"FreeCell" Neural Network Heuristics

Alphonsus Dunphy, Malcolm I. Heywood
Dalhousie University,
Faculty of Computer Science,
6050 University Avenue, Halifax, Nova Scotia. B3H 1W5

***Abstract*— In areas, such as planning, state space searches are often conducted to find solutions. Usually, the heuristic is derived from knowledge of the domain. In many cases the knowledge of a domain is limited or the domain is so complex that an effective heuristic cannot be formulated. As an alternative, machine-learning techniques such as neural networks may be used to derive the heuristic. The game of FreeCell was selected as a suitable benchmark domain, in which "knowledge based heuristics" and "neural heuristics" were employed to find solutions for randomly generated games. An amalgamation of the two, in which the neural network developed a heuristic from several knowledge based heuristics, was also used. Of the neural derived heuristics, the best-case architecture did not employ the "knowledge based heuristics." Moreover, neural heuristics were not able to improve upon those defined *a priori*.**

***Index terms*—Benchmarking, State Space Search, Search heuristics, MLP, SOM.**

I. INTRODUCTION

FreeCell is a popular card solitaire game invented by Paul Alfille in 1978 [1]. Its inclusion in PC operating systems has enhanced its popularity and there are several tournaments in which FreeCell enthusiasts participate. Its simplicity of rules and its diverse number of games and solutions make it suitable for heuristic search techniques. Since games are diverse, with 1.68038×10^{66} possible initial card layouts, basic searches such as breadth and depth first are ineffective.

The playing rules of FreeCell game are straightforward. The deck is the standard deck of 52 cards. The cards – ace, deuce, three to ten, jack, queen, king – are ranked 1 to 13 respectively, with 1 being the lowest rank. The color of the suits, ♥hearts and ♦diamonds, is red; the color of the suits, ♠spades and ♣clubs, is black. The arrangement of the cards is as follows:

- 8 container stacks or columns of unlimited size, into which a standard deck of 52 cards is, at the beginning of the game, randomly placed face up with 7 cards in the first four stacks and 6 in the remaining four.
- 4 free cells, which are allowed to contain a single card.
- 4 collector stacks into which all cards are eventually collected.

In FreeCell literature the stacks and cells are often referred to as tableau piles, cells, and foundation piles respectively [2]. The object of the game is to relocate all cards of the container stacks to the collector stacks. Each card may be moved according to the following rules:

- A card may be moved from a free cell or container stack into a second container stack if the receiving container stack is empty *or* if the card is different in color and one rank below the top card of the receiving stack.
- A card may be moved from a container stack (or a free cell) to a free cell if the free cell is empty.
- A card may be moved from a container stack or free cell to a collector stack if the card is an ace and the receiving stack is empty *or* if the card is of the same suit and one rank above the top card in the collector stack.

Once a card has been placed in a collector stack it cannot be removed.

Many programs, with diverse names such as *FreeCell Pro* [3], *Xcell* [4], *FreecellTool* [5], and *AutoFree* [6], have been written to automatically solve FreeCell. Some use specifically developed algorithms, while others use state space searches such as heuristic search and A* Search. Normally, the heuristic is based on knowledge of the FreeCell game. Few, if any, use machine-learning techniques such as neural networks. Since games are diverse and a large number of attributes are necessary to describe a layout, it would be difficult to train a neural network to solve the game directly by predicting a move for each possible layout. An alternative would be to train a neural network to calculate a heuristic to direct a best-first search. In a similar fashion, Chellapilla and Fogel use neural networks to evaluate board positions in the game of checkers [7, pp 1482-1495]. This is the approach employed here. Moreover, the specific interest is to identify the significance, if any, of different neural network architectures, relative to a baseline of performance established by *a priori* defined heuristics or "knowledge based heuristics", in both places applied in conjunction with a state space search.

The paper is organized as follows: Section II defines the state space approach within the context of the FreeCell game, and introduces the "knowledge based" heuristics. Section III provides the methodology utilized to define the neural network architectures and representation of the input space. The associated neural network learning algorithms are defined in Section IV, whereas Section V describes the results. Conclusions are drawn in Section VI.

