Analysis of Artificial Intelligence Techniques for Konane

SciCom Let's Talk Science

Kaitlin Hendrick

Advisor: Dr. Michael Scherger

Department of Computer Science

TCU College of Science & Engineering

Introduction

The game Konane, also known as Hawaiian checkers, is a two-player, zero-sum strategy board game ideally suited for this type of research. Players take turns making moves to capture an opponent's piece. The game ends when a player does not have a move in which he can capture an opponent's piece, and he loses the game. In order to have a successful strategy, a player must consider many future possibilities. Our research focussed on designing computer agents to play the game Konane using artificial intelligence techniques. Specifically, we aimed to create computer agents that utilized the minimax and minimax with alpha-beta pruning algorithms to select a move that would maximize the likelihood of winning the game. The outcome of this research will be an analysis of the effectiveness of each computing agent.

The Game Konane

Background:

- Konane dates back to ancient Hawaii
- Traditionally shells or pebbles are used as game pieces and the board is made out of stone with indentions carved to show where the game pieces should go
- Board can be square or rectangular ranging from at least 6x6 to 18x18

Gameplay:

- Initially setup with game pieces filling every space in an alternating checkerboard pattern
- Black removes one of his own pieces from a corner space or a space in the very center of the board
- White then removes one of his own pieces adjacent to the now empty space
- The two players take turns making moves that involve an orthogonal jump to capture an opponent's piece until a player cannot make a legal move and they lose the game

The Minimax Algorithm

The minimax algorithm presents a strategy for choosing a move that will lead to a terminal state that is a win in the game Konane. This algorithm recursively searches the game tree depth first returning the utility value of terminal nodes and assigns a utility value to parent nodes

Figure 1 shows a game tree for an arbitrary game. Each node represents a state in the game and each line is a move to get to the next state. A triangle indicates a MAX node in which is in MAX's turn to move while a circle indicates that it is the MIN player's turn. The numbers inside the shapes represent the utility value of that node. MAX will always choose a move that has a higher utility value while MIN prefers a lower value, as their names describe.

minimax-value (n) =

- utility (n) if n is a terminal state
- $\max_{action \in moves}$ minimax-value(nextboard(action)) if n is a Max node
- $\min_{action \in moves}$ minimax-value(nextboard(action)) if n is a Min node

The Minimax Algorithm Cont.

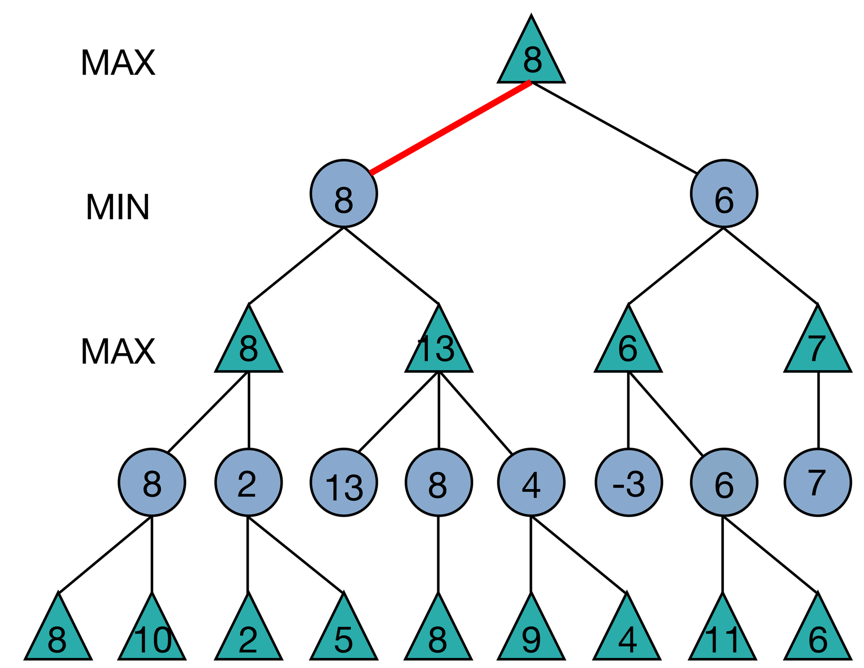


Figure 1 - Game tree for an arbitrary game with the minimax values for each node shown

```python
def minimaxDecision(self, board, player, depth):
    tup1 = self.maxValue(board, player, depth-1)
    return tup1[1]

def maxValue(self, board, player, depth):
    moves = self.generateMoves(board, player)
    if moves == [] or depth == 0:
        maxTup = (self.utility(board, player), )
        return maxTup
    v = -math.inf
    selectedMove = []
    opponent= self.opponent(player)
    for move in moves:
        testSuc = self.minValue(self.nextBoard(move), opponent, depth-1)
        if testSuc[0] > v:
            v = testSuc[0]
            selectedMove = move
    return (v, selectedMove)

def minValue(self, board, player, depth):
    moves = self.generateMoves(board, player)
    if moves == [] or depth == 0:
        minTup = (self.utility(board, player), )
        return minTup
    v = math.inf
    selectedMove = []
    opponent= self.opponent(player)
    for move in moves:
        testSuc = self.maxValue(self.nextBoard(move), opponent, depth-1)
        if testSuc[0] < v:
            v = testSuc[0]
            selectedMove = move
    return (v, selectedMove)
```

Figure 2 - Python code for minimax recursive algorithm

Once the algorithm reaches a leaf node which indicates a terminal state, the utility value of the node is returned.This value is then backed up through the tree and compared to other potential moves' utility values. Depending on which player's turn it is, an internal node's utility value becomes either the minimum or maximum utility value of all the child nodes' utility values.

