Exploring Minimax and Temporal Difference Learning for Connect Four

Nick Abegg nicholas.abegg@stvincent.edu
Jake Buhite jake.buhite@stvincent.edu

I. Problem Summary

Connect Four is a simple two-player game that involves each player taking turns to place a disc onto a board. The objective for each player is to be the first to have four of their own disc in a row horizontally, vertically, or diagonally. Often, a player's strategy involves balancing the formation of four-in-a-row while simultaneously obstructing their opponent's efforts to achieve the same. The game's simplicity, turn-based nature, and immense depth make it particularly suitable for the implementation of intelligent agents such as minimax as well as Temporal Difference Learning (TDL) agents.

II. Literature Overview

A. Connect Four Agents

1) Minimax Agent: Minimax with alpha-beta pruning is particularly well suited for handling the game of Connect Four, capable of significantly reducing the time it takes for the agent to decide on a move compared to a normal minimax agent. alpha-beta Pruning enables minimax to discard certain nodes that it knows not to be useful, thus allowing it to explore the game state much faster and at a much deeper depth [1].

2) TDL Agent: Connect Four gameplay can also benefit from the use of learning agents, such as a Temporal Difference Learning (TDL) agent. A TDL agent, coupled with n-tuple networks, is capable of self-learning in playing Connect Four, eventually achieving the potential to beat an optimal minimax agent [2]. The immense potential of TDL agents becomes evident when considering that these agents, relying solely on self-play and the rules of the game for reference, demonstrate great learning capabilities.

B. Real World Applications

1) Minimax Application: In real-world applications, minimax proves valuable in minimizing manufacturing costs for equipment and maximizing the effectiveness of radiotherapy treatment against tumors while minimizing damage to healthy tissue [3]. Another application of minimax involves calculating the shortest travel time for a delivery person, taking into account real road maps and traffic information [4]. Additionally, TDL has been applied to other games such as Backgammon, being capable of learning to play from scratch at an intermediate level [5].

2) TDL Applicaiton: The flexibility of TDL in independently discovering optimal solutions makes it applicable to a variety of real-world scenarios. One particular example is utilizing TDL to manage a wide-area network to handle delays in packet transit as well as deciding the most optimal route for network traffic. This TDL agent has been demonstrated to better preserve network stability compared to conventional methods [6].

III. Algorithm Description

Although minimax and TDL were developed separately, both utilize the same Connect Four structure to facilitate the game and its rules. Minimax is a game-playing algorithm specifically designed for zero-sum games, such as Connect Four. Unlike minimax, which traverses the Connect Four state space to find the most optimal decision, TDL is a reinforcement learning algorithm in which an agent is trained to learn to make decisions over time, learning from its experience to improve its play.

A. Minimax

In Connect Four, minimax, coupled with alpha-beta pruning, efficiently calculates its next move. Additionally, it incorporates a depth limit to control the time the agent spends considering its next move. While determining its potential moves, minimax recursively analyzes possible moves at each depth, transitioning between minimizing and maximizing strategies. When minimax has reached its depth limit or the goal state is reached, it uses a utility function to generate a value for the resulting game state. The utility function considers various factors of the board, including the number of discs in a row for each player and potential win conditions. These factors generate scores that are accumulated to determine whether the game state is favorable for the minimax agent. The higher the value, the more favorable the game state is. By alternating between the minimum and maximum node values while traversing the game tree, minimax is able to account for turns in which the opponent makes their most optimal move and return the best move.

1) Pseudocode Minimax:

```
function MINIMAX(a, b, d)
  return MAXVAL(a, b, d).second

function MAXVAL(a, b, d)
  if d is 0 or game is terminal
    return UTILITY(d)
  maxV = -inf
  actions = VALIDACTIONS()
  bestMove = actions[0]
  for each move in actions
      ADDMOVE(move)
      v = MINVAL(a, b, d - 1).first
      maxV = max(maxV, v)
      if (v > maxV)
        bestMove = move
   a = max(a, maxV)
   if (a >= b) break
  return maxV, bestMove

function MINVAL(int a, int b, int d)
  if d is 0 or game is terminal
    return UTILITY(d)
  minV = +inf
  actions = VALIDACTIONS()
  bestMove = actions[0]
  for each move in actions
      ADDMOVE(move)
      v = MAXVAL(a, b, d - 1).first
      minV = min(minV, v)
      if (v < minV)
         minV = v
         bestMove = move
      b = min(b, minV)
      if (a >= b) break
  return minV, bestMove
```

2) Theoretical Complexity Analysis: The expected time complexity of minimax, with alpha-beta pruning and an optimal move order, is $O\left(b^{\frac{d}{2}}\right)$, where b denotes the branching factor, and d is the depth of the search tree. However, given that the algorithm may not have the optimal move ordering, the time complexity is likely between $O\left(b^{\frac{d}{2}}\right)$ and $O(b^d)$. The memory complexity of minimax with alpha-beta pruning is dependent on the depth of the recursion for the minimax algorithm, with the expected complexity being $O(bd)$ [7].

