Helping Human Play Freecell

University of Kent
Author: Chris Chan (cksc2@kent.ac.uk)

ABSTRACT

This report aims to describe the basic problem facing computer programs when attempting to write an algorithm which mimic the behaviour of human solving a given Freecell and the method that can be explored in solving this problem. Different Artificial Intelligence techniques are evaluated in their suitability at solving this problem. After analysing my solution to the problem future direction for research is discussed paying particular attention to researches involving human participants.

I. BACKGROUND

Freecell is a solitary card game invented by Paul Alfille. The board for the game consists of 4 foundation piles that are empty in the board's initial state, 4 Freecells providing a storage area for one card in each cell, and 8 tableau piles build down by alternate colour. The aim of the game is to move every card from the tableau piles into the foundation piles in ascending order (from Ace to King), each foundation having a different suit. The complete game rule can be found in the Freecell tutorial [Keller]. Due to the game's popularity, most modern operating system will have an implementation of this game. In aiding players in solving the Freecell, many computer implementation of Freecell contains a Freecell solver which can be used at any stage of the game to complete the Freecell if a solution is available.

Freecell has several trails that are attractive for Human – Artificial Intelligence analysis. FreeCell is an open solitaire, meaning that all of the cards are dealt out face-up at the start. The player will have the exact same view of the problem as the solver. The effect of each move is deterministic meaning that no probability is involved. Since there is only a limited combination of Freecell deals, recreation is possible if the particular deal and the sequence of play are described.

Freecell cannot be described as an NP-complete problem since there is a finite combination of Freecell deals. However, there is no algorithm for discovering the shortest sequence of play on the fly that does not involve the use of brute force searches. A comparison of every possible move can yield the shortest sequence of play but at an expense of CPU computing time. Repeating the process for every user made modification to the board would not lead to an interactive experience for the user. A brute force approach does not enhance our understanding in the area of Artificial Intelligence and their relations to the action of a human player. The motivation of this project is to inform and explore the aforementioned relationship.

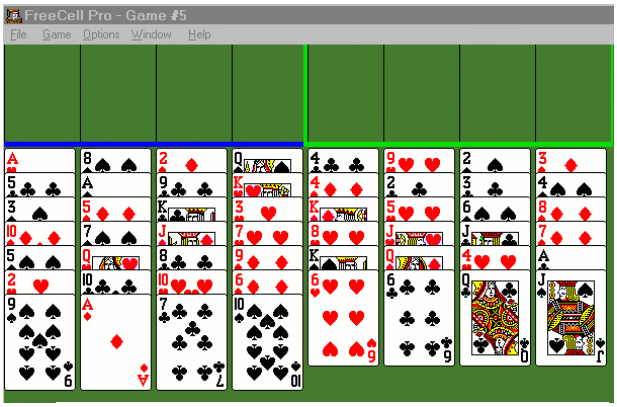

Figure 1. Example of a typical Freecell game. The spaces on the top left are the *Freecells* while the space on the top right are the *Foundations.* Each column with the cards is a tableaux pile

For the purpose of this report, we shall be relying on KPat to form an example of our studies. KPat is an open source implementation of a Freecell program that is included as a part of KDE's Entertainment package. It is included in most Linux distribution. The advantage of using KPat for the purpose of this investigation is due its open source nature – the source code can be easily view and modified.

II. HEURISTIC AND AI TECHNIQUES

A wide range of heuristic search is available to apply to a specific set of problems. The best fit heuristics would have taken into account the characteristic and the scope of the problem. The use of a combination of heuristic techniques may be demanded of the problem. A list of questions posed by [Rich] can be helpful in defining the problem. The game can be seen as a series of subproblems of moving sequential cards onto the foundation pile. Placing a sequential card onto a foundation pile is a meta-solution in this framework. However, since moving each card would modify the state of the board in obtaining a solution for another card, one meta-solution can have multiple effects on several meta-solutions. A single solution step cannot be undone as the now inaccessible card has been moved onto the foundation pile. Solving the special case of the solution card being on the top of the pile does not change the state of the tableau board but modify the state of the foundation pile. However, looking at Freecell as a series of individual card moves, if a given move does not enter the foundation pile it is recoverable. Therefore, Freecell is a mixture of recoverable and irrecoverable problem. As previously discussed the universe is predictable as the result of the open solitaire. Freecell is a best-path problem in view of my project motivation in aiding human. The solution is a path to the final state of all the

cards being in the foundation piles. The role of knowledge in this particular problem is determining constrains for the search and modify the search strategy on the fly. For this particular task, solving a given Freecell does not require interaction with human. However, a player can ask for help midway in his game. The possibility of human aiding the AI making the decision cannot be overlooked.