II. State Space Search

FreeCell was 'solved' by a best-first or heuristic search. The heuristic search is a state space search conducted through a tree or graph of nodes. In FreeCell, each state or node is a card layout of the game. The initially dealt hand is the start node or first node of the search. The goal node is the node at which all cards have been placed in the collector stacks and which signifies that a solution has been found. Successor nodes are the possible card layouts that can be generated from a node by making all legal moves. The parent node is the node from which successor nodes are generated. The heuristic value of a node is the estimated cost of reaching the goal node from the node. In FreeCell the cost of a move is unity and, hence, the heuristic value is generally the number of moves required to reach the goal node. However, it may also be an arbitrary value that tends to decrease as the distance from the goal node decreases.

An open node is one that has not been reached in the search, while a closed node is one that has been searched. OPEN and CLOSED are lists of nodes. A solution is the path from the start node to the goal node.

A simplification of a best-first search is as follows [8, pp 92-93]:

```
1. Start with OPEN = start node;
2. While (node $\neq$ Goal) OR (OPEN == $\varnothing$)
   a. Pick node on OPEN with best heuristic value and
      move node to CLOSED;
   b. Generate Successor Nodes;
   c. For (all Successor Nodes)
      i. Calculate heuristic value and add node to OPEN;
      ii. If generated before, change the parent node for a
          better path;
3. A solution, if it exists, is the path from the start to the goal
   node.
```

Only single moves were used to generate successor nodes. Super or meta moves were not used. A super or meta move is a series of several legal single moves, which can transport a sequence of several cards from one collector stack to another. Naturally, the performance of the best-first search is strongly influenced by the heuristic employed. To this end the following 5 knowledge-based heuristics are defined.

A. *Number of Cards Collected (NCC)*

The number of cards that have been collected is an indicator, albeit a weak one, of progress in the search for a solution. The negative of the number of cards that have been moved to the collector stacks can serve as a heuristic.

B. *Distance of Node from Goal (NfG)*

This metric basically sums the number of cards between the position of each card in the node under consideration and the position of that card in the goal node. When the distance has been calculated for a card, that card is considered removed from the container stack of the node under consideration and will not be used in calculations for the remaining cards. A card that has already been moved from a container stack is obviously not blocking any others. If the node already has cards in the collector stacks, then the distance for these cards is 0. If the node has a card in a free cell, than the distance for that card is 1.

C. *Rank Order (RO)*

Games, which have a high degree of card order in the container stacks, tend to be easier to solve and, hence, closer to the goal node. One means of estimating the order or (lack of order) is to estimate the difficulty to sort the cards in the collector stacks. Hence, the card order heuristic is the sum of the number of moves that it would take to sort the cards of each container stack in ascending order of rank (from top to bottom) ignoring card suit.

D. *Sequence Order (SO)*

This heuristic measures the order of the cards in the container stacks of the node in terms of both rank and color. To do so, the sum of costs of the cards is estimated as follows:

1. If, in a container stack, a card and the one above is in proper rank and color sequence such as a black 2 on a red 3, a negative cost equal to the depth of the card is assigned;
2. If the rank sequence is correct but the color sequence is incorrect, then (the difference in rank + 1) assigned as a cost;
3. If the rank sequence is incorrect and the above card is higher in rank, (the difference in rank − 1) × the depth of the card is assigned as a cost.
4. If the rank sequence is incorrect and the above card is equal or lower in rank, (the difference in rank + 1) is assigned as a cost.

E. *Problem Reduction (PR)*

Often a good heuristic for a search is the number of moves required to complete the search for a simplified version of the node. One means of simplifying a FreeCell node is to add an unlimited number of free cells. To solve the simplified version, low ranking cards are collected after continually moving covering cards to free cells. Each movement of a card from a container stack or free cell to a collector stack or from a container stack to a free cell is considered a single move. The heuristic of the original node is the number of moves it would take to reach the goal node from the simplified node. Since the heuristic is close to being admissible, this heuristic is useful in finding short solutions via an A* Search [8, pp 96-101].

