FreeCell Solitaire Optimization

Chapp Brown
Kylie Beasley

University of North Carolina Wilmington

Abstract

Multiple algorithms have been developed to attempt to solve FreeCell Solitaire. It is impossible to solve every deal of FreeCell, but the goal of these algorithms is to attempt to solve as many deals as possible. These algorithms include breadth first search, depth first search, modified versions of depth first search, and evolving heuristics. This paper explores how these algorithms perform at solving FreeCell using benchmarks such as how much time and memory they require, as well as the percentage of deals that these algorithms solve.

Key Words: Search Tree, Breadth First Search, Depth First Search, Modified Search, optimization, Genetic Algorithms

1.Introduction:

In a game of FreeCell solitaire, there are 8 columns, 4 foundations spaces, and 4 free cells. At the beginning of each game, columns 1 through 4 are dealt 7 cards each; columns 5 through 8 are dealt 6 cards each. The four foundation spaces are used to store the cards from each suit (Hearts, Diamonds, Spades and Clubs). Cards must be placed in the foundation of their suit in ascending order from the ace to the king, but once a card has been placed in the foundation it cannot be removed. The free cells start empty and can hold a single card of any suit as long as the cell is empty. Cards can be placed in the columns onto cards that are of the opposite suit color and one value higher. Stacks of cards can be moved as long as the stack is alternating suit colors and is stacked in ascending order. To move a stack of cards of size n, the number of open free cells and the number of empty columns must add up to n-1. The objective of the game is to move all 52 cards into the foundation spaces. To complete the objective the cards must be organized in a way that allows the cards to be placed into the foundations, thus solving the game.

2. Methods

The following methods have been implemented for solving the most games of the original 32,000 FreeCell games that were released by Microsoft, breadth first search, depth first search, stage deepening, and genetic algorithms.

2.1 Breadth First Search

A Breadth-first Search (BFS) approach to solving FreeCell Solitaire would guarantee a solution to a deal in the fewest possible moves. A Breadth-first Search uses a search tree. The root node of the tree represents the initial board state. The search looks for all the possible board states that are one move away from the initial state. These board states become leaf nodes. From there, the algorithm recursively repeats the same process for each leaf node, where each leaf node becomes the root. It repeats this process until a solution is found, it runs out of memory, or it runs out of moves to make and the deal is proved to be unsolvable. Since the algorithm creates all possible

board states from the least moves away to the most moves from the initial board state, breadth-first search is guaranteed to return the shortest path to the solution, or a solution with the fewest number of moves. However, when looking just a few moves ahead, the memory begins to fill up extremely quickly. Table 1 shows the estimate of the size of the search tree on board #14, looking up to 16 moves ahead with increased numbers of trials for increased accuracy. Therefore, Breadth-first Search is not a feasible algorithm for solving any of the original FreeCell deals.

Moves	Tree size
1	10
2	97
3	958
4	9,670
5	80,709
6	521,687
7	3,148,528
8	19,229,400
9	102,380,038
10	629,428,522
11	3,683,996,355
12	29,217,980,095
13	2.0209E+11
14	1.35773E+12
15	6.35592E+12
16	8.99903E+13

Table 1

2.2 Depth first Search

A Depth-first Search is similar to a Breadth-first Search in that it utilizes a search tree and recursion. A depth-first search starts at the root node, and calculates a single possible first move. From there it calculates a possible move from that state, and so on until it finds a solution, runs out of memory, or runs out of moves to make. At this point, it utilizes a technique called backtracking. It backtracks up the tree one move, looks for a different move from that state, and continues. Since this is a technique that will also calculate every single possible board state, and does not place any value on “good” board states, it should not be surprising that this method

runs out of memory just as quickly. One difference between breath-first search and depth-first search is the ability to apply a depth-bound. A depth-bound of six would keep the depth-first search from venturing down the search tree any farther than six moves from the root node. The problem is that depth-first search runs out of memory for depth bounds as small as seven. Since none of the original 32000 deals of FreeCell can be solved in six moves, depth-first search is also not a feasible algorithm for solving FreeCell.

2.3 Stage Deepening

Heineman's Staged Deepening (HSD) is the second best algorithm known for solving a large number of FreeCell deals [1]. It is a modified version of depth-first search, which takes advantage of a few unique properties of FreeCell. The first property is that there are usually multiple ways to arrive at the solution for a given FreeCell deal. Secondly, there are many cases where moving a card is irreversible. For example, once a card is moved to the foundation, it cannot be returned to the board. If a card is moved off a column, it cannot be returned to that column unless the top card on that column is of the opposite suit color and has a value that is greater by one. These two properties allow the HSD algorithm to periodically throw out old board states, significantly reducing the storage requirements of the algorithm.