Now that we have refined the problem in terms of its properties, we can explore the numerous AI techniques that can be employed in solving this problem.

1) Search Algorithm

Most basic Freecell solver algorithm involves the use of Depth First Search involving atomic moves (moving one card at a time). The application of DFS in terms of Freecell is repeating legal move in sequence. The algorithm backtracks to the last legal move made when no legal move is possible and continues searching for another legal move until the Freecell deal is solved. The process does not lead well into interactive solving of a given deal but is effective at brute forcing the smallest sequence. The main danger of using Depth First Search is creating a sequence of move that is thousands of moves long involving many useless moves in between. The main symptom of this behaviour is repeating states. A check would be made between two paths of the same states. The shortest path arriving to that state is picked. The detail of this subset of DFS called Backtracking algorithm is discussed in [Pearl].

A Breath First Search implementation involves looking at all the possible move in the initial state while purging all the illegal moves, and then extending the tree into the second level repeating this process to the lone end branch. A typical deal of Freecell would contain around 10 starting legal moves which would expand exponentially into a big tree. The algorithm is memory intensive and the algorithm has O(n!) solving time. There are methods for speeding up a given Breath First Search. Various algorithms can predict the risk of failure of a given branch. For example, branches leading to no Freecell available for play in an extended period of time can be discarded since the probability of solving a Freecell deal is greatly reduced by not utilising the Freecells.

A strict A* implementation [Jones] of Freecell Solver is not possible since the number of moves for the shortest solution must be informed as part of the starting condition. The utilisation of Iterative Deepening can approximate the heuristic but the completeness of the algorithm is not assured. A* can be seen as a subset of Best First Search (see later) in that the heuristic used is an estimated number of moves. The use of Iterative Deepening [Moreland] can lead to the rough A* implementation to have the advantage of searching limited space but also has access to more of the available search tree. A suitable estimate of the search depth which in the case of Freecell can be classified in terms of difficulty can vastly improve search time. However, a bad estimation can have sequences on effectiveness of this search algorithm [Korf]. Iterative Deepening A* can learn from previous searches to obtain the estimate of depth limit such that the estimate depth limit is as close to the actual depth limit without going over.

In the game of Freecell, certain branch can be proven to be unproductive. The importance of using branch pruning techniques such as Alpha-beta pruning [Atallah] is indispensable in vastly shortening the search tree for the goal of real-time searches. Since Freecell does not involve any use of probability in its game state alpha-beta pruning technique can be used safely provided that the condition for an untransversable branch is strictly defined. An mortification to the original algorithm has to be made in that there is no scoring provided for Freecell. The alpha and beta value has to be another heuristic value that is dependent on the state of the board. The optimum search window for Freecell in terms of alpha beta pruning should be over a small range of cards since the algorithm should ideally put cards in the foundation pile over a small number of moves.

The above three implementations if applied purely are not admissible – it is not guaranteed to return an optimal solution if a solution exists. Refinement of the algorithms combined with applying specific constraints can lead to the algorithm finding a short solution for a specific type of deal. Since the aim of most Freecell solvers available are to search for the shortest solution for each combination, there is a lack of motivation to use those refinements since a complete Depth First Search of all possible solutions can yield the shortest path.

2) Best First Search using Heuristic Values

Using the Best First Search, in-cooperating domain specific information assigned as a value that measures the potential success of picking a given node would generate a search tree which employ domain information in selecting the next path. The heuristic value can be constructed by using variables that determines how much the puzzle is solved. The four variables are the following:

- **Number of cards to the foundation piles**: The most obvious indicator of how successful the heuristic is clearly the end goal of putting cards onto the foundation piles.
- **Sum of distance of cards away from the foundation pile**: Each card is assigned a value according a recursive algorithm working out the difficulty of taking away the cards on top. E.g. an ace in the top of the pile is assigned a value of 1 while a 3 under that ace would have a value of 1 plus however many value it would take to put 2 in the foundation pile. The total value of each given deal is the sum of the different moves that would take to move the cards onto the foundation pile.

