Learning to play Dominoes

Ivan de Jesus P. Pinto[1], **Mateus R. Pereira**[1], **Luciano Reis Coutinho**[1]

[1]Departamento de Informática – Universidade Federal do Maranhão
São Luís,MA – Brazil

navi1921@gmail.com, mateus.rp.slz@gmail.com, luciano.rc@ufma.br

Abstract. *Dominoes is an old and popular game played around the world. Even so, there's little interest in researching artificial intelligence capable of playing it. Here we use and compare a variety of reinforcement learning-based techniques to learn the perfect information version of Dominoes, which can be applied to the original version through determinization in the future.*

1. Introduction

A domino set is formed by rectangular 1x2 tiles, with each end marked by dots indicating a number or blank to indicate zero. A traditional set contains 28 tiles with all numbers between 0 and 6, called a "double-6" set(figure 1), but larger sets can include all numbers between 0 and 9 (double-9) or 0 and 12 (double-12).

The most basic way of playing, called "Blocking game" or "Blocks", is played by two people with a double-6 set. All the tiles are shuffled face-down and form the stock or boneyard, and each player draw the seven starting tiles from there. After the initial tile is played, the players take turn to use their tiles to form a double-ended chain of dominoes, with the values matching at each link. The game ends when one player wins by playing their last tile or the game is blocked because neither player can play. In that case, the total sum of points in a player's hand is the deciding factor, winner is the one with the least amount and the opponent's points are added to his score. In a match, the winner is the first player who reaches a certain accumulated score, commonly 50, 100, 150 and etc.

Variations include being able to draw from the boneyard when blocked until they get a playable tile, called "Drawing game" or "Draw", having three players, four independent players or two teams of two players[Hoyle and Morehead 2001]. Two player Draw domino is the chosen version for this work.

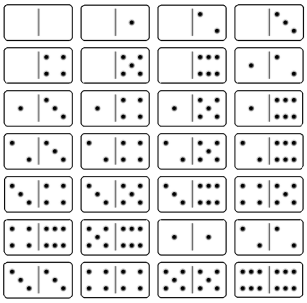

Figure 1. Domino double-6 Set

1.1. Problem Analysis

There are several aspects that turn the game of dominoes in a computational hard problem [Demaine et al. 2014]. The first is the large branching of the decision tree, which is defined by the number of possible tiles that can be played in a turn as well as their position in the board. The stochastic nature of the draws from the boneyard turn this worse, not only mid-game but also creating a large number of starting states, varying even more with a different number of players. Added to it is the the partial observability that comes from the opponent's hidden hands and boneyard [Myers 2014].

1.2. Determinization

The normal game of dominoes, with the hidden hands and stochastic boneyard, is a imperfect information game, which is a computationally hard problem. In the perfect information version the state is fully observable, with both players knowing each other hands and the stochastic behavior of the boneyard is fixed to a predetermined sequence known by the players. A perfect information variant is relatively easier to solve, and that solution can be used to construct one for the imperfect information game. Such approach is called determinization and has been used on [Borsboom et al. 2007] phantom go. Though this technique has some drawbacks, as highlighted on [Long et al. 2010], it has been successful in many sort of games. On dominoes it consists of fixing uniformly chosen pieces know not to be in our hand and field, to the opponent hand and boneyard of the current state, and evaluating the chosen action on this perfect information setting. This enable us to make use of methods with theoretical soundness or proven effectiveness on perfect information games. Many different sets of fixed cards are explored with a best possible action to be taken on each, and the final chosen action is simply the most frequent best action.

1.3. Objectives

The main objective of this paper consists on comparing different reinforcement-based algorithms. Our choice of such technique comes from its history of success on the genre of game solving, an relative fast evaluation, and use of self-play to learn. We tackle the problem of perfect information as it is more simple, keeping in mind that the solution can be applied trough determinization to the original version. The work is structured as the following: On the next section we describe the standard approach of reinforcement learning to our problem, the chosen function approximation, the chosen features and variations. The following section describes the experimental comparison of such techniques and the results. Lastly we give our conclusion of this study and future improvements.

