to 52.5%.

Another interesting point of the above experimental results is the performance percentage for the case where the training of both computer players contains games against a human opponent. We would expect a reduction in the white player's performance, but we were surprised to observe its performance increasing from 54.2% of winning games to 55%, which contradicts our intuition. A reason could be the (comparatively) small amount of experiments, so that a decrease of 0.8% may be actually misguiding.

7. Design assistance

Playing a game successfully requires getting and keeping the attention of a player by challenging his intellectual abilities. Some of them we practice daily: we search for solutions, we combine them, we trade-off, we plan, we observe patterns and we learn. Getting the right mix of these activities might lead to a nice game, but this

is a question of more a qualitative than quantitative nature. Short of pitting human players against each other, armed with new set of rules each time the game designer needs to modify them, one can expect that prescribing a methodology whereby game designs can be simulated and evaluated in some "objective" norms, could be a means to drastically trim down initial design cycles. We could then only concentrate in human responses when some solid knowledge about game playing is already in place. However, game openings can be very tedious, therefore, we propose that RL can be of great value.

Verifying a design should be best thought of as a process to automate the initial design steps, or as a "basic competence" test for the design itself. There will be no substitute for the talented designer but there will be a better output if we can take the boring evaluation chores out of the agenda and allow him to concentrate in essence.

This particular direction in game design is being proposed by our work. By designing a program that can evolve winning strategies (and tactics, especially end-game ones) we can expect that we will have designed a modest opponent to the human game designer who will use fewer training rounds to explore the playability of a game if new rules have to be added or constraints must be relaxed. We do not aim to improve productivity by providing fine details, which we doubt we can, but by being able to fast create knowledgeable opponents that can help human-generated exploration.

8. Conclusion

Experimental results presented in this paper show that computer performance can take advantage of human knowledge, even when such knowledge is presented to a program in a loosely structured way. By obtaining such knowledge from a breadth of (human) opponents, over-fitting problems that can arise due to too-focused experience can be avoided.

We expect to speed up learning by exploring Explanation Based Learning techniques. A combination of RL and EBL could benefit the game with faster learning and the ability to scale to large state spaces in a more structured manner [10]. Even though we have not observed the particular methodology being taken up by many other researchers, we expect that its principle of symbolic-assisted chunking of the state space is important.

A parallel improvement of practical value is to develop a benchmark computer player; however, this is best viewed as a by-product of the game design improvement.

We are confident, however, that this is a most promising research direction with widespread application implications, especially so in simulation of educational environments.

9. References

[1] C. Shannon. "*Programming a computer for playing chess"*, Philosophical Magazine, Vol. 41 (4), pp. 265-275, 1950.

[2] A. Samuel. "*Some Studies in Machine Learning Using the Game of Checkers",* IBM Journal of Research and Development Vol. 3, pp. 210-229, 1959.

[3] R.S. Sutton. "*Learning to Predict by the Methods of Temporal Differences*", Machine Learning, Vol. 3, pp. 9-44, 1988.

[4] A. Leouski. "*Learning of Position Evaluation in the Game of Othelo",* Master's project: University of Massachusetts, Amherst, 1995.

[5] S. Thrun. "*Learning to Play the Game of Chess".* Advances in Neural Information Processing Systems 7, 1995.

[6] G. Tesauro. "*Practical issues in temporal difference learning",* Machine Learning, Vol. 8, No. 3-4, 1992.

[7] G. Tesauro. "*Temporal Difference Learning and TD-Gammon",* Communications of the ACM, Vol. 38, No 3, 1995.

[8] D. Kalles and P. Kanellopoulos. "*On Verifying Game Design and Playing Strategies using Reinforcement Learning"*, ACM Symposium on Applied Computing, special track on Artificial Intelligence and Computation Logic, Las Vegas, March 2001.

[9] R. Sutton and A. Barto. "*Reinforcement Learning - An Introduction",* MIT Press, Cambridge, Massachusetts, 1998.

[10] T. Dietterich, N. Flann. "Explanation-Based Learning and Reinforcement Learning: A Unified View ", Machine Learning, Vol. 28, 1997.