Monte Carlo Tree Search with temporal difference enhancement in Pommerman

Miao Yu, 151008494. Yu-Jhen Hsu, 190249963.Zhongjie Xu,190622114
Queen Mary University of London, UK
{jp2015212856, ec19264, ec19500}@qmul.ac.uk

Abstract. Monte Carlo Tree Search (MCTS) is a popular technology in the field of real-time games and has good performance comparing with the other agents in pommerman. Recently, few studies introduce using the concepts in reinforcement learning to enhance the performance in MCTS. By combining the concept of temporal difference (TD) in the tree policy, it gives a better result in the two-player game. This report implements the TD-UCT in Pommerman, being multiply agents' game, and shows it does not have significant improvement. This report also enhances the opponent model and shown the opponent model can increase the performance.

Keywords: *MCTS, TD-learning, pommerman*

1 Introduction

Monte Carlo Tree Search (MCTS) using upper confidence bounds for trees (UCT) policy is shown an outstanding performance in game playing (Perez et al. 2019, Iihan and Etaner 2017 and Vodopivee and Ster 2014). Vodopivee and Ster (2014) argue that enhancing the tree policy in MCTS by temporal difference (TD) learning can increase the win rate in two-player games. This finding gives inspiration for the performance of using TD in multiple player games.

The Pommerman game, being four players game, requires the agents preform in multiple player environment with both competitive and cooperative skills (Perez et al. 2019). It can be used in testing the performance of MCTS with TD-UCT in both personal work and teamwork under multi-players environment compare with the plain MCTS. Therefore, using Pommerman can test the performance of competitive and cooperative skills.

This report is focused on the parameter setting in TD-UCT and the performance of MCTS with TD-UCT comparing with the other forward planning method agents. The following report is divided into five sections. Firstly, the brief introduction of Pommerman will be given, followed by the introduction of the method is using in this paper. In section four, the experiment results are given, followed by the discussion of finding. The conclusion and future work are given in the last section.

2 Pommerman

Pommerman is a variant of Bomberman which introduce the concept of partially observation environment, meaning the agents can get the information within the specific range.

The 11*11 symmetric board with four agents in each corner is created in the initial stage of the game. The board contains nine types of tiles being passage, rigid, wood, bomb, flames, fog, power-ups, dummy agent and agents. Agents, being the players, can only walk through the passage and are bit allowed to walk through rigid or wood tile while a bomb can destroy the latter one and create a passage. Moreover, destroying wood tile will create a power-up by 50 chance, which allows agents to walk through and enhance their ability. There are three kinds of power-up: extra bomb, bomb range increment which increase the flames ranges of that bomb and the ability to kick the bomb. The ability of kick allows agents to kick the bomb to a wall, and it can explode while travelling. The bomb will explode after 25 ticks, and it will create the flame tile which will eliminate any agents who are within this range in that tick. The board will narrow by one grid in each side, being 10*10 board, after 500 ticks, and the narrowed area will be replaced by rigid tile and any agents within that area will be eliminated.

Each agent has 6 actions to choose during one tick, being up, left, down, right, bomb and stop. The first four actions will change the agent's position. The bomb action will drop a bomb, and the stop function will do nothing, and it is also the action the agent acts if decisions are made more than 100 milliseconds during a tick.

The goal of the game is eliminating the other agents, and it has two modes: Find for All (FFA) mode and Team mode. The FFA mode, the one who survives in the end will be the winner while in the team mode the teammate is placed in the opposite corner and the one of teammate or all of team member survive in the end will be winning team.

The game AI agents using more human-level actions and the strategies to win the game are still demanding in this game. Resnick et al. (2018) suggest that laying bombs can increase the winning rate. Perez et al. (2019) compare agents using forward planning methods in Pommerman and state plain MCTS, being laying the fewer bombs, has a better performance in both full and partial observation environment.

In the similar game: Ms Pac-Man, Pepels et al. (2014) introduce the concept of the decay rate, reduce the importance of information, to enhance the performance of MCTS and show that it provides the better result compare with the other improve methods. Therefore, it seems that using the concept of delaying reward might be able to enhance the performance in MCTS. These studies provide the idea of using delaying reward to enhance MCTS in Pommerman.

3 Background

3.1 Monte Carlo Tree Search (MCTS)

MCTS, being one of best-first tree search, can solve complex problems by the usage of sampling. It uses the domain knowledge to evaluate each action by UCT algorithm, used to balance the agent to choose the best action or explore the other actions with higher uncertainty.