Alpha-beta pruning is an additional technique that can be used in combination with the minimax algorithm. This variant of the minimax algorithm should theoretically have the same statistical chance of winning as the basic minimax algorithm but be much more computationally efficient because it eliminates branches that cannot possibly influence the final decision. For example:

minimax-value (*root*) = max(min(3, 12, 8), min(2, *x*, *y*), min(14, 5, 2))
= max(3, min(2, *x*, *y*), 2)
= max(3, *z*, 2) where $z \leq 2$
= 3

In other words, even if we let *z* be the minimum of *x* and *y*, the minimax decision for the node *root* is independent of the values of *x* and *y* so we can prune away these branches.

Evaluation

On an 8x8 game board, we evaluated a minimax agent and minimax with alpha-beta pruning agent both at a fixed depth of 5 against:

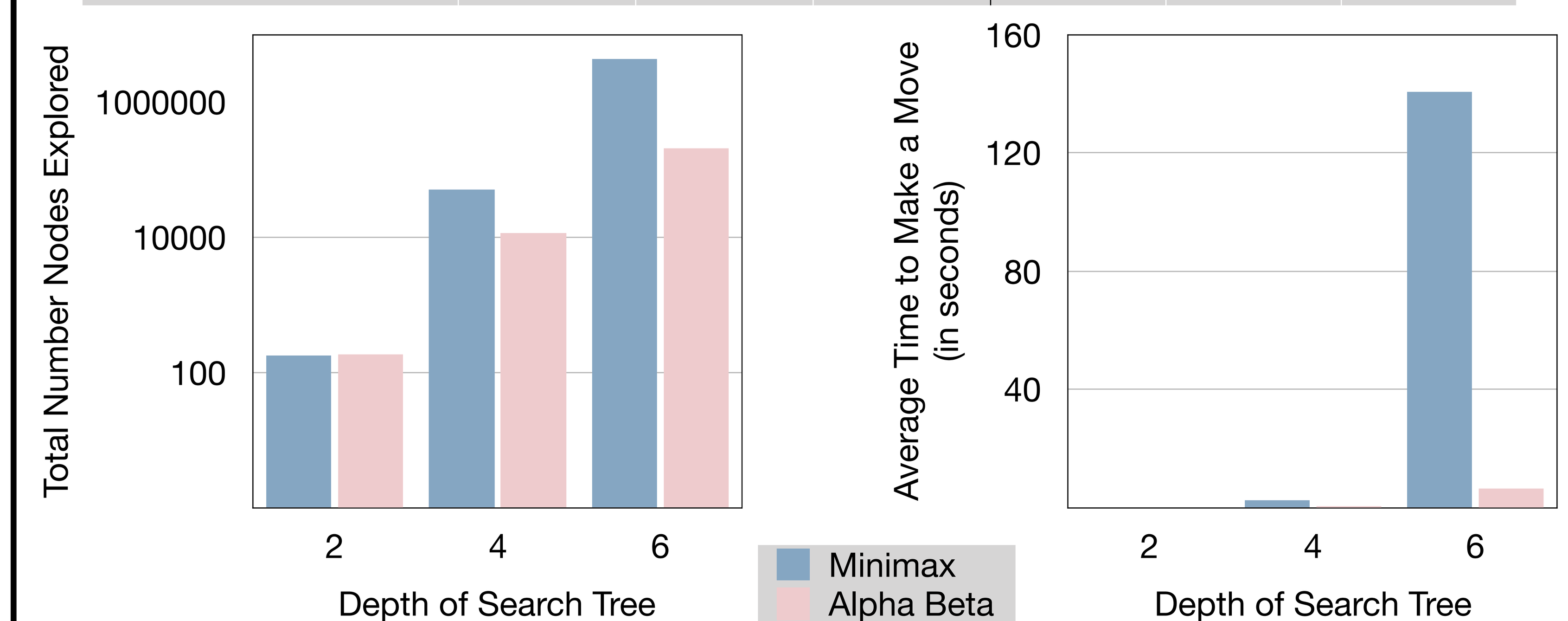
- each other
- a human agent (me)
- an agent that selects a random move for the list of potential moves
- an agent that selects a move based on the following criteria/strategy:
  - Avoid moving corner pieces
  - Move a piece that is in danger of being jumped
  - Avoid moving a piece that is not in danger of being jumped
  - Make a move that puts the player in a position to make a future move

Then, we further compared the minimax agent and minimax with alpha-beta pruning agent at various fixed depths.

Results

	Minimax Agent	Alpha-beta Agent
Wins against Human Agent:	100%	100%
Wins against Random Agent:	90%	91%
Wins against Strategy Agent:	89%	89%
Wins against other AI Agent:	~50%	~50%

	Minimax Agent			Alpha-beta Agent		
Tree Depth:	2	4	6	2	4	6
Total Number of Nodes Explored:	178	50,093	4,245,710	189	11,585	202,735
Average Time to Make a Move:	0.00790	2.6839	140.72	0.00645	0.43078	6.3444



Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Dr. Michael Scherger for his guidance, patience, and constant support throughout the duration of this project. I would also like to thank my thesis committee, Dr. Antonio Sanchez and Dr. Wendy Williams, for their encouragement, insightful comments, and hard questions as well as the rest of the Department of Computer Science for their support throughout my years in the Computer Science program.

References

- Stuart Russell and Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach (2nd ed.)*. Prentice Hall Press, Upper Saddle River, NJ, USA.
- Joel H. Gyllenskog. 1976. Konane as a vehicle for teaching AI. SIGART Bull. 56 (February 1976), 5-6.
- David Poole, Alan Mackworth, and Randy Goebel. 1997. *Computational Intelligence: A Logical Approach*. Oxford University Press, Oxford, UK.