III. Neural Networks

Instead of using the above knowledge-based heuristics, neural networks were used to calculate the heuristic value. That is to say, the search for a FreeCell solution is conducted in exactly the same manner as described previously, with the neural network calculating the heuristic value used in the search. Training data for the neural network would consist of previously solved FreeCell games with a coding scheme to describe the current state and the number of moves to complete the game as the target. Naturally, many options exist in terms of different neural network learning algorithms. However, in this work, our principle interest lies in the contribution of different architectures and the input space employed. To this end, three different architectures are considered.

Firstly, architecture 1, a single neural network, is used to find suitable search heuristics based on the card layout. This represents the neural network base line, as no *a p riori* information to encourage modular problem solving is incorporated [9]. Architecture 2 uses the knowledge-based heuristics from Section II as the input to the neural network. The question being, can a neural network learn to combine heuristics without recourse to the card layout? A combination of card layout and heuristics is considered to result in an undesirably large input space. In the final case, architecture 3, card layout is retained as the input, but two neural networks are employed, one to perform feature extraction and one to make the heuristic.

In all the above architectures, Multi-layer Perceptrons (MLP) with one hidden layer are employed for identifying the heuristics; Architectures 1 and 2. Architecture 3 utilizes a Self Organizing feature Map (SOM) to perform the feature extraction. Moreover, the SOM output provided to the following MLP takes the form of the quantized (real valued) SOM outputs, Section IV.

A. *Encoding the Input Space*

The selection of attributes of the domain is critical to the performance of a neural network. The training time, the ability of the network to generalize, and the minimum mean square error (MSE) are affected by the choice of attributes. FreeCell is, especially, problematic since there are numerous ways of encoding to produce layout patterns. In the case of this work we consider there to be two basic schemes: cell content or card location.

The FreeCell game is described in terms of labeled cells that can contain a card. The attributes are variables describing the card in a particular location. For a cell content encoding, two attributes are used to describe the card in a cell. One integer describes the suit of a card and a second describes the rank of the card. In the case of the card location scheme, there is one attribute per card, which identifies the card location.

Experiments using various coding schemes were conducted. The most successful and the one chosen to encode the training and test data for FreeCell was the Cell Content scheme, in which 2 attributes described the location of all cards, including those in the collector stacks. The success of the Cell Content scheme as compared to the Card Location scheme is not surprising, if we view the schemes in human terms. The Cell content scheme clearly displays the sequence of cards in the stacks; whereas in the Card Location scheme the sequences are not apparent.

B. *Training and Test Data*

Generating FreeCell solutions to train neural networks is a cumbersome process. Ideally, solutions should be perfect, that is, the solutions should contain the minimum number of moves. A breadth first search would generate optimal solutions. However, these are impractical on a FreeCell game using the full 52-card deck. Even with reduced games of 24 and 32 card configurations, which contain cards of 3 suits in rank ace to 8 and cards of 4 suits in rank ace to 8 respectively, breadth-first search is unfeasible. That is, tens of thousands of nodes would be generated requiring unattainable memory and CPU requirements. However, for the 24 and 32 card configurations it was still relatively easy to generate test data. An A* Search using the PR heuristic attains nearly optimal solutions. Only on rare occasions, less than 1%, in testing on the simple 14-card game would a solution be one greater than that found by a breadth first search. Therefore, the 32 Card Deck, which is sufficiently complex, was chosen as the domain for analysis.

C. *32 Card Deck Domain*

The layout of the 32 Card FreeCell game and the number of attributes in the Cell Content coding scheme required to describe each state is as follows:

- 5 container stacks, each large enough to contain at least 8 cards. The number of attributes = $(5\times8\times2)= 80$.
- 4 free cells. The number of attributes = $(4\times1\times2)= 8$.
- 4 collector stacks, each to contain the 8 cards of each suit for the solved game. The number of attributes = $(4\times8\times2)= 64$.

A total of 152 attributes are required.