3) Heuristic Analysis: The heuristic utilized with the Connect Four minimax agent evaluates the current state of the game board by assessing consecutive player or opponent pieces (known as threats) in various orientations. Specifically, the heuristic assigns a score of 9999999 for an agent win and -9999999 for a player victory. Each of these values is incremented or decreased to ensure that minimax chooses moves that will differentiate moves based on the number of turns it takes to reach the game state. If one winning state is closer to the current board state than another, minimax will want to choose the winning state that is closer and requires fewer moves. If the heuristic discovers that the game is a tie, zero is returned. Otherwise, if the utility function is called because the agent's horizon has been reached, the heuristic generates a score by incorporating various strategic advantages, including three-in-a-row and two-in-a-row alignments. To strengthen the heuristic, these consecutive pieces do not affect the score unless an empty cell is detected for each missing piece in the alignment. This enables minimax to differentiate between genuine threats or advantageous moves and those that are useless or disadvantageous. Three-in-a-rows with one empty cell are given a score of 1000 and two-in-a-rows with two empty cells accruing a score of 100. Once each is calculated separately, the opponent score is subtracted from the agent's score. Overall, this heuristic ensures that minimax is efficiently choosing the most optimal move while considering its limited perspective.

B. Temporal Difference Learning

The TDL algorithm involves training a Connect Four playing agent to self-learn an effective decision-making policy through repeated interactions with the Connect Four game. The agent maintains a set of weights that are continuously adjusted as the agent adapts to the learning process. Adjustments are made to its learning rate and exploration rate over time, allowing the agent to fine-tune its strategy.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)],$$

- $V(s_t)$: The current estimated value of state s_t.
- α: The learning rate, controlling the step size of updates and influencing how quickly the agent adapts.
- r_{t+1}: The immediate reward received after taking action a_t in state s_t.
- γ: The discount factor, emphasizing the importance of future rewards in the learning process.
- $V(s_{t+1})$: The estimated value of the next state s_{t+1}.

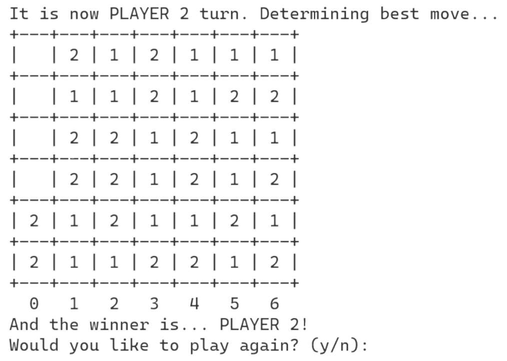
The term $[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$ represents the Temporal Difference (TD) error. This error quantifies the discrepancy between the agent's current estimate of the state's value and the sum of the immediate reward and the discounted estimated value of the next state. The agent adjusts its estimate based on this TD error, gradually improving its value estimates over successive interactions with the game [8].

Specifically, the TDL implementation trained and used to play Connect Four utilizes a set of n-tuples to efficiently extract relevant information from the Connect Four board. To generate 70 n-tuples of length 8, we utilized an approach proposed by Thill, Koch, and Konen [2]. This method involves selecting a random cell, followed by one of its neighbors, then one of the neighbors' neighbors, and so forth, until 8 positions are included in the tuple. This process is iterated for all n-tuples. By using n-tuples, the number of weights that are stored and loaded into memory changes from a potential four trillion to only 70 tuples * 4^8 (four possible positions per element in a tuple of length 8) = 4,587,520 weights. While training, the TDL agent updates its weights using the temporal difference error signal. Furthermore, the agent continuously decreases its learning rate α and exploration factor ϵ throughout the training process to maintain a balanced trade-off between exploration and exploitation.

