A Monte Carlo Tree Search Player for Birds of a Feather Solitaire

Christian Roberson, Katarina Sperduto
Florida Southern College
croberson@flsouthern.edu, ksperduto@mocs.flsouthern.edu

Abstract

Artificial intelligence in games serves as an excellent platform for facilitating collaborative research with undergraduates. This paper explores several aspects of a research challenge proposed for a newly-developed variant of a solitaire game. We present multiple classes of game states that can be identified as solvable or unsolvable. We present a heuristic for quickly finding goal states in a game state search tree. Finally, we introduce a Monte Carlo Tree Search-based player for the solitaire variant that can win almost any solvable starting deal efficiently.

Introduction

Games are a good way for students to get involved in the intricacies of scholarly research. Games with perfect information are especially nice because they allow students to focus on the rules and strategy of the game without having to worry about the inherent uncertainty that comes with imperfect information (Rosenthal 1981). One such game that has been developed for research is Birds of a Feather Solitaire. Much like other variants, this version of solitaire requires making a series of moves to combine various piles of cards into a single stack. This game is particularly interesting because while most initial configurations of the game state are solvable, there are many ways to end up in an unsolvable state and lose the game. There are also several distinct characteristics of game states that could potentially be used to determine if a state is solvable.

This paper is organized as follows. In the Background section, we provide an overview of the Birds of a Feather game and highlight relevant search algorithms including heuristic search and Monte Carlo Tree Search. In the Game State Solvability Section, we outline three different characteristics of game states that can be used to identify unsolvable states. In the Player section, we describe the structure of our AI player for Birds of a Feather. The Results section contains experimental tests and validation for our developed tools. Finally, we provide some conclusions and possible avenues for future work.

Background

Birds of a Feather Solitaire

Birds of a Feather is a variant of a classic solitaire game proposed as a research challenge for facilitating research experiences for undergraduates (Neller 2018). It is played with a standard 52-card deck, however out of the 52 cards only 16 will be used. Once shuffled, the player deals 16 cards out face-up in a 4-by-4 grid as shown in Figure 1. Each card (or bird) can be moved onto other cards in the grid to form a stack of cards (a flock).

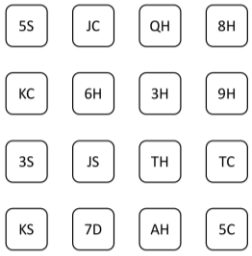

Figure 1: Example initial game state

In this game each position in the grid is considered a stack of cards, so in the initial layout each stack or flock has a size of one. In order to move one stack on top of another, the following conditions must be met: (1) The two stacks must be located in the same row or column. (2) The top card of each stack must either have the same suit, same rank, or adjacent ranks. For this version, aces are considered to be low and are only adjacent in rank to twos. The score for a game state is determined by taking the square of the stack size of each remaining column and adding them together. For example, if there were a total of three stacks remaining: one of size 9, one of size 4, and one of size 3, the final score would be 106 (81 + 16 + 9). The ultimate goal is to move all 16 cards into a single stack, yielding a

maximum score of 16^2 or 256. For each initial configuration, or seed, it is highly probable there is a solution, but it is not guaranteed.

Heuristic Search

A common approach to finding solutions to games is to create a game tree representing the possible game states and moves and use a tree search algorithm to locate a goal state (Paul and Helmert 2016). Classic tree search algorithms, like Depth First Search (DFS), blindly expand nodes during the search process until either a goal node is found or all search options are exhausted (Chijindu 2012). This approach can be time consuming, especially if the search tree has a high branching factor, is likely to search many unnecessary nodes, and the solution it produces may not be the optimal solution. One possible improvement to an uninformed search is to use a heuristic function to help guide the order of node expansion. A heuristic function $h(n)$, takes a node n and returns a non-negative real number that is an estimate of the cost of the least-cost path from node n to a goal node. The function $h(n)$ is an admissible heuristic if $h(n)$ is always less than or equal to the actual cost of a lowest-cost path from node n to a goal. With a well-crafted heuristic it is likely that a solution can be found with significantly less nodes expanded than an uninformed search.

Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a directed search technique that has gained prominence in recent years and has been used with success for several types of games such as Go (Silver et al. 2016) and Kriegspiel (Ciancarini and Favini 2009). The basic algorithm involves an iterative construction of a search tree until some computational limit is achieved. (Browne et al. 2012). There are four steps performed during each iteration of MCTS: selection, expansion, simulation, and backpropagation. Figure 2 shows the structure of each MCTS phase and the search tree associated.

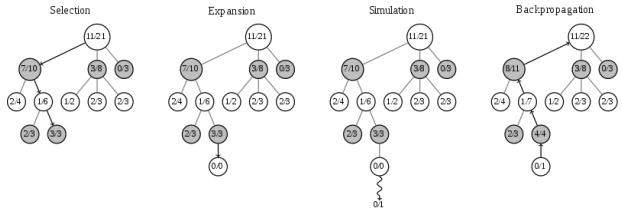

Figure 2: Monte Carlo Tree Search Steps[1]

The selection phase chooses the next node to expand by starting at the root node of the tree and recursively selecting optimal child nodes until a leaf node is reached. The optimal child node j is selected to maximize the Upper Confidence Bound for Trees (UCT), given as:

$$UCT = \bar{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

where $\bar{X}_j$ is the average reward for child j, n is the number of times the current (parent) node has been visited, n_j the number of times child j has been visited and $C_p > 0$ is a constant used to influence the exploration/exploitation balance. This ensures that the node with the maximum desired quality is chosen next.

The simulation phase begins when a node outside of the stored MCTS tree is selected during the selection process. The basic strategy for this phase is to make random moves until a terminal node is reached. In some variations a strategy is applied to determine node selections during the simulation.

The expansion phase grows the stored MCTS tree, typically by one node. A simple approach is to add the first node of the playout phase to the MCTS tree for each simulation that runs. This ensures the tree grows in areas selected during the selection phase because of their increased likelihood of being a strong move.

The backpropagation phase takes the result of the simulation phase and updates nodes along the selected path in the MCTS tree. A common approach is to simply record and track the average reward value by examining the ratio of games won to games played. A discounting factor may sometimes be applied to nodes as the result is propagated.

MCTS is used a lot for game algorithms for a few key reasons (Chaslot et al. 2008): it requires no tactical knowledge of the game, it has the ability to focus on more interesting nodes, and it is very easily adaptable. MCTS has been adapted for other variations of solitaire (Yan et al. 2005), but has not yet been applied to Birds of a Feather.

Game State Solvability

One focus of our research was to understand what makes a particular game state either solvable or unsolvable. We have identified several classes of states that classify as solvable or unsolvable.

Stranded States

A game state is considered stranded if it contains one or more stranded cards. A stranded card is a card that is the only card in both its row and its column, as long as there is more than one card remaining. A card in this state will never have another move available to it since cards can only be moved in their respective rows or columns. In Figure 3 the Jack of Spades is stranded because there are no cards in the same row or column to move to.

[1]https://commons.wikimedia.org/wiki/File:MCTS_(English)_-_Updated_2017-11-19.svg

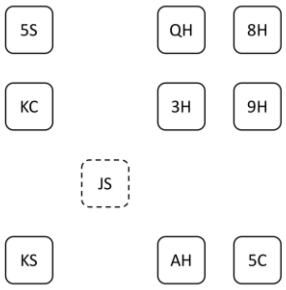

Figure 3: State with a stranded card

To determine if a card is stranded, we first create an undirected graph of all the theoretical moves possible in the current game state. A theoretical move is defined as any move that respects the position rules of the game while ignoring any rules about the value of the cards. In this case any two cards that occupy the same row or column would be a valid theoretical move. We then analyze the theoretical graph to see if there is more than one component. If there is, there are stranded cards and that game state is considered unsolvable.