Heineman's algorithm also employs a heuristic that evaluates board states and returns an integer that represents how "good" that board state is, with smaller numbers representing board states the heuristic judges to be closer to a solution. Storing these integers instead of the entire board state further reduces the storage requirements of the algorithm. The heuristic is as follows; for each foundation pile, locate within the columns the next card that should be placed there, and count the cards found on top of it. The sum of this count for each foundation is what the heuristic returns. This number is multiplied by 2 if there are no available FreeCells or there are empty foundation piles.

Heineman's Staged Deepening actually works by performing a depth-first search with a depth-bound of six. The algorithm then uses its heuristic to evaluate all of the board states exactly six moves away from the initial board state. It takes the board state with the best score and does another depth-first search with a depth-bound of six from that state. It repeats this algorithm, throwing out all stored board states as the number of stored board states approaches some set limit. The algorithm only holds onto enough data to be able to perform backtracking if necessary. This algorithm does have two disadvantages when compared to the breadth-first search and the depth-first search. First, it is possible to get stuck in a loop since the algorithm does not keep track of board states already visited. Secondly, it is possible to throw out an important state that is needed to reach the solution. These two weaknesses result in the algorithm occasionally erroneously reporting deals as unsolvable, and occasionally it gets stuck in loops, meaning it runs for too long without finding a solution. The results of running HSD algorithm on Microsoft's original 32,000 deals are reported in the following table.

Result	Number FreeCell states	Percentage
Solved	30,859	96.4%
No Solution	1,110	3.5%
Search took too long	31	0.1%

Table 2

2.4 Genetic Algorithm and Evolving Heuristics

A genetic algorithm mimics the process of natural selection. It usually employs methods like mutation, crossover, reproduction and selection on a population of solutions to a problem.

Elyasaf, Hauptman and Sipper's Genetic Algorithm evolves heuristics to be implemented by the Staged Deepening algorithm, resulting in the best FreeCell solver to date. They determined that in order to improve upon the HSD algorithm, additional heuristics were necessary. Combining heuristics is an extremely difficult problem, so they turned to a genetic algorithm to do this for them. They combined these heuristics linearly, transforming them into a genome. The fitness of these individuals is represented by

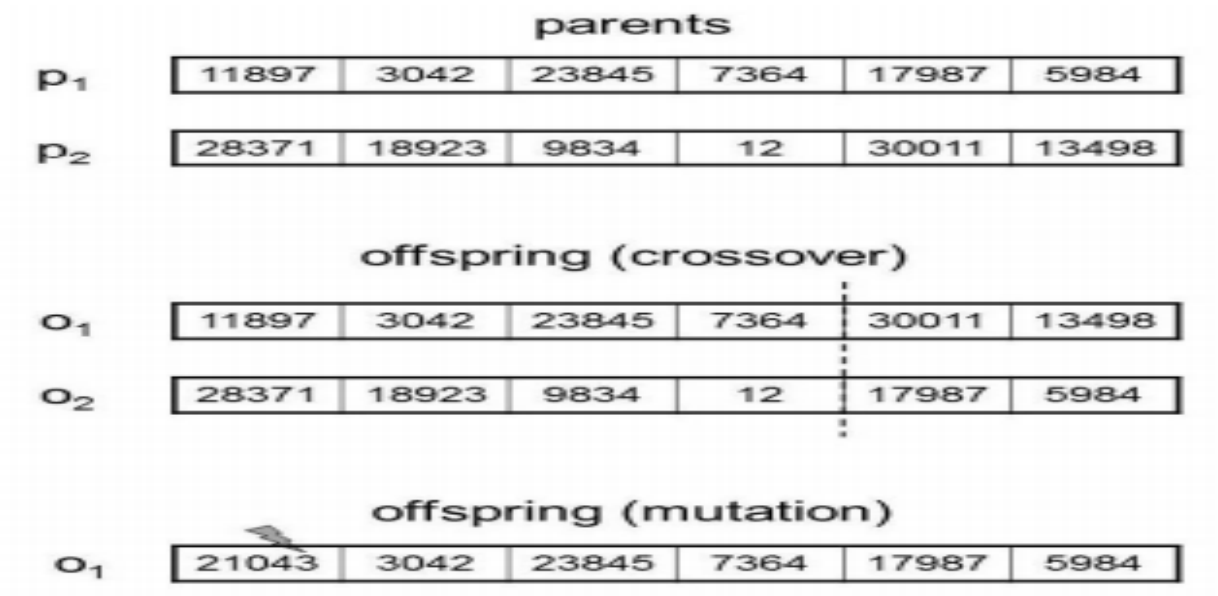
$$H = \sum_{i=1}^{9} w_i h_i,$$

where w represents the weight and h describes the value of the heuristic at position i. The following table shows the heuristics they used for their genome.