1) Psuedocode Train Connect Four TDL Agent:

```
function TRAINTDL()
     agent1 = TDLAGENT()
     agent2 = TDLAGENT()
     games = 0
     evalCheckpoint = 20000
     previousScores[2] = {}
     while no convergence:
        set state to TRAIN
        games++
        while games % evalCheckpoint != 0
            RESETGAME()
            PLAYGAME()

            agent1.NEWALPHA()
            agent2.NEWALPHA()

            games++

        if games % 1000000 == 0:
            SAVEAGENT(agent1)
            SAVEAGENT(agent2)

        set state to EVAL
        PRINT("Alpha ", agent1.alpha)
        PRINT("Epsilon ", agent1.epsilon)

        score = AGENTEVAL()
        if score >= TARGET and
           score <= previousScores[0] and
           score <= previousScores[1]
            // Training complete
            SAVEAGENT(agent1)
            SAVEAGENT(agent2)
            return
        previousScores[1] = previousScores[0]
        prevScoreScores[0] = score
```

C. Example Connect Four Games



```
It is now PLAYER 2 turn. Determining best move...
+---+---+---+---+---+---+---+
|   | 2 | 1 | 2 | 1 | 1 | 1 |
+---+---+---+---+---+---+---+
|   | 1 | 1 | 2 | 1 | 2 | 2 |
+---+---+---+---+---+---+---+
|   | 2 | 2 | 1 | 2 | 1 | 1 |
+---+---+---+---+---+---+---+
|   | 2 | 2 | 1 | 2 | 1 | 2 |
+---+---+---+---+---+---+---+
| 2 | 1 | 2 | 1 | 1 | 2 | 1 |
+---+---+---+---+---+---+---+
| 2 | 1 | 1 | 2 | 2 | 1 | 2 |
+---+---+---+---+---+---+---+
  0   1   2   3   4   5   6
And the winner is... PLAYER 2!
Would you like to play again? (y/n):
```

Fig. 1. 6x7 Connect Four AI vs AI

IV. Testing & Data

We conducted two separate experiments on the minimax algorithm in which we recorded the runtime of the algorithm and the heap memory allocation with varying board dimensions. Both tests involved two minimax agents playing a game to completion. The test consisted of thirteen iterations, with each dimension increasing by two each iteration, starting at a standard 6x7 board and incriminating to 30x31. The depth of the minimax algorithim for all of the test was set to six.

A. Minimax Runtime Test

To conduct the runtime test, we utilized the chrono header in C++ to record the total execution time of a full Connect Four game, for each of the board sizes.

B. Minimax Memory Test

We opted to use Valgrind to conduct our memory test. Valgrind is a memory profiling tool that allows for a consistent means of recording the memory allocations of our algorithm. To run the test on each board size we constructed a Ubuntu virtual machine as well as composing a bash script which executes the Valgrind test on the program with the different board sizes. The specific memory metric we measured was total bytes allocated in the heap.

C. Minimax Test Results

Below are our results for both the memory and runtime test. Note that the x-axis labels only display every other label for sake of clarity. Fig. 2 showcases the runtime results and Fig. 3 showcases the memory results.