The MCTS consists of four steps in each iteration- selection, expansion, rollout and backpropagation and the figure is given in figure 1. The details of each step are as follows:

- Selection: From the root node, the child is selected by tree policy recursively until reaching the expandable child.
- Expansion: An unexplored node is chosen randomly or based on the other policy and is evaluated.
- Rollout: The action is chosen by default policy until the terminate state or reaching the depth.
- Backpropagation: Update the value and the count of all visited node.

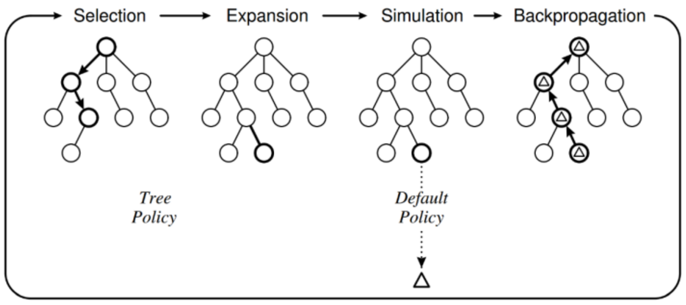

Fig.1. Main step of MCTS (Iihan and Etaner 2017)

The tree policy plays a critical role in MCTS, which is used to decide the node to visit, while the default policy decides the action randomly. The most common method uses in tree policy is UCT-1 (Perez et al.,2019), and the formula is given as follows:

$$UCB1_i = X_i + C\sqrt{\frac{2*\ln N_i^p}{N_i}} \qquad (1)$$

“ X_i is the exploitation term of the equation and $c\sqrt{\frac{2*\ln N_i^p}{N_i}}$ is the exploration term. The exploitation term is the average of the simulated scores. In exploration term,N_i^p is the parent visit count,N_i is the current node visit count. C is the constant” (Zhou et al., 2018).

The pseudocode for MCTS is provided in Algorithm1.

Algorithm 1. The iterative process of MCTS.

```
 1: procedure TD-UCT-ITERATATION ($rootNode$)
 2:     $leafNode \leftarrow$ TREEPOLICY ($rootNode$)
 3:     $R \leftarrow$ SIMULATEPLAYOUT ($leafNode$)
 4:     BACKPROPAGATE ($leafNode, R, P$)
 5: end procedure

 6: procedure TREEPOLICY (node)
 7:     while $node.state$ is not terminal do
 8:         if every child of $node$ visited at least once then
 9:             $node \leftarrow$ BESTCHILDREN ($node$)
10:         else
11:             $node \leftarrow$ random unvisited child of $node$
12:             expand tree by adding $node$ as leaf
13:             return $node$
14:         end if
15:     end while
16:     return $node$
17: end procedure

18: procedure BESTCHILD ($node$)
19:     evaluate each child of $node$ by (1)
20:     return child with highest value (break ties randomly)
21: end procedure
```

```
22: procedure SIMULATEPLAYOUT (node)
23:     $s \leftarrow node.state$
24:     while $s$ is not terminal do
25:         $s \leftarrow$ state following $a$ random action from $s$
26:     end while
27:     get reward $R$ from final state $s$
28:     return $R$
29: end procedure

30: procedure BACKPROPAGATE
31:     while node is not null do
32:         $node.state \leftarrow node.state + 1$
33:         $node.rewards \leftarrow node.rewards + R$
34:         $node \leftarrow$ parent of $node$
35:     end while
36: end procedure
```

3.2 Temporal Difference (TD) learning

TD is seen as the centre of reinforcement learning which combines the advantage of Monte Carlo, learning from the environment rather than predicts the next state based on possibility, and dynamic programming, being able to learn in part of the observation rather than all episode (Sutton and Barto, 2017). TD(λ), the λ refer to eligibility trace which will propagate updates back in time to node visit earlier, is one of the most popular methods (Sutton and Barto, 2017 and Vodopivee and Ster 2014).

TD updates the current state value V_{TD} by both immediate reward and the next state value and the formula is as follows.

$$V(s_t) \Leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (2)$$

$V(s_t)$ is the expected return starting from the state s_t. Alpha $\alpha$is the fixed step-size parameter which used to control the rate the update the value and range from 0 to 1. R_{t+1} is the reward when the TD method forms a target at time t+1. γ is a discount factor and will decrease the influence of future reward.