2. Temporal Difference Learning with Function Approximation

2.1. Introduction

TD-Learning is a prediction method related to Monte-Carlo and dynamic programming, where it can learn from the environment without requiring a model and approximate the actual estimation, based on other learned estimates without waiting for the final return [Sutton and Barto 1998]. Control methods to find a optimal policy have been developed as online and offline, namely Sarsa and Q-Learning. Sarsa is called on-policy because its decision making is dependent of the policy being followed, being aware of the costs from

exploration steps. Sarsa is our algorithm of choice for control method in this paper, with its update defined as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{1}$$

where α is the learning rate and γ is the discount factor. TD control algorithms can be improved by a mechanism called eligibility trace, that permits shifting from one-step TD methods to Monte Carlo methods and between. The update is modified to:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t e_t(s, a) \tag{2}$$

where

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{3}$$

and

$$e_t(s, a) = \begin{cases} \gamma\lambda e_{t-1}(s, a) + 1 & if \;\; s \leftarrow s_t \;\; and \;\; a \leftarrow a_t \\ \gamma\lambda e_{t-1} & otherwise \end{cases} \tag{4}$$

2.2. Function Approximation with Perceptron

Reinforcement learning has a good performance on limited or minimal worlds (e.g gridworld)[Sutton and Barto 1998]. But as more challenging environment arises, the time required for visiting every state increases exponentially, and it becomes unfeasible to store on a table all possibles states. Therefore, even on more simplified cases than real world such as games, an approximation of the table is necessary. Linear function approximators such as perceptron particularly have been chosen for the task due to its simplicity to analyze the weights and efficiency given good enough features. The following considerations are made:

1. $\hat{v}$ is the approximate function
2. w is the set of weights
3. $x(s)$ = $(x_1(s)x_2(s)...x_n(s))$
4. $\hat{v}$(s,w) = $\sum_{i=1}^{n} w_i x_i$

Where on 1 and 2 $\hat{v}$ is a heuristic evaluation function and w is a set of weights that need to be learned for good representation. Weights can be adjusted by hand, by supervisioned learning from a expert, or through self-play. A standard update method for adjusting weights is stochastic gradient descent on the temporal difference error, with its rule being:

$$\nabla_w \hat{v}(S, w) = x(S) \tag{5}$$

$$\Delta w = \alpha(V_t - \hat{v}(S_t, w_t))x(S) \tag{6}$$

On 3 we have a vector of features approximating every state possible, with same number *n* of weights and 4 its linear combination of features and weights estimate how good is the given state.

2.3. Features

Finding a meaningful set of features can be a considerable manual-intensive labor validatable only by extensive experiments and often requires expert knowledge. Our approach consists on specifying a single binary feature for each domino piece in our player and opponent's hands (eg. feature 1 is true if Player 1 has piece [0,0]) and one for each possible side of the field, resulting on 28 possible features for our hand, plus 28 possible features for the opponent's hand, plus 14 for all possible features on the left and right side of the field. 28 more features for the boneyard is added, summing up a total number of 98 features. Such a set fully specifies the game, and while we've tried more high level features, their performance wasn't relevantly superior, so for sake of simplicity and speed we choose this simple representation. On the experimental setup we trained those features with the following: The learning rate was set to 0.01/(number of active features), λ was set to 0.8 and the discount factor γ was set to 0.03. The reasoning over such values is empirical, was they just happen to perform better on the experiments.

2.4. Combined Features

Automatic generation of higher level features through combinations on useful features(called atomic features) has the objective of achieving a rich set of features for more accurate evaluation function. There have been many successful applications of such techniques on games, such as GLEM on world champion-caliber Othello [Buro 1998] and expert-level skat[Buro et al. 2009] or simple Feature Construction on card game of hearts by Nathan[Sturtevant and White 2006]. In this work we apply Nathan approach, which is taking a brute-force combination of features by a logic operator. Similar to Nathan work we choose pair-wise AND combinations, resulting on 4851 features.

2.5. TD Leaf

Minmax techniques can be employed to adjust the parameters of the neural network towards the value of a deeper search, where such technique is called search bootstrapping. It has been utilized before on [Baxter et al. 2000]. This method goal is to couple deep search with a approximation method to update its parameters towards leaf nodes of the search tree or a heuristic value replacing unreachable leafs that gives the truest representation of state. The update is show as :

$$\Delta w = \alpha(V_t^D - \hat{v}(S_t^D, w_t^D))x(S) \tag{7}$$

3. Experimental Setup

Empirical experiments for evaluating the most effective technique for learning to play dominoes have been conducted by direct confront between the players, where the accumulated reward over the number of plays is the factor highlighted for choosing the best technique. As in other study's [Sturtevant 2008] we compare perfect information versions as the better solution generally is better too on the imperfect scenario. The measurement

chosen for comparison is the total reward accumulated over number of matches, since we believe that gaining most reward will always be the crucial factor regardless of a fixed quantity for winning a match. The training setups have been over 100000 matches between the players, resulting on smooth lines showing clear dominance between winner players over loser players.