IV. Learning Algorithms

A. *Self-Organizing Feature Map*

Kohonen's Self-Organizing Feature Map (SOM) algorithm is an unsupervised learning algorithm in which an initially 'soft' competition takes place between neurons to provide a topological arrangement between neurons at convergence [10]. The learning process is summarized as follows,

1. Assign random values to the network weights, w_{ij};

2. Present an input pattern, $\boldsymbol{x}$, in this case a series of taps taken from the shift register providing the 'reconstruction' state space on which the SOM is to provide a suitable quantized approximation.
3. Calculate the distance between pattern, $\boldsymbol{x}$, and each neuron weight $\boldsymbol{w}_j$, and therefore identify the winning neuron, or

$$d = \min_j \left\{ \left\| x - w_j \right\| \right\} \quad (1)$$

where $\|\cdot\|$ is the Euclidean norm and $\boldsymbol{w}_j$ is the weight vector of neuron j;
4. Adjust all weights in the neighborhood of the winning neuron, or

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)K(j,t)\{x_i(t) - w_{ij}(t)\} \quad (2)$$

where $\eta(t)$ is the learning rate at epoch t; and $K(j, t)$ is a suitable neighborhood function, in this case of a Gaussian nature;
5. Repeat steps (2) – (4) until the convergence criteria is satisfied.

Following convergence, presentation of an input vector, $\boldsymbol{x}$, results in a corresponding output vector, $\boldsymbol{d}$, the Euclidian distance between each neuron and input or quantization error. It is this concept of quantization error at each neuron which is forwarded to the following MLP.

B. *Multilayer Perceptron*

Various learning algorithms where evaluated including the LMS algorithm with adaptive learning rates and momentum and second order derivative approaches [11]. In this case, best results where achieved using second order derivatives as estimated using the Levenberg-Marquardt quasi-Newton method ('trainlm' in Matlab™ [12]). Specifically, the back propagation algorithm popularized the utilization of gradient methods for training MLP networks [13]. Weight updates, $\Delta \boldsymbol{w}$, take the form of a gradient vector, $\boldsymbol{J}$ – changes in the sum square error, $\boldsymbol{E}$, with respect to weights, $\boldsymbol{w}$ – and expanded in terms of the chain rule to provide the 'back propagation' across multiple layers. However, second order methods have the potential to provide a faster learning algorithm if efficient estimation of the Hessian matrix, $\boldsymbol{H}$, is possible. In this case the gradient vector, $\boldsymbol{H} \approx \boldsymbol{J}^T \boldsymbol{J}$, approximates the Hessian. The Levenberg-Marquardt algorithm [14] then employs a combination of this Hessian approximation with a quasi-Newton gradient update scheme and a suitable annealing schedule to control the adaptation of the learning rate parameter, η, as follows,

$$\Delta w \approx -[\boldsymbol{J}^T \boldsymbol{J} + \eta \mathbf{I}]^{-1} \boldsymbol{J}$$

The selection of the learning rate parameter, η, enables variation between the special cases of gradient decent ($\eta \rightarrow 1$) or Newton's method ($\eta \rightarrow 0$), where the latter is most desirable when an error minimum is encountered. The Matlab™ 'trainlm' routine employs one of two multipliers to update the learning rate at each epoch [11], depending on whether the last epoch resulted in an error decrease ($\eta(t + 1) = \alpha \times \eta(t)$) or an error increase ($\eta(t + 1) = \beta \times \eta(t)$).

V. Results

For neural training purposes 101 games comprised of 4852 patterns, generated using the Reduction heuristic of Section II.E, were utilized. Each pattern represents a node or move in a game. An additional 1000 games, again generated with the Reduction heuristic, served as data to test the networks. If the search was not completed before 5000 nodes were generated, the search was terminated and the search deemed to have failed. Normally, the unit of testing is the pattern, in which a neural network is judged by the proximity to which it calculates the target from a pattern. However, in FreeCell a network is judged by performance on solving FreeCell games. In all cases the Neural Network toolbox of Matlab™ was employed [12].

A. *Performance metrics*

The performance of any approach – knowledge based or neural network – is evaluated in terms of two basic search parameters: Efficiency and Effectiveness.

1) Efficiency: expresses the number of *nodes closed* as the number of nodes from which successor nodes have been generated and which have been placed in the CLOSED list of nodes. A closed node is the one with the best heuristic value of those that have yet to be expanded. The 'total number of nodes' is the actual number of nodes that have been generated. Efficiency, therefore, measures the quality of the search algorithm and the heuristic.

