Presidents Game Playing Agent

Nicholas Larus-Stone   Juan Perdomo   Matt Goldberg

December 10, 2015

1 Introduction

A description of the purpose, goals, and scope of your system or empirical investigation. You should include references to papers you read on which your project and any algorithms you used are based. Include a discussion of whether you adapted a published algorithm or devised a new one, the range of problems and issues you addressed, and the relation of these problems and issues to the techniques and ideas covered in the course.

This paper provides an in-depth investigation of a number of algorithms applied to the multiplayer imperfect information card game Presidents. We first applied two common multiplayer game playing algorithms to this problem: paranoid and max-n [**?**]. Neither algorithm inherently handles uncertainty, so we had to develop our own methods for dealing with imperfect information. We also investigated more complex algorithms that are better at handling uncertainty: Monte Carlo Tree Search [**?**] and reinforcement learning [**?**].

Investigating the best way to make an AI agent that plays the game Presidents. Chose this because of multiplayer games and uncertainty. Sturtevant was primary expert on this, see his thesis here. Also see MCTS paper. Adapted published algorithms, with inventions based on how to deal with uncertainty (sampling and heuristic). Built a lot on ideas from two player game in course, but more complicated by expanding them.

2 Background

Research into game playing has been a main focus of members of the artificial intelligence community since the mid 20th century. As mentioned in lecture by Prof. Rush, computer scientists worked on topics such as chess AI from 1959 [1]. Although research into game playing algorithms might seem like an inefficient investment of resources by talented researchers, breakthroughs in the algorithms and techniques designed for simple games such as chess or Go have often had repercussions in other fields that have more direct impact on society. At their essence, these algorithms work to find solutions to problems in which players compete to maximize their rewards under certain constraints. In the case of Presidents, however, there is the additional complication of having imperfect information as each individual agent must operate without knowing the hands of his opponents. Developments in game playing algorithms for problems like Presidents therefore can provide insights into different real life problems such as auctions, elections, AND X

[1] Rush, Adversarial Search and Games

in which a number of agents compete to achieve certain goals using incomplete information.

The first two algorithms discussed in this paper, Paranoid and Max-N are essentially extensions of the traditional MiniMax algorithm developed by John von Neumann in 1928 [2]. Since its origins, however, several optimizations have been introduced such as the use of Alpha Beta Pruning to reduce the size of the search space and techniques to deal with imperfect information. Moreover, research into techniques to deal with uncertainty has become a hall mark of 21st century computer science and artificial intelligence. Conversely to Paranoid and Minimax, Monte Carlo Tree Search has been developed more recently, first appearing in a paper by Remi Coulomb in 2006. Although Monte Carlo methods have their origins in the 1940s and the Manhattan Project, the creation of the MCTS algorithm for games is a 21st century discovery.

http://www.theoremoftheday.org/Docs/Kjeldsen.pdf, Gray

3 Related Work

During our research into game playing methods for Presidents. We found no algorithms that dealt specifically with the game of presidents, however, we did find research in related fields such as card playing AI agents and agents for games of incomplete information. Multiplayer Games, Algorithms and Approaches by Sturtevant was was particularly helpful to learn about extensions of Minimax to multi agent games. Koller's work on imperfect information provided insights into adaptations of traditional game playing agents for scenarios in which there is imperfect information. Lastly, papers by Fujita and Browne on Reinforcement Learning agents and Monte Carlo Tree Search methods served as guides that led our initial development of the algorithms used.

For instance, [**?**].

4 Overview of Algorithms

Sampling based on cards that haven't been played

4.1 Paranoid

The first algorithm we implemented was a paranoid agent. The paranoid algorithm is a modified minimax algorithm which assumes that all the other players in the game are collaborating against it. This reduces the multiplayer game to a two player one in which traditional minimax can be applied. At a leaf node (a terminal state), the score is determined by

Modified minimax, where leaf/terminal nodes are a tuple of values representing the scores (or a heuristic estimate of those scores). Calculate intermediate values by subtracting all scores from first player score. Too deep to expand full game tree, so terminate at terminal node or after a certain depth or a certain number of nodes has been expanded (keep track of nodes expanded for pruning).

[2]Kjeldsen, John von Neumann's Conception of the Minimax Theorem: A Journey Through Different Mathematical Contexts

4.1.1 Pruning

Standard a-b pruning works here

4.2 Max-n

4.3 fasdfa

4.4 Monte Carlo Tree Search

A clear specification of the algorithm(s) you used and a description of the main data structures in the implementation. Include a discussion of any details of the algorithm that were not in the published paper(s) that formed the basis of your implementation. A reader should be able to reconstruct and verify your work from reading your paper.

The MCTS algorithm is divided into 4 key stages: selection, expansion, simulation, and back-propagation. In selection, a node is chosen starting from the root of the game tree by recursively selecting the most promising child node until a leaf in the tree is reached. After a node has been chosen, the game is played out using a default policy until a result is reached. Lastly, in backpropagation, the result of the playout is incorporated recursively into each node located in the path from the root to the selected child.