- **Number of moves required to complete the board assuming infinite number of Freecells**: Assuming infinite amount of Freecells, each of the lower ranked card that cannot be place onto the foundation pile are placed onto a Freecell. This variable has the advantage of being easy to compute using a standard best-first search and gives an estimate of the problem that is close approximate to the solving of the actual problem.
- **Sum of number of moves required to sort a given tableaux pile:** Since one of the goals of Freecell is to sort cards of a single suit into a pile, if we assume that all cards have an available destination the difficulty of clearing a pile can be seen as the number of moves required to sort a pile into the correct order.

Best First Search has the advantage of being an algorithm that direct relates to the success its progress without using any learning techniques. However, it is obvious that computationally it is at least an order higher than most basic search methods, since the heuristic value has to be computed. For some of those, computing the heuristic value can be quite expensive computationally.

An algorithm refinement to Best First Search would be to only transverse paths that are promising. Many metric can be used in determining which path has the least potential for a successful solve but the most useful indicator is the number of open Freecells measured over a period of 10 moves (moving an incomplete stack from one pile to another would use up all available Freecell in several moves but would still leave the Freecell available for use after the operation). This additional refinement to Best First Search is called Beam Search.

3) Neural Network

A Neural Network approach can also be explored in solving the Freecell problem. Training data would consist of a large library of human playing Freecell. Different architecture can be used in constructing the Freecell. The encoding of the cards to be processed by a neural network can deeply affect the effectiveness of the neural net techniques. I have hoped to experiment with the different encoding techniques but unfortunately I have not been able to do so. The basic requirement of the encoding is for the location of all cards to be represented. A desirable properties of the encoding scheme is to make the constrains of the domain explicit in the encoding. Therefore, the best option is to encode based on the location of each card, assigning variables that would include their location on the board including if they are in the foundation pile and the Freecell. The hope is that the neural network training (supervised or otherwise) can pick up on the domain rule with no outside help. Although the AI engine should not be able to make moves that are illegal on the gaming board, it is highly undesirable for a trained neural network to make such an attempt in the first place.

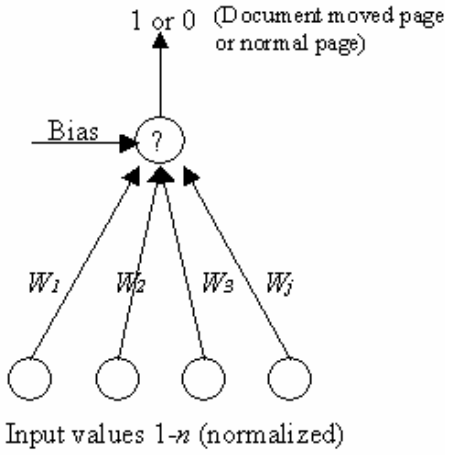

Figure 2. Logical view of a typical Single Layer Perceptron

The attraction to using a neural network is its use in analysing non-linear problems. An environment where the rules of the system are not immediately obvious lead itself to a neural network approach rather than a rule based approach [Lisboa]. In the case of Freecell, we have available to us empirical data that can allow the neural network to undergo a process of pattern recognition and classification.

3.1 Single Layer Perceptron

The simplest kind of neural network is single layer perceptron. This method was invented by Warren McCulloch and Walter Pitts. The network consists of multiplies of simple feed-forward network that simulates the state of a group of neurons. Each perceptron has a clear and concise operation that simulates a simple neuron. The operation of a perceptron can be classified in a simple equation:

$$P * W - b > 0 \quad (1)$$

Where P is the input vector, W is the weight and b is the bias. A given perceptron has two states of output, 1 and 0. As seen in the equation, if the sum of the left hand side is less than zero then the perceptron would be inactive (output 0). The learning process comes from an equation called the Delta Rule [Gurney] where the value of W and b is selected. A summary of the properties of the Delta Rule can be seen as the following:

1. The network has input units directly connected to output units.
2. The network is trained to produce output states for particular input states
3. The error between the actual output and desired output is minimised.

Single layer perceptron is too simple to compute a complex puzzle such as Freecell because they are limited to linearly separable problems. Single Layer Perception also has a problem with output that is significantly different from the input because of the lack of internal representation of the input pattern to be made. In the

highly influential paper *Perceptrons* [Minsky] Minsky and Papert has shown that it is impossible for Single Layer Perceptron to learn an XOR function (although they incorrectly state that multiple layers of perceptron would fail to do so as well).