3.3 Temporal Difference (TD) learning in UCT

In order to further improve the performance of the UCT algorithm, the TD values Vtd and the parameters Wtd used to weigh them will be extended in the tree strategy (Vodopivee and Ster 2014). So, the new UCT formula is as follows:

$$Q_{TD-UCT} = W_{TD}V_{TD} + (1 - W_{TD})\bar{X} + C\sqrt{\frac{2 * \ln N}{N}} \quad (3)$$

The TD-UCT method requires the modification in both tree policy and backpropagation. During backpropagation, the value of TD is added. However, the update of the TD value requires Vt(st+1) of the next state, this method assumes that these values have converge and the final reward R_i is obtained from the rollout. So the node in this state will save the rollout value $r^p R_i$, where p is the number of steps of simulation time, and $V_{TD}(st)$ is the node value of the leaf in the current tree. TD error is given as follows:

$$\delta_1 = \gamma^p R_i - V_{TD}(st) \quad (4)$$

In the backpropagation step, all visited node is updated based on original update rule and TD value. TD value is given as follows:

$$V_{TD}(S) = V_{TD}(S) + \alpha_{TD}(\lambda\gamma)^{dL}\delta_1 \quad (5)$$

Four new parameters (α_{TD},λ,γ) were introduced in this rule. $(\lambda\gamma)^{dL}$ correspond that eligibility trace with initial value 1 and updated by multiplying λ and γ when updating the parent's node.

4 Experiment

The experiment is testing the TD-UCT performance compare with the other agent and explore the optimal setting for TD-UCT. The agent using TD-UCT MCTS with the opponent policy enhancement by make sure the other agent will choose the safer action to achieve the better simulation.

All experiment is running by 5 level (26456, 6883, 56352, 65105 and 55636) and each level runs 10 times. Two kinds of agents will be located in the opposite corner and run first in each corner (being located in 1,3 first and then 2,4). Therefore, the experiment runs 100 times.

The whole experiment is divided into three parts MCTS using opponent model agent compare with plain MCTS, followed by the TD-UCT agent with different weights competition. Finally, TD-UCT MCTS agent compares with the other agents.

4.1 MCTS using the opponent model experiment

The opponent model will make sure the other opponent is using the safe action rather than random action or do nothing. Opponent model is used in rollout state, which allows the state to move to the following state. Table 1 shows the win rate of using opponent model is slightly increased.

Table 1: MCTS and MCTS with opponent model (MCTS-OM) in FFA mode. The first column is vision range $\{2, \infty\}$

VR	Agents	Wins (%)	Ties (%)	Losses (%)
∞	MCTS-OM	18.5	23	58.5(-3)
	MCTS	13.5	25	61.5
2	MCTS-OM	12.5	32	55.5(-2.5)
	MCTS	10.5	31.5	58

4.2 TD-UCT MCTS weights experiment

In order to find the best parameter settings, two TD-UCT agents are needed for comparison, and one of the TD-UCTs serves as a reference to keep the original parameters unchanged. Four parameters are adjusted in this experiment, being DiscountFactor – decreasing the importance of future reward, StepRate – adjusting the Td value each time, DeceyRate – decrease the td error's influence and C – explore rate. Table 2 shows the best parameter settings for TD-UCT might be [DiscountFactor, StepRate, DeceyRate and C.]={0.5,0.03,0.3,0.2}

Table 2: Different Parameter Testing

Parameter	Agents	Wins (%)	Ties (%)	Losses (%)
Discount Factor	TD-UCT (0.1)	8.5	41.5	50
	TD-UCT (0.3)	7.5	49	43.5
	TD-UCT (0.1)	5.5	47.5	47
	TD-UCT (0.5)	10.5	47.5	42
	TD-UCT (0.1)	11	41.5	47.5
	TD-UCT (0.7)	9	44	47
	TD-UCT (0.1)	9.5	39.5	51
	TD-UCT (0.9)	7.5	47	45.5
Step Rate	TD-UCT (0.01)	9.5	42.5	48
	TD-UCT (0.005)	10	41	49
	TD-UCT (0.01)	5	46	49