Algorithm 1 Pseudocode for Monte Carlo Tree Search

```
procedure MCTS(state)
    root ← mctsNode(state)
    while budget do
        nextNode ← selection(root)
        $result \leftarrow simulation(nextNode)$
        backpropagate(nextNode, result)
    end while
    action ← bestChild(root)
    return action
end procedure
```

In the context of Presidents, a separate tree data structure to the state formalism had to be developed in order to implement MCTS. Each node in the tree contained variables describing the number of times each node had been visited as well as the score of the node. Moreover, each node contained a list of hands for each player and the id of the player whose turn it was to play. Since the hands of the other players in the game are unknown, when instantiating the tree, hands had to be sampled for the other agents in the game based on the sizes of each player's hand and the set of cards yet to be played. Each child node corresponded to the actions the player whose turn it was could make based on his hand and the top card on the deck. Scores were defined as the finishing position of the agent in the game. Since higher scores are better, we took the inverse of the finishing rank to indicate the score.

In the selection stage, we used the UCT algorithm developed by Kocsis and Szepesvari discussed by Browne in his paper on MCTS methods. It takes into account exploration and exploita-

	Score
Approach 1	
Approach 2	

Table 1: Description of the results.

tion of paths down the tree by weighing the score of each node and the number of times it and its parent had been visited. Once a node is selected, the tree is expanded to include the children of that node and one of the child nodes is selected at random. In simulation, a game was played out based one the hands of each player at that node. Finally, in the backpropagragation stage the normalized score (raw score divided by number of visits) is sent up the tree to the node updating the values of each node along the way. This is run until a predetermined budget is used up, which in this case we chose to be time. After the time ran out, the agent made a move based on the child of the action with the best score.

5 Body 2

Unsure if we want this?

6 Experiments

Analysis, evaluation, and critique of the algorithm and your implementation. Include a description of the testing data you used and a discussion of examples that illustrate major features of your system. Testing is a critical part of system construction, and the scope of your testing will be an important component in our evaluation. Discuss what you learned from the implementation.

Can test on a number of different axes–time to run, nodes expanded, how it does playing against dummy agents, how it does playing against other algorithms, how it does playing against itself, number of times we sample.

6.1 Methods and Models

Wrote dummy agents to play lowest legal card–naive algorithm. Baseline to test against that for how good our agents were.

6.2 Results

For algorithm-comparison projects: a section reporting empirical comparison results preferably presented graphically.

Table 1: showing time to run on 50 games, average score
Graph 1: Paranoid (5 trials, include error bars)
Bar graph showing:
Average score for Dummy vs Dummies
Paranoid vs Dummies (sample once, 500 nodes)
Paranoid vs Dummies (sample 5 times, 500 nodes)

Paranoid vs Dummies (sample 25 times, 500 nodes)
Paranoid vs Dummies (sample 25 times, 1500 nodes)
Graph 2: Max-n
Graph 3: MCTS

6.3 Discussion

Overall, there are optimizations to be made in the approach to solving the constraint of imperfect information. As of now, the sampling of hands is done uniformly for each player. However, in the game of presidents, since the president and the asshole switch hands it is more likely that the asshole have a worse hand than the secretary,who didn't switch any cards, and that the secretary in turn have a worse hand than the president. This issue could be approached in several ways.

The first way would be to estimate the average hand strength of each respective player rank at each point in the game and then sample hands for that player based on that average. After playing a large number of games, we could infer the distribution for the hands of each depending on how many cards they have and then sample from that distribution. One major issue with this approach is that it is blind to past behavior of each agent. An agent could have been playing high cards all through the game and then be left with just low cards. However, the distribution for the player's hand at that point in the game could still be very high giving the player an unrealistic set of cards.

The second way to approach the problem would be to create a filtering algorithm that estimates the current hand strength based on the history of play of the player. Each card played would be considered an emission and the hidden states would be the players hands after each time it plays. The trouble in this approach would be estimating the transition probabilities between hidden states as well as the emission probabilities for the played cards. Emissions are not independent for example of the top card on the deck. Also, it is not clear whether playing a high card means your next state is weaker or stronger. If for example, an agent plays a 2 and gets to go again (this is a rule of the game), he might: 1) have another 2 and a low card which menas he has the chance to win, or 2) he might be left with a 3 and be forced to pass for the next rounds.

In terms of improving MCTS, there is a lot of work that can be done refining the different parameter values. The performance of the algorithm is dependent on how well it selects nodes in the tree and then expands them. This exploration is regulated by means of a constant that can be altered. Moreover, the time budged can be changed to that it expands more of the tree. By optimizing the values for these two parameters, one can increase the efficiency of the algorithm and the quality of the actions taken.

A Program Trace

Appendix 1 A trace of the program showing how it handles key examples or some other demonstration of the program in action.

B System Description

Appendix 2 A clear description of how to use your system and how to generate the output you discussed in the write-up and the example transcript in Appendix 1. N.B.: The teaching staff must be able to run your system.

C Group Makeup

- Matt Goldberg - Wrote the state, agent, and game modules that provide the necessary formalisms and rules to encode the game of Presidents. Moreover, implemented the Max-N algorithm.
- Nicholas Larus-Stone - Implemented the repeated games function that allows a list of agents to play multiple games. Designed the Paranoid algorithm.
- Juan Perdomo- Implemented the formalisms necessary to implement the MCTS algorithm as well as implemented the algorithm itself.