Separated Flock States

In certain deals of Birds of a Feather, it is possible to end up with cards that cannot be moved onto other cards because they do not meet the suit or rank requirements of a legal move with any other cards in the grid. A move that is a legal move with respect to rank and suit, but without the same row or column constraint for the two cards is called a relaxed move. A single card with no relaxed moves is referred to as an odd bird. It is also possible to end up with multiple cards that are connected to each other, but none of the cards in this subgroup can be connected to the remaining cards. This is referred to as a separated flock. Odds birds are separated flocks of size one. In Figure 4, the Ace of Diamonds and the Seven of Diamonds are a separated flock because no other cards have relaxed moves (Diamonds, Twos, Sixes, or Eights) that connect to either of them. In this case it is impossible to combine those cards with the remaining cards in this state.

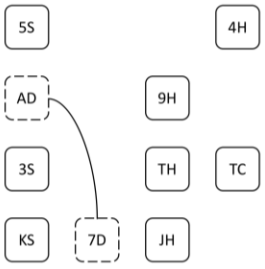

Figure 4: State with a separated flock

To determine if a state contains a separated flock, we create a graph of the relaxed moves in the current state. If the relaxed graph contains more than one component, there is a separated flock in the state and it is considered unsolvable.

Lynchpin Cards

Another interesting group of states are related to cards that occupy critical positions on the board, referred to as lynchpin cards. A lynchpin card is any card in a game state that if moved would lead to at least one card being stranded. For example, in Figure 5 moving the Jack of Spades means cards in the column cannot be moved into other rows anymore and cards in the row can't be moved to another column. Any move that moves a lynchpin card is unsolvable.

If there are multiple lynchpin cards whose only relaxed move is to move to another lynchpin card, that state is considered unsolvable. It is also possible to find a lynchpin in an impossible structure. If a lynchpin is connected to two or more cards that have a relaxed graph degree of one, it is impossible to move all of those cards onto the lynchpin. Given that a lynchpin cannot be moved, if any one of the multiple connected cards is moved to cover the lynchpin, the remaining cards would lose that connection in the graph and would therefore have a relaxed move degree of zero. These cases are also considered unsolvable as well.

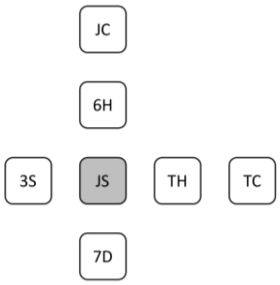

Figure 5: State with a lynchpin

A Birds of a Feather AI Player

Another focus of our research was to develop an effective player for Birds of a Feather. Given a game state, we first applied a simple two-step lookahead to the game tree to see if any goal states were present. If so, we would select the first child node that moved toward an identified goal state. To identify goal states, we developed a solvability checker to run each candidate state through for evaluation. If the lookahead was unable to find a goal state, we would then determine our move by applying a variation of Monte Carlo Tree Search.

Solvability Checker

Our solvability checker function accepts Birds of a Feather game states as input and classifies them into one of three

categories: solvable, unsolvable, or unknown. The bulk of the solvability checker's work for identifying unsolvable cases comes from analyzing and determining if a game state meets one of the discovered classes of unsolvable states. Any state that is identified as containing stranded cards, separated flocks, or unsolvable lynchpin cards is returned as unsolvable. Any two-card case that does not fall into these categories is solvable.

We also discovered several identifiable subgroups for three-card, four-card, and five-card cases that can be used to identify solvable or unsolvable states. Each of these subgroups was coded into the checker to identify additional states. The specifics of these configurations are not included in this paper.

Monte Carlo Tree Search

For states in which a goal node is not found using the two-step lookahead, we apply a variation on classic Monte Carlo Tree Search to determine which move to play.

During the selection phase, our algorithm explicitly requires each child of a node to be explored at least once before moving to exploiting the best score when choosing the next node to process. This change helps to ensure all nodes are at least sampled during the MCTS process. We also leverage the solvability checker during the selection phase. If a node is classified as solvable, we select that node automatically regardless of UCT score.