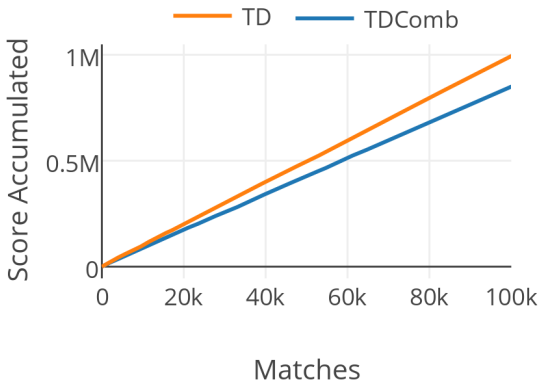

Figure 2. TD vs TDComb

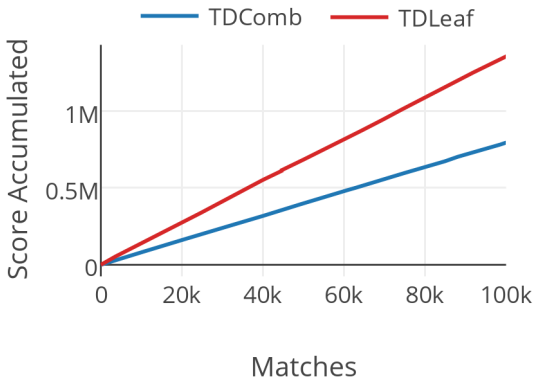

Figure 3. TDComb vs TDLeaf

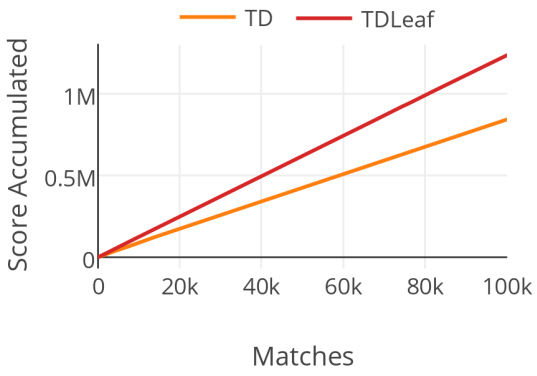

Figure 4. TD vs TDLeaf

On figure 2 the results show that applying the combination technique on the neural network renders a weaker player than expected, losing even to the standard application of TD. We speculate that there is a overfitting on training, and as we tried bigger combinations such as three-wise and four-wise the networks performance gets worse. Figures 3 and 4 show that a superior gameplay was reached by the search-trained network, TDLeaf, over both techniques.

4. Conclusion

On this paper we shown application of Reinforcement learning techniques to train neural networks for playing domino, resulting on finding the best approach to be the one with its

weights trained by search algorithm. This show us that application of search algorithms to create betters players of dominoes are interesting for this particular game. For future improvements other types of approximate functions can be evaluated such as deep neural networks, or more elaborated search bootstrapping like Rootstrap or Treestrap. Coupling the networks with search methods as heuristic values of leaf nodes is a already planned improvement, together with inference methods or explicit modeling of opponents to deal with imperfect information. Evaluation with ranked human players for knowing how strong our players are is considerable for future researches.

References

Baxter, J., Tridgell, A., and Weaver, L. (2000). Learning to play chess using temporal differences. *Machine Learning*, 40(3):243–263.

Borsboom, J., Saito, J.-T., Chaslot, G., and Uiterwijk, J. (2007). A comparison of monte-carlo methods for phantom go. In *Proc. BeNeLux Conf. Artif. Intell., Utrecht, Netherlands*, pages 57–64.

Buro, M. (1998). From simple features to sophisticated evaluation functions. In *International Conference on Computers and Games*, pages 126–145. Springer.

Buro, M., Long, J. R., Furtak, T., and Sturtevant, N. R. (2009). Improving state evaluation, inference, and search in trick-based card games. In *IJCAI*, pages 1407–1413.

Demaine, E. D., Ma, F., and Waingarten, E. (2014). *Playing Dominoes Is Hard, Except by Yourself*, pages 137–146. Springer International Publishing, Cham.

Hoyle, Edmond, M.-S. G. M. P. and Morehead, A. (2001). *Hoyle's Rules of Games 3rd Ed.* Signet.

Long, J. R., Sturtevant, N. R., Buro, M., and Furtak, T. (2010). Understanding the success of perfect information monte carlo sampling in game tree search. In *AAAI*.

Myers, M. M. (2014). *Outperforming Game Theoretic Play with Opponent Modeling in Two Player Dominoes*. Master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio.

Sturtevant, N. R. (2008). An analysis of uct in multi-player games. In *International Conference on Computers and Games*, pages 37–49. Springer.

Sturtevant, N. R. and White, A. M. (2006). Feature construction for reinforcement learning in hearts. In *International Conference on Computers and Games*, pages 122–134. Springer.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.