3.2 Multi-Layered Perceptron

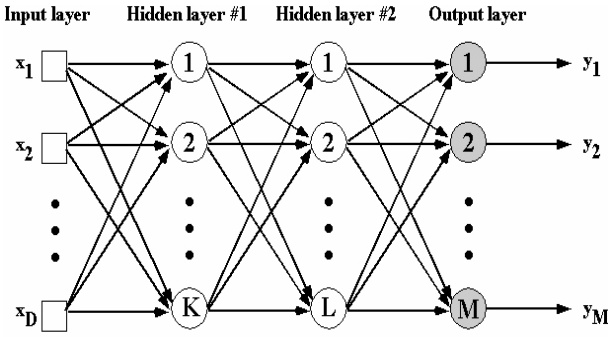

Figure 3. A generalised picture of Multi-Layer Perceptrons. The arrows are connections which have different weightings depending on the training

Using combinations of perceptrons some arranged within an internal layer, the lack of implementation of the XOR rule can be solved. This approached is called Multi-Layered perceptron. The Multi-layered perceptron is arguably the most popular neural network architecture. [Lisboa]. A set of MLP consists of 3 layers. The *inner layer* accepts the inputs from a source which is then passed though one or more *hidden layers* of perceptrons. The perceptrons of the last hidden layer is then passed though into the *output layer*. Every node in a given layer is usually connected to the layer above and below. Each interconnection between neurons has a weight which determines the importance of each connection. When the network is nearly constructed, each connection does not have a set value of weights. It is up to the training procedure to determine the weights of each neuron.

Non linear functions such as sigmoid and hyperbolic tangent are often used as the activation function allowing mild error to be ignored while strong signal are heavily favoured. Back Error propagation is the most popular training algorithm for multi-layered perceptron network. The correct output for the output units are known in training. However, we do not know the values which the units in the hidden layer should take. Hence we would require the weights of the hidden layers to be modified based on its strength in affecting the final prediction. Since the number of nodes is less than the number of weights, there is no linear equation that can determine the weights exactly corresponding to the desired output. The only recourse is that we minimise the errors at the output. We take the gradient of the error function with respect to each network weight and modify the gradient to the lowest point. This technique employs convergences towards minimising the error. The main weakness of back propagation is the possibility of local minimum stopping the function from converging towards the global minimum. Supervised training in the form of user training can avoid that particular pitfall.

Adding hidden unit gives the network additional properties that are very handy for the use of solving Freecell. 7 properties are identified from [Mammone]. I will provide a summary on why the reverent properties aids in solving Freecell.

1. Hidden units are among the determinants of the "attractors" or "stationary points" in activation space and of the net's capacity when recurrences are present. The identification of recurrences is inherent in determining a given neural network to learn the behaviour of Freecell.
2. Hidden units have virtually no constraints on the sorts of linear projections they may perform; they are "weak" feature extractors which would identify trends rather than the specifics (knowing going for lower number cards rather than a specific ruleset of moving a 2 of spade after taking the ace of spade)
3. Hidden units allow a powerful sort of discriminant analysis to be performed: A single layer of a sufficient number of hidden units can be shown to construct nonconvex regions in either a continuous or a Boolean feature space. Combining hidden layers would lead to the ability to analysis complex and implicit behaviour that is often found in Freecell.

The particular danger of MLP is that due to a large sample set available from user made data or from "perfect" solves of Freecells, overtraining is very easily applied. The number of parameters involved in solving Freecell can lead to generalisation error where the network is too sensitive to the significant of a specific move. The technique of back propagation [Carling] is used as an aid of supervised learning, as an extension to Delta Rule in determining rules that are not well defined but follow group behaviours. In particular to Freecell, the error rate should be flexible since the network should allow flexibility in the types of move rather than having the network learning moving a specific card to a specific cell. Error reducing techniques such as gradient descent [Carling p. 135-136] should not be employed for this particular problem. The focus of the learning model is on a *weak* learning algorithm since the training data is available but incomplete.

4) Self-Organising Map

Self-Organising Map [Kohonen] is a form of unsupervised learning that is invented by Finnish professor Teuvo Kohonen so it is sometimes referred to as Kohonen's Self-Organising Map. SOM employs a grid of neurons that adapt itself to the training set by self-selecting their individual weights using a technique

called vector quantization [Phamdo]. Initially the weighting of each neuron is set to a random value.