R=*HSDH*	Heineman's staged deepening heuristic
R=*NumberWellPlaced*	Number of well-placed cards in cascade piles
R=*NumCardsNotAtFoundations*	Number of cards not at foundation piles
R=*FreeCells*	Number of free FreeCells and cascades
R=*DifferenceFromTop*	Average value of top cards in cascades minus average value of top cards in foundation piles
R=*LowestHomeCard*	Highest possible card value minus lowest card value in foundation piles
R=*HighestHomeCard*	Highest card value in foundation piles
R=*DifferenceHome*	Highest card value in foundation piles minus lowest one
R=*SumOfBottomCards*	Highest possible card value multiplied by number of suites, minus sum of cascades' bottom card

Mutation in this circumstance would involve changing the value of the number on the genome with some low chance, and replacing it with a random number in the range from 0 to 1. They used single-point crossover to exchange two values from two parents, keeping them in the same location on the genome.

Elyasaf, Hauptman and Sipper ran into problems evolving solutions to random deals, where individuals that were good at solving specific deals were not good at solving other deals. They turned to coevolution, where a population of solutions evolves side-by-side with a population of problems. In this case, the population of problems became sets of deals of size six. This population of deals evolved once for every five generations of solutions. In this scenario, the solvers were forced to evolve to be good at solving not just specific deals, but multitudes of deals with varying degrees of difficulty. The following image shows evolution in the population of problems.



These numbers represent the deal number for each deal in the set of six, where the set represents one problem in the population.

0.09	0.01	0	0.77	0.01	0.08	0.01	0.01	

The above image shows the genome of the best solver they received from their genetic algorithm.

3. Comparing Algorithms

The following table represents the best two algorithm-based solvers versus the 3 players with the most games played on www.freecell.net. This website has been active for two decades and the players have each logged more results than the 32,000 deals the algorithms tackled. The table shows that the best solution from the genetic algorithm is one of the best solvers. If sorted by win rate, only 8 players on the site with over 32,000 games played have a higher win-rate than the GA's solver. None of the 3 players with the most games played have a better win-rate. Finally, GA's time to finish is absolutely miniscule in comparison to the average amount of time it takes players or the HSDH to solve deals.

Name	Deals played	Time	Solved
sugar357	147,219	241	97.61%
volwin	146,380	190	96.00%
caralina	146,224	68	66.40%
BFS	32,000	NS	0%
DFS(with bound)	32,000	NS	0%

HSDH	32,000	44	96.4%
GA-FreeCell	32,000	3	98.36%

Even though the genetic algorithm only succeeded in solving 2% more of the 32,000 deals, it should be remembered that these 2% are some of the most difficult deals to solve.

Table 5

Heuristic	Nodes	Time	Length	Solved
	unsolved problems penalized			
HSDH	75,713,179	709	4,680	30,859
GA-FreeCell	16,626,567	150	1,132	31,475
	unsolved problems excluded			
HSDH	1,780,216	44.45	255	30,859
GA-FreeCell	230,345	2.95	151	31,475

Finally, table 5 shows the totals for time and nodes used to solve the original 32,000 Microsoft deals. It also shows us the percentages of reduction from the HSDH to the GA solver. Path length was reduced by 41%, time by 93% and nodes searched by 87%.

4. Conclusion

This paper examined several different methods of obtaining solutions for deals of FreeCell Solitaire. First, the blind searches were examined. Breadth-first search would provide the solution that is the fewest number of moves away. Unfortunately, the search tree grows too rapidly for this method to be feasible. Next, Depth-first search was examined. This method has a similar problem with extremely large search trees. Placing a depth-bound can keep the search tree small enough to be held in memory, but FreeCell usually cannot be solved in a small enough sequence of moves for this method to work either. Heineman took advantage of several unique characteristics of FreeCell to implement a modified depth-first search he called Staged Deepening. This method throws out old board states as we approach an arbitrary cap, and succeeded in solving over 96% of Microsoft's original 32,000 FreeCell deals. Finally, Elyasaf, Hauptman and Sipper used a genetic algorithm to combine and evolve a set of heuristics to beat Heineman's single staged deepening heuristic. They succeeded in massively reducing the search trees, time required, and increased the win-rate by successfully developing a solver to solve some of the most difficult deals.

5. References:

[1] Heineman, G. (2009, January 17). January Column: Algorithm to Solve FreeCell Solitaire Games. Retrieved April 6, 2015, from

http://broadcast.oreilly.com/2009/01/january-column-graph-algorithm.html

[2] Mol, M. (2015, February 13). Deal cards for FreeCell. Retrieved April 6, 2015, from http://rosettacode.org/wiki/Deal_cards_for_FreeCell

[3] Rules to Freecell. (n.d.). Retrieved April 6, 2015, from http://www.freecell.org/rules.html

[4] Elyasaf, A., Hauptman, A., & Sipper, M. (2011). GA-FreeCell: Evolving Solvers for the Game of FreeCell. Retrieved April 10, 2015, from http://www.genetic-programming.org/hc2011/06-Elyasaf-Hauptmann-Sipper/Elyasaf-Hauptmann-Sipper-Paper.pdf