2) Effectiveness: measures the number of nodes contained in the solution or solution length. In FreeCell the solution length is the quality of the solution and corresponds to the number of moves to complete the game. The breadth-first search and A* Searches will find the optimal solution, if a solution exists. The "number of games solved" is the number of games that could be solved within the prescribed time frame (node generation limit of 5000).

B. *Network Topology*

Neural Network heuristics based on the MLP all utilize three hidden layer neurons, tansig activation function, and a single linear output neuron. MLP#1 excludes attributes for the cards that have been placed in container stacks, whereas the patterns for MLP#2 include these attributes. Both MLP#1 and #2 are representative of architecture 1.

In the case of architecture 2, six knowledge-based heuristics are employed in conjunction with a multilayer perceptron, MLP#3. The first three are the Distance from Goal, Rank, and Sequence and the remaining three are variations of the Rank and Sequence heuristics. (Note the PR heuristic is used for directing the best-first search during creation of the datasets and, hence, cannot be employed as training data.)

Architecture 3 also utilizes a MLP with 3 hidden layer neurons. Three alternative SOM configurations – SOM#1, #2 and #3 – are considered. SOM#1 excludes attributes for the cards that have been placed in container stacks, whereas the patterns for SOM#2 and SOM#3 include these attributes. The topology of the map for SOM#1 and SOM#2 was 15 × 7 hexagonal grid, whereas the topology of the map for SOM#3 was a 105 × 1 linear grid.

C. *Static Analysis*

Table I details the test set performance in terms of *nodes closed*, *nodes open*, and solutions found for heuristic and neural solutions. The simplest heuristic, the Number of Cards Collected Heuristic managed to solve 970 of the 1000 test games, albeit it was the least effective. On average it searched 406.5 nodes to find solutions of average length 152.2. The number of *nodes closed* is equivalent to the number of nodes searched. The other knowledge-based heuristics performed much better with the PR heuristic searching an average of 205.5 nodes to find solutions of average length 63.1. It solved 992/1000 or 99.2% of the games.

TABLE I
COMPARISON OF SEARCH HEURISTICS ON 1000 TEST GAMES

Heuristic	Avg. Nodes Closed	Avg. Nodes Open	# Solution Found	Avg. Solution Length
Knowledge Based Heuristics				
NCC	408.6	878.5	970	152.2
NfG	205.5	448.7	994	64.4
RO	203.0	494.9	990	64.7
SO	251.2	543.9	972	70.7
PR	181.5	441.1	992	63.1
Architecture 1 – Card layout, no independent feature extraction				
MLP#1	285.8	683.4	979	70.7
MLP#2	353.2	960.4	983	93.7
Architecture 2 –Heuristics, no input partitioning				
MLP#3	211.6	542.5	991	64.0
Architecture 3 – Card layout, independent feature extraction				
SOM#1	241.1	600.4	990	67.0
SOM#2	234.6	554.0	996	65.5
SOM#3	208.4	502.8	992	62.7

The performance of the architecture 1 neural networks – MLP#1 and #2 – was significantly less than that of the knowledge based heuristics. MLP#1 searched through an average of 285.8 nodes or 57% more than the PR heuristic. The higher performance of MLP#1, in comparison to MLP#2, on *nodes closed* and *nodes open* was probably attributable to having fewer attributes on which to converge. Architecture 2, MLP#3 – an amalgamation of 6 heuristics – tended to be biased toward the better performing heuristics.

Architecture 3, the introduction of an SOM, is an improvement over a single MLP, whether the MLP is based on the card layout or heuristics. SOM#3 performs at a level approaching that of the "knowledge based heuristics." The corresponding average *nodes closed*, 208.9, is near that of the Distance from Goal and Rank heuristics, whilst providing more and shorter solutions. SOM#2 and SOM#3, in which patterns contained attributes for all cards including those in the collector stacks, performed better than SOM#1.