The algorithm is explained most easily in terms of a set of artificial neurons, each having its own physical location on the output map, which take part in a winner-take-all process (a competitive network) where a node with its weight vector closest to the vector of inputs is declared the winner and its weights are adjusted making them closer to the input vector. Each node has a set of neighbours. When this node wins a competition, the neighbours' weights are also changed. They are not changed as much though. The further the neighbour is from the winner, the smaller its weight change. This process is then repeated for each input vector, over and over, for a (usually large) number of cycles. Different inputs produce different winners.

The neural network techniques described above can be implemented in programs such as matlabs (with the neural network toolbox), and Clementine. However, there is no available package that will accept real time input of a given Freecell board and compute the neural net based on that input. For the purpose of a real time player aid for Freecell, a custom made engine has to be written.

Most of the techniques discussed above only apply to solving a Freecell board from the start. Some techniques can be employed midway, such as neural network and knowledge based heuristic. Regardless, a large challenge is to allow a method of communication between the AI algorithm and the Freecell board in a modular function that allows the AI algorithm to be interchanged.

5) Decision Tree

Decision Tree [Breiman] is a machine learning technique that at first glance could be a very easy way of solving the Freecell problem. The induction of past performance into the predictive model would lead to a natural algorithm that would discover logical pattern in existing human game play of Freecell. The main feature of decision tree is its use of divide-and-conquer approach to the classification problem. A decision tree is a predictive model, suitable for mapping the observations of each given decision to form a generalised pattern from the training data.

Decision tree are constructed recursively by first selecting an attribute to compare and make a branch for each possible subset of value. The classification of each subset are analysed by the induction of the given machine learning technique employed for the decision tree in looking for regularities in data. The most common method of tree induction is divide and conquer, which the classification system C4.5 is based upon.

There are however many problems: the complexity of the Freecell board state would mean a working decision tree would be extremely large. The number of failed tree route would also be large since the pattern is formed by evaluating lower ranked cards first based on the training pattern. One method of overcoming the hard decision making is the use of MLP is the implementation of the tree classifier. [Ishwar] The use of soft nonlinearity functions such as sigmoid or hyperbolic tangent in the neurons can overcome many of the decision tree weakness in dealing with a “soft” problem such as Freecell. The noise suppression ability of sigmoid for example enhances general pattern recognition of MLP. [Havey] The main problem with this technique is that the number of neuron in the first hidden layer must be equal to the number of internal nodes of the decision tree (since each of the neuron has to implement the decision function of the internal nodes). Even when we reduce the function of the decision tree to consider atomic moves and use a sum modifier function to represent the “hidden” cards rather than an entire decision tree being drawn out, there are still a vast number of internal nodes.

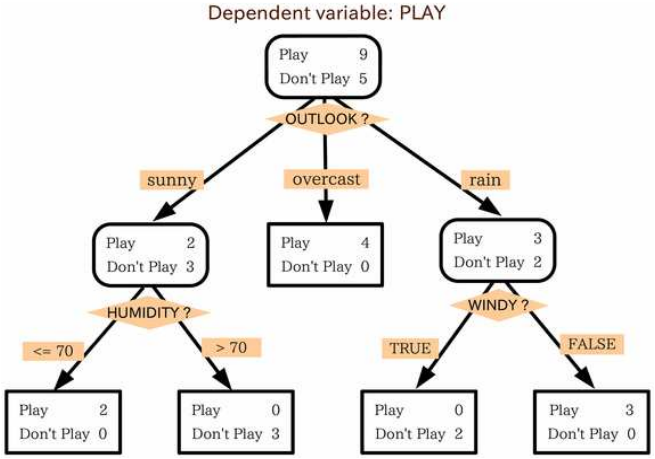

Figure 4. Example of a decision tree involving the playing of golf.

The number of neurons required would be vast. However, with today’s processor speed simulating thousands of neurons should not pose a significant computational problem for real time analysis.

5) Human preference

When playing Freecell, there are many preferences a human would have that would be easily be identifiable without doing extensive testing. First of all, human have an instinctive need to hunt for Aces even if buried under a few cards. When the Ace is obtained, the lower numbers that follows the Ace would be given preferences. When a player has finished placing cards inside the foundation pile initially, they usually have a need to arrange a few stacks of cards in order to aid them in the future when the lower number cards are uncovered. A beginner would usually fill up all the available Freecell but after a few games a player would develop an instinct to leave 2 Freecells open because most “stuck” Freecell board occur because of a lack of Freecell. Below a certain number of cards in a tableaux pile, a player will try to empty that tableaux pile. Having created that empty column, a player will usually fill that

column with large run of cards. An advanced player will leave all Freecell open unless he is performing a sequence of moves. Having identified the behaviour of a player, it would be very handy to develop a model describing the behaviour of a given player mathematically.