	TD-UCT (0.03)	8	53.5	38.5
	TD-UCT (0.01)	9	47.5	43.5
	TD-UCT (0.05)	5	47.5	47.5
	TD-UCT (0.01)	8	46	46
	TD-UCT (0.1)	7.5	51	41.5
Decay Rate	TD-UCT (0.99)	8.5	42	49.5
	TD-UCT (0.9)	7.5	46	46.5
	TD-UCT (0.99)	7.5	55.5	37
	TD-UCT (0.7)	7.5	41	51.5
	TD-UCT (0.99)	9.5	39.5	51
	TD-UCT (0.5)	10	42.5	47.5
	TD-UCT (0.99)	6	42	52
	TD-UCT (0.3)	13.5	36	50.5
	TD-UCT (0.99)	9	43.5	47.5
	TD-UCT (0.2)	7.5	43	49.5
	TD-UCT (0.99)	9.5	34	56.5
	TD-UCT (0.1)	12.5	39	48.5
	TD-UCT (0.99)	6.5	43	50.5
	TD-UCT (0.01)	10	46.5	43
C	TD-UCT (0.2)	11	37.5	51.5
	TD-UCT (0.5)	8.5	42	49.5
	TD-UCT (0.2)	37	26	37
	TD-UCT (1)	0	0	100
	TD-UCT (0.2)	34	32	34
	TD-UCT (10)	0	0	100

4.3 TD-UCT and the other agent experiment

The performance of TD-UCT with Simpleplayer, MCTS and RHEA are tested. Table3 shows the TD-UCT has the better performance when compete with the other agents.

Table 3: TD-UCT with the other agent in vision range {2, ∞}

VR	Agents	Wins (%)	Ties (%)	Losses (%)
∞	TD-UCT	13.5	32.5	54 (-0.5)
	MCTS	9.5	36	54.5
	TD-UCT	30.5	29.5	40 (-55)
	Simpleplayer	4.5	0.5	95
	TD-UCT	28.5	43	28.5 (-71.5)
	RHEA	0	0	100
2	TD-UCT	8	49	43 (-4.5)
	MCTS	7	45.5	47.5
	TD-UCT	31	31	38 (-58)
	Simpleplayer	3	1	96
	TD-UCT	25.5	49	25.5 (-74.5)
	RHEA	0	0	100

5 Discussion

Using the opponent model in MCTS can effectively improve performance because the opponent model reduces the uncertainty of the next state, being reducing the number of possible next state. The tree policy, UCT, combines with temporal difference learning shows slightly improvement in the performance in Pommerman. The TD-UCT agents cannot have a stable winning rate comparing with MCTS since the TD is more biased, meaning the certainty of next state might influence the performance. Although TD-UCT cannot have a stable performance and cannot effectively improve the MCTS, it still has higher winning rate comparing with the agents using rule-base and forward planning methods.

The location of agents seems to influence the performance of agents. In some seeds, if the agent's performance is similar, the agents in the specific location will have much higher win rate in that specific seed. In order to address this problem, it might be a good idea to make sure the agents are placed in all position in order to get a better result. In the other word, the exchange position will be held in two experiment using the same seed, and the experimental results will be returned to the average of the two experiments.

6 Conclusions and Future Work

MCTS is still a mainstream approach in this area. Using TD-UCT to improve tree policy in multiply agent games cannot have the significant improvement compare with the improvement in two players games. Increasing the certainty of next state by better opponent model might increase the performance of TD-UCT. The opponent model shows the significant increased in winning rate while it might still can be improved by using more complex method.

In order to enhance the performance of TD-UCT by adjusting weights, it might be a good idea to combine deep learning to adjust the weights based on different situation while might increase the calculation time.

References

IIhan, E. and Etaner, S. A., 2017. Monte Carlo tree search with temporal-difference learning for general video game playing. *2017 IEEE Conference on Computational Intelligence and Games (CIG)*.

Pepels T., Winands, H. M. and Lanctot, M., 2014. Real-Time Monte Carlo Tree Search in Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in games*. 6(3):245–257.

Perez, L. D., Gaina, D. R., Drageset, O., Ilhan, E., Balla, M. and Lucas, M. S., 2019. Analysis of Statistical Forward Planning Methods in Pommerman. *In Proceedings of Artificial Intelligence in Interactive Digital Entertainment*.

Resnick, C., Eldridge, W., Ha, D., Britz, D., Foerster, J., Togelius, J., Cho, K. and Bruna, J., 2018. *Pommerman: A Multi-agent Playground*. In MARLO Workshop, AIIDE-WS Proceedings, pages 1–6.

Sutton, S. R. and Barto, G. A., 2017. *Reinforcement Learning: An Introduction*. Cambridge: The MIT Press, CH. 6.

Vodopivec, T. and Ster, B., 2014. Enhancing upper confidence bounds for trees with temporal difference values. *2014 IEEE Conference on Computational Intelligence and Games*, pp.1–8.

Zhou, H., Gong, Y., Mugrai, L., Khalifa, A., Nealen, A. and Togelius, J., 2018. A hybrid search agent in pommerman. *13th International Conference on the Foundations of Digital Games.*