D. *Generalization*

Further experiments were conducted on 100 randomly selected games with the objective of assessing the degree of generalization provided by SOM#2. To do so, new games were created from the original games, by swapping randomly selected pairs of cards. For each of these 100 games, new games were generated by randomly swapping from 1 to 5 pairs of cards from the original game configuration, and then measuring performance in terms of *nodes closed*.

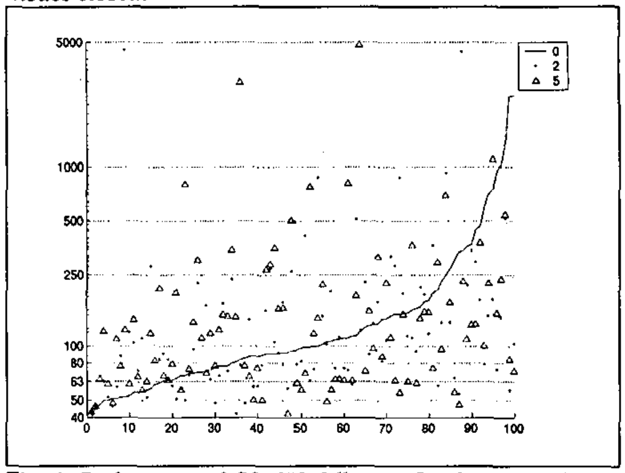

Fig. 1. Performance of SOM#2 following Random Swapping of cards. X Axis: game i.d. Y Axis: $\log_{10}$ *number of nodes closed.*

Figure 1 summarizes the experiments. The solid line represents the performance of SOM#2 on the original 101 games with 0 cards swapped. Altered games with 2 and 5 cards swapped are identified as '•' and 'Δ' respectively. Easy games are games that can be solved with a low number of *closed nodes*. Swaps to easier games such as games 1 to 10 in Figure 1, result in comparable or more difficult games, require an equivalent or greater number of *nodes closed* to solve and generally fall above the solid line. As the original games become more difficult (games 10 to 80, Figure 1), the swapped cards result in both more difficult and easier games. The '•' and 'Δ' points fall both below and above the line. Swaps to the most difficult (games 80 to 100, Figure 1) tend to result in easier games. Most '•' and 'Δ' points fall below the line. In summary, easy games become more difficult and more difficult games easier as cards are swapped. This supports the premise that the networks are providing general solutions that do not appear to be overtly sensitive to specific game layouts.

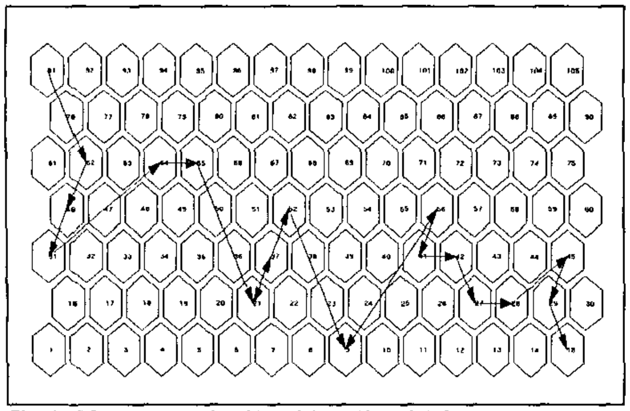

Fig. 2. SOM#2 on Nodes Closed for a 'Simple' Game.

E. SOM dynamics

Searches on 15 games were traced from neuron to neuron through the SOM identified as SOM#2. As each node is searched, the winning neuron for that node is calculated. Figure 2 and 3 contain traces of 2 games, one game solved in a low number of moves and a second game solved in a large number of moves. The SOM, used in training, was a 15 × 9. Numbers from 1 to 105 were arbitrarily used to label the neurons. Arrowheads indicate the direction of the solution. Consider a simple game. The initial card layout or node is located at neuron 91. The next node searched is located at neuron 62. The search gradually progresses through the SOM and eventually reached the final or goal at neuron 15. The most interesting feature of the trace is that solutions normally start at neuron 91, move across the SOM, and exit at neuron 15. It is understandable why the last winning neuron is always 15, since the goal node is always the same. However, a simple explanation for why solutions always start at neuron 91 is not obvious. Perhaps, the fact that the start node of all games has the same card positions filled may influence the assignment of the winning neuron. Moreover, the initial configuration also represents the least organized, where this is identified by one neuron performing an averaging function.