A concise mathematical model can compute the probability of player performing a given function based on the sum each modifier on every card in a given pile. Since a player can only take a card from the top of each pile, we can compute the probability for each pile and select the highest probability pile as the result.

Some considerations a human may take in judging which cards to move:

Ace:
Amount of card it is below

Normal cards:
Any possible move to the foundation pile
Distance from the foundation pile
Any possible move to a stack
Size of the stack being moved to
Is it a King and is an empty tableaux pile available

Freecell:
Number of remaining Freecells
Whether the card can go into a foundation
Is there a pile for the card to return

Each different card players has a preference on the weighting of each modifier. A collection of games playing by human players should be made, and the frequency of a player picking a given move should be set against the probability of the given move in the board to compute the modifier. An easier way but less natural is to make a series of test asking a player how likely he/she is to want to make that move from 1-10. This test has the advantage of giving an easily obtainable modifier but since the situation is not a real game playing scenario we lose some accuracy, and we have lost the interaction between the board and the player over a long term basis.

Earlier on we have discussed decision tree and why it is unsuitable for the problem of Freecell. However, if we see the process of decision tree as a single tree that only base its decision on a given modifier for a single move rather than following the tree for a whole Freecell solve (i.e. Use the decision tree for a single move rather than worrying about the dependences of multiple moves), we can employ the use of decision tree together with the modifier we have computed for this section. The decision tree simply has to deal with deciding which of the 7 piles of cards to move if any, and whether to move the cards inside a Freecell.

6) Bayesian Learning

Bayesian learning is a technique based on the usage of probability (Bayes theorem [Stigler]) to select the path with the minimum conditional risk. There can never be certainty, but as evidence accumulates, the degree of belief in a hypothesis changes; with enough evidence it will often become very high (almost 1) or very low (near 0). Bayesian theorem is used to differentiate between the different hypotheses. At its very basic, Bayesian learning allows the assignment of prior probability into the belief network – we can assign the above described user discovered modifier into the system to speed up the learning rate of the network. The frame work of generalising equations of user probability in taking away a card based on the analysis of the underlying cards and the state of the board can be applied into a Bayesian Learning framework.

III. INCREASING DEPENDENCIES USING DIRECTED GRAPH

KPat's solver [kdegames] employs the use of a variation of Depth First Search that has its main emphasis on speed rather than finding the shortest path in a solution. The key mistake KPat's solver is responsible for is its lack of recognising moves that is stuck in a circular motion. The purpose of my modification is to employ a tree structure identify potential points for such circular motion and move the tree structure to eliminate such circular motion when it is required.

The cause of the circular motion is KPat's lack of ability to recognise moves that is dependent on another card underlying the card that is being moved. Therefore KPat would undertake several independent moves that are irrespective of the state of the board without giving the impression of having an overall strategy regarding the game state. This lack of focus lead to the program taking back moves that the program has committed several moves before.

To solve this problem, we need a program that would identify dependences and be able to group them in a visual way. The program take each possible atomic move, work out the dependence of the various move, and draw a directed graph of the various moves based on those dependence. The directed graph would have an in-degree of =< 2, and the number of out-degree would also be =< 2.

The use of this program is to guide the player towards moves that are dependent upon each other rather than doing several independent moves. The side effect of this approach is that moves would be increasing the impression that the solver has a strategy that it would follow in.

The main difficulty in creating the program is constructing the data structure from KPat's solver algorithm, and injecting the modified move routine back into KPat. The modular approach I have taken has shown to be a hindrance in creating a comparability problem.

The main problem with the directed graph tool is that sometimes it would choose paths that would produce a stuck board where the normal KPat solver would have solved the board. Also, there are a few bugs in the behaviour of the direct graph tools in retracting moves. However, generally the program produces a more elegant solution to the existing KPat solver that takes a shorter number of moves to solve. Due to a lack of time extensive testing is not performed.