For the expansion phase, whenever a new node is explored, all of the children of that node are generated and added to the MCTS tree. This is primarily a limitation of the codebase we used for the project. One of the new children would then be selected at random for the simulation phase.

We also developed a pruning technique to apply to the MCTS tree as it is developed. Each node is pre-processed through the solvability checker before it is added to the MCTS tree. If the node is classified as unsolvable, it is thrown away and not added to the tree. If it is either solvable or unknown it is expanded normally. This ensures that iterations in the algorithm are not wasted on known unsolvable nodes in the tree.

In the traditional simulation phase, moves are selected at random from the available child nodes of the current state during simulation of the game. We instead choose to select moves during the simulation phase using a heuristic for evaluating potential moves.

Player Heuristic

To better select moves that would lead us to winning states, we developed a heuristic for evaluating and ranking potential moves for consideration. While many factors were considered, ultimately two factors provided the most influence on the quality of a game state: the current score of the game state and the number of legal moves in the game state. Our heuristic is represented by the following formula:

$$h(s) = \sum_{i=0}^{15} w_0 size(s[i])^2 + w_1 L(s)$$

where s represents the game state to examine, $s[i]$ represents the stack of cards at position i in the grid of cards, the $size$ function determine the size of a stack of cards, w_0 and w_1 represent the weights of both factors, the summation represents the score of state s, and $L(s)$ represents the number of legal moves in state s. For our heuristic, higher values are considered more valuable states.

Results

In this section we present findings for the various aspects of our Birds of a Feather Solitaire research. First, we analyze our heuristic for its efficiency. Next, we provide details of our solvability checker's coverage of known test data. Finally, we show how our player, which uses both the heuristic and solvability checker, performs against a variety of possible starting seeds of the game.

Heuristic

To tune our heuristic, we applied a heuristic search algorithm to the search tree for the first 10,000 seeds of the dataset. The weight parameter for the state's score was fixed to 1.0 and the weight for the number of legal moves was tuned to find the combination that optimized for the lowest average number of nodes expanded across the test seeds.

Moves Weight	Nodes Exp.
1.0	387.43
1.5	197.80
2.0	117.84
2.5	102.85
3.0	128.68
3.5	185.96

Table 1: Heuristic Evaluation

The optimal value for weighting the legal moves factor was 2.5, where we averaged just under 103 nodes expanded per seed. In many cases the number of nodes expanded was well under the average, with several more difficult cases driving the average up. Our heuristic performs much better than a traditional depth-first search approach, with DFS averaging 5,274.23 nodes expanded per seed on the same dataset. For states early in the game with many stacks and a low score, the driving factor in node selection for expansion is the number of legal moves. For late-game

nodes, the score is significantly more important for node selection than the number of legal moves. Our heuristic captures this dynamic.

Solvability

Included in the research challenge was a dataset for 10,000 different initial game states. While not every possible state for each game was included, the set focused on interesting states. An interesting state is defined as a state that includes both solvable and unsolvable children. Each provided state was classified as solvable or unsolvable. To validate the different classes of game states identified in our research, we ran every state in the provided dataset against our solvability checker to determine its effectiveness.

State Class	# of States	% of Unsolvable
Stranded Cards	95,405	14.50%
Separated Flock	147,330	22.39%
Lynchpin Issues	192,339	29.24%
Combined Total	249,534	37.93%
Total Unsolvable	657,888	100.00%

Table 2: Unsolvable state analysis

Table 2 provides a breakdown of unsolvable states from the provided dataset. For each class of states, both the total number of states identified by our solvability checker and what percentage of the overall number of unsolvable states this represents are listed. It is worth nothing that the combined total of all these states represents a smaller total that the sum of the individual classes. This is because there are some states that are represented in multiple classes. Overall our solvability checker was able to correctly identify almost 40% of the unsolvable states with no false positives.