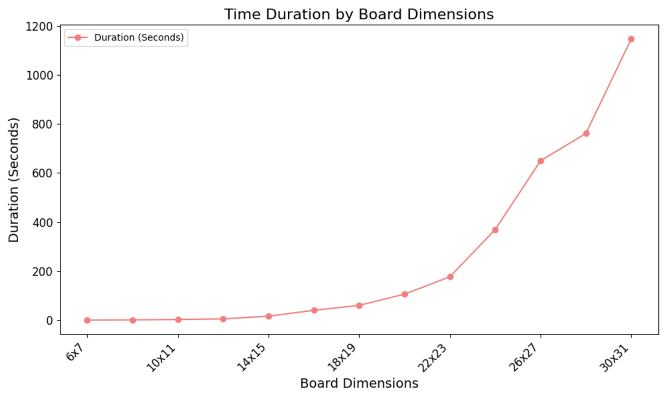


Fig. 2. Runtime with Varying Board Sizes

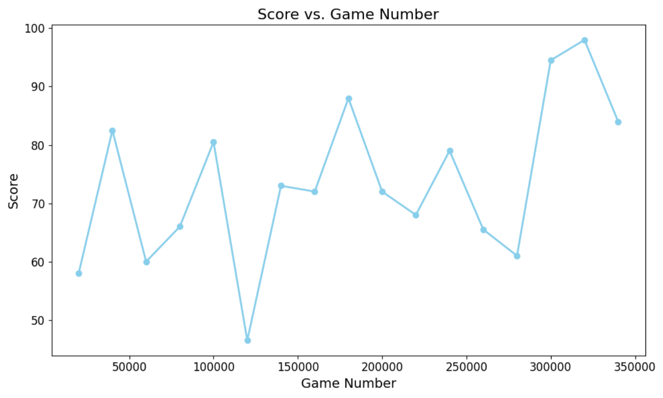


Fig. 4. TDL Agent Score Over X Games

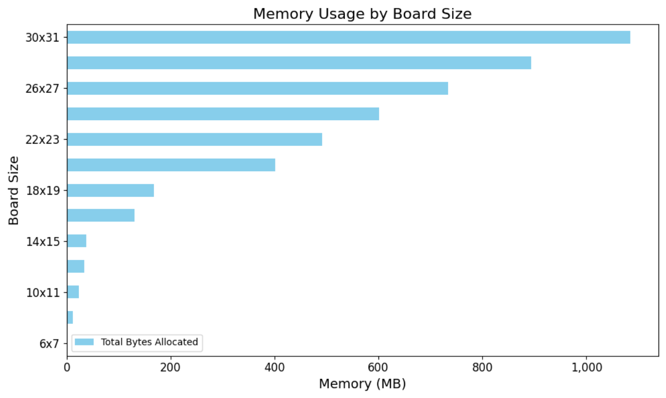


Fig. 3. Space Complexity with Varying Board Sizes

Specifically, for the 30x31 iteration, the total memory allocated is roughly 1.1 GB. The total runtime of the 30x31 took roughly 19 minutes to execute. As the size of the board increased, the time duration and memory usage increased sharply. Therefore, our findings support the expected asymptotic runtime and space complexity of minimax with alpha-beta pruning.

D. *Temporal Difference Learning Results*

The TDL agent was trained using another TDL agent. We evaluated the agent using another TDL agent every 20000 games. The evaluation score was based on 100 games, where the TDL agent's evaluation score was updated depending on whether it won (+1), lost (0), or tied (+0.5). The training ended when the agent's evaluation score was at or above 80 and less than its previous two scores.

V. Conclusion

In this paper we explore utilizing minimax with alpha-beta pruning and TDL to approach the real world game of Connect Four. Minimax was successfully implemented and was able to play the game on various board sizes. Increasing the board size resulted in an increase in memory allocated to heap as well as an increase in total runtime. The data collected in our experiments indicated that our implementation aligned with the theoretical asymptotic runtime and space complexity of minimax with alpha-beta pruning.

While our TDL agent successfully completed its training, it is not expected to perform well against agents using algorithms like minimax, as it may lack the strategic depth or optimization achieved by traditional search-based algorithms. Since the agent was not exposed to other decision-making algorithms, it's likely that it will lack the proper weights to successfully defeat these agents. The agent's inability to consistently defeat the agents can be seen in the sporadic fluctuations demonstrated in Fig. 4. Due to time constraints, priority was directed to other essential aspects of our study. As a result, the integration of minimax as a referee for our TDL agent was not achievable within the scope of this research. Future work can be conducted to integrate minimax into the TDL agent's training or evaluation and assess its impact.

Future work can be conducted on our minimax agent through investigating how increasing the depth that minimax searches effects the runtime and memory allocation. All of our experiments were conducted utilizing only a depth of six, future testing a different depths could yield additional information regarding our minimax implementation's ability.

References

[1] R. Nasa, R. Didwania, S. Maji, and V. Kumar, "Alpha-Beta pruning in Mini-Max algorithm –an optimized approach for a connect-4 game," 2018.

[2] M. Thill, P. Koch, and W. Konen, "Reinforcement learning with n-tuples on the game connect-4," in *PPSN'2012: 12th International Conference on Parallel Problem Solving From Nature*. unknown, Sep. 2012.

[3] M. Ehrgott, J. Ide, and A. Schöbel, "Minmax robustness for multi-objective optimization problems," *Eur. J. Oper. Res.*, vol. 239, no. 1, pp. 17–31, Nov. 2014.

[4] J. Ochiai and H. Kanoh, "Solving Real-World delivery problem using improved Max-Min ant system with local optimal solutions in wide area road network," *International Journal of Artificial Intelligence & Applications*, vol. 5, no. 3, pp. 21–36, May 2014.

[5] G. Tesauro, "Practical issues in temporal difference learning," in *Reinforcement Learning*. Boston, MA: Springer US, 1992, pp. 33–53.

[6] R. Yousefian and S. Kamalasadan, “Design and Real-Time implementation of optimal power system Wide-Area System-Centric controller based on temporal difference learning,” *IEEE Trans. Ind. Appl.*, vol. 52, no. 1, pp. 395–406, 2016.

[7] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson, 2009.

[8] F. Kunz, “An introduction to temporal difference learning,” 2013.