IV. FUTURE WORK

The intent of this project is to intent and inform the state of research in the field of AI that can be used to attack the problem of Freecell. The investigation has opened many revenues that can be explored in creating an AI that can aid human play Freecell.

Both neural network techniques described has not been employed in solving the problem of Freecell. Since the success rate of the neural engine is not known, we cannot work out how computationally rewarding both techniques are. A initial test using a set package such as matlab trained using shortest path information obtained from DFS can be applied for both techniques to work out the suitability of employing such techniques. If the result shows that both neural network techniques can arrive at a solution then a custom engine could be written for both neural network architectures. This report has shown that MLP in particular has many tweaking options that can be explored.

Ultimately the aim is to compare the effectiveness of each AI modules using techniques described above to form an algorithm that is suitable for adapting to the gameplay of Freecell. It may be that certain algorithms excel at certain state of game board (e.g. BFS for the ending of each solve). The easiest way to calculate the effectiveness of the given algorithm is to test it against a large number of Freecell boards and compare the average move required by each algorithm to solve the boards. The aim of the exercise here however is produce human like behaviour. The best judge of this behaviour is though the use of user testing.

A method of comparing the effectiveness of 2 techniques in their ability to copy a certain human player can be to do direct user testing. The user should play the Freecell as normal with the selected AI engine predicting the user's next move. When the AI engine derivates from a given user input the two engines each offers the next move for the user. Based on the choice the user makes, we can understand which algorithm the given user prefers. The advantage of this technique is that a user is given a straight choice that can leave no ambiguous answer, and a clear score can be computed in the evaluation of two techniques.

V. CONCLUSION

This report has outlined the numerous routes in which a researcher may explore in constructing a sophisticated algorithm for the purpose of solving Freecell. The main difficulty of this field is the lack of journals published on Freecell or any solitaire-like game. Most AI research techniques concentrate on multiplayer games such as Chess where the AI would make a response to a human's move in a confrontational sense. A co-operative approach of human-AI has not been widely considered in Game Theory. It is my hope that the several techniques outlined in this report would be explored in creating an algorithm that would lead to more human-like behaviour in playing Freecell.

REFERENCES

[Keller] Michael Keller Freecell Tutorial http://solitairelaboratory.com/tutorial.html

[Rich] Artificial Intelligence by Elaine Rich, Kevin Knight, McGraw Hill, 1993 (page 44).

[Jones] H. Jones, A* Algorithm and Implementation Tutorial, http://www.geocities.com/jheyesjones/astar.html

[Pearl] Heuristics- Intelligent Search Strategy for Computer Problem Solving by Judea Pearl.

[Phamdo] Data Compression Techniques by Nam Phamdo http://www.data-compression.com/vq.html

[Gurney] An Introduction to Neural Networks by Kevin Gurney, Taylor & Francis, Page 59, 1997

[Moreland] Bruce Moreland, Iterative Deepening Tutorial, http://www.seanet.com/~brucemo/topics/iterative.htm, 2002

[Lisboa] P.G.J. Lisboa Neural Networks Current Application, Chapman & Hall, 1992, Page 15

[Carling] Alison Carling, Introducing Neural Networks, Sigma Press, 1992 Page 133-154

[Breiman] L. Breiman, J. Friedman, R. A. Olshen and C. J. Stone, "Classification and regression trees". Wadsworth, 1984.

[Mammone] Mammone, Zeevi, Neural Networks Theory and Applications, Academic Press Inc, 1991, Page 182-183

[Stigler] Stephen M. Stigler, Thomas Bayes' Bayesian Inference, Journal of the Royal Statistical Society, Series A, 145:250–258, 1982

[Minsky] Minsky M L and Papert, Perceptrons, MIT Press, 1969.

[Korf] Richard E. Korf, Michael Reid and Stefan Edelkamp, Time Complexity of Iterative-Deepening-A*, Artificial Intelligence, 129 (June 2001), no. 1-2, pp. 199-218

[Kohonen] T Kohonen, Self-Organizing Maps, 1995

[Atallah] Mikhail J Atallah, Algorithms and Theory of Computation Handbook, CRC Press, 1998, page 36-12 to page 36-14

[Ishwae] Ishwar K Sethi, Decision tree performance enhancement using an artificial neural network implementation, Machine Intelligence and Pattern Recognition Series, volume 11, 1991

[Havey] Robert Harvey, Neural network Principles, Prentice Hall, 1994, page 63

[kdegames] http://games.kde.org/kde_cardgames.php