Player

To test our player, we played games against a variety of seeds. For each experimental run, the player runs 40 games against each of the 10,000 seeds. Each seed is first checked to determine if it is solvable. Any unsolvable seeds are then skipped. For the test dataset there were 24 unsolvable seeds, leaving 399,040 games played per run of the system.

Iterations	Player Win %
100	44.26%
200	77.20%
300	94.55%
400	96.78%
500	98.09%
1000	99.41%
2000	99.75%
4000	99.85%
8000	99.88%

Table 3: Player win rates

Table 3 provides the overall win rates for the player against our dataset. The player was run several times with different limits on the number of iterations for Monte Carlo Tree Search simulation. It is notable that our player very quickly improves its win rate to almost 95% by 300 iterations per turn. After that, it is a slow but steady increase towards 100%. As a limit test we ran our player for 8,000 iterations per turn and only lost 480 games, resulting in a 99.88% win rate.

Conclusions and Future Work

There are several conclusions that can be drawn from our work on Birds of a Feather Solitaire. First, using a heuristic search for analyzing the search tree of a given starting seed of the game provides significant improvement in terms of number of nodes expanded for solution search. In our experiments we found that a heuristic using the state score and the number of possible legal moves as factors provided the best results.

Second, there are several characteristics of a game state that can be used to determine whether it is solvable or not. For our research we focused on unsolvable states and identified three classes of states that cannot be solved: states with stranded cards, states with separated flocks (a subset of cards that can only be moved onto each other), and lynchpin cards (critical cards that cannot be moved without stranding other cards) that are put into impossible situations to resolve legally. By developing these characteristics into a solvability checker that can evaluate states, it is easier to develop an effective player for this game.

Finally, we present a strategy for developing an AI player capable of winning most starting arrangements of Birds of a Feather. Combining simple lookahead, Monte Carlo Tree Search, and contextual search tree pruning using solvability data, our player is able to achieve win rates in the mid-90% range using only a few hundred iterations in Monte Carlo Tree Search simulation.

There are some areas open for future expansion of our work, especially in the area of determining solvability of a state. One area of interest is the examination of difficult starting configurations of Birds of a Feather. While our player was able to successfully solve almost every solvable state in the dataset, there were some states that our player was not particularly effective at. We would like to investigate these cases further to understand why these pose such a challenge and look for improvements to be made to our player. In addition, our solvability work focused mostly on determining unsolvable cases. There is potential to determine characteristics that make a solution solvable, which would expand the number of states that could be correctly identified by our solvability checker.

References

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games 4*(1): 1-43.

Chaslot, G.; Bakkes, S.; Szita, I.; and Spronck, P. 2008. Monte-Carlo Tree Search: A New Framework for Game AI. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 216-217. Palo Alto, Calif.: AAAI Press.

Chijindu, E. V. 2012. Search in Artificial Intelligence Problem Solving. *African Journal of Computing & ICT 5*(5): 37-42.

Ciancarini, P., and Favini, G. P. 2009. Monte Carlo Tree Search Techniques in the Game of Kriegspiel. In *Proceedings of the Twenty-First International Joint Conferences on Artificial Intelligence*, 474-479. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence, Inc.

Neller, T. W. 2016. AI Education: Birds of a Feather. *AI Matters* 2(4): 7-8.

Paul, G., and Helmert, M. 2016. Optimal Solitaire Game Solutions Using A* Search and Deadlock Analysis. In *Proceedings of the Ninth Annual Symposium on Combinatorial Search*, 135-136. Palo Alto, Calif.: AAAI Press.

Rosenthal, R. W. 1981. Games of Perfect Information, Predatory Pricing and the Chain-Store Paradox. *Journal of Economic Theory* 25(1): 92-100.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; and Dieleman, S. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529: 484-489.

Yan, X.; Diaconis, P.; Rusmevichientong, P.; and Roy, B. V. 2005. Solitaire: Man Versus Machine. In *Proceedings of the Eighteenth Advances in Neural Information Processing Systems*, 1553-1560. Cambridge, Mass.: MIT Press.