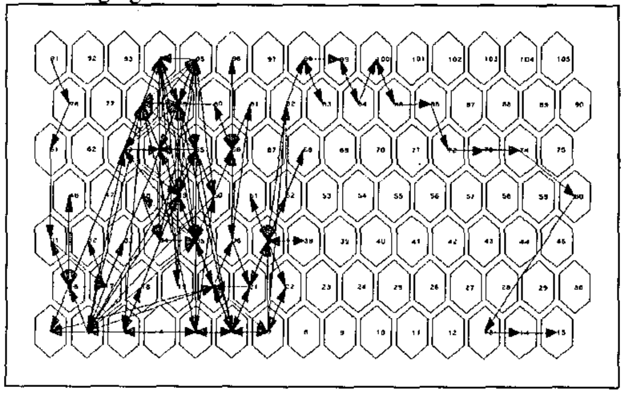

Fig. 3. SOM#2 on Nodes Closed for a 'Difficult' Game.

Complex or longer solutions, Figure 3, have many feedback and internal loops. The loops are generally in the early part of the solution. Once the solution approaches the winning goal the trace becomes less complex. This suggests that neurons may be overworked. Any technique that would spread the organization of attributes more evenly over the SOM would likely improve performance.

VI. Conclusion

Performance of neural network architectures is evaluated against a set of *a priori* selected knowledge based heuristics on a 32-card version of the FreeCell game. Particular emphasis is given to the evaluation and identification of appropriate neural architecture. A combination of SOM and single hidden layer multiplayer perceptron is found to provide competitive performance with the knowledge based heuristics. Trace analysis of the SOM once trained indicates that the SOM clearly partitions the problem into different stages of play. Moreover, the SOM may well benefit from the ability to encode temporal relationships in an attempt to provide additional context to the current spatial encoding of the input space.

In short, we believe that the FreeCell game provides a good benchmarking environment for a wide range of learning algorithms, providing both a concise definition and interesting spatial and temporal learning problems.

References

[1] History of FreeCell, http://www.solitaire-freecell.com/history_of_freecell.htm.

[2] Solitaire Games Week – FreeCell, http://www.solitairegames.com/freecell.html

[3] FC Pro, http://home.earthlink.net/~fomalhaut/fcpro.html.

[4] Xcell, http://www.gamesdomain.com/directd/2154.html

[5] FreecellTool - Program Detail - WindowsPC.com, http://www.windowspc.com/games_misc/FreecellTool.htm

[6] Windows 95/98 » Games, http://www.bookcase.com/library/software/win9x.games.html

[7] Chellapilla K and Fogel DB (1999) "Evolution, Neural Networks, Games, and Intelligence," Proc. IEEE, Vol. 87:9, Sept., pp. 1471-1496.

[8] Russell, Stuart J. and Norvig, Peter, *Artificial Intelligence A Modern Approach*, New Jersey: Prentice Hall.

[9] J.F. Kolen, A.K. Goel, "Learning in Parallel Distributed Processing Networks: Computational Complexity an Information Content," IEEE Transactions on Systems, Man, and Cybernetics. 21(2), pp 359-366, March/ April 1991.

[10]T. Kohonen, Self-Organizing Maps, 3rd Ed., Springer-Verlag, ISBN 3-540-67921-9, 2000.

[11]Widrow B., Lehr M.A., "Adaptive Neural Networks and Their Applications," International Journal of Intelligent Systems," 8, pp 453-507. 1993.

[12]Demuth H., Beale M., Matlab – Neural Network Toolbox, Users Guide 4.0; http://www.mathworks.com

[13] D.E. Rumelhart, J.L. McClelland, et al., Parallel Distributed Processing – Explorations in the Microstructure of Cognition. Volume 1: Foundations, MIT Press, ISBN 0-262-68053-X, 1986.

[14]R. Fletcher, Practical Methods of Optimization. 2nd Edition. John Wiley and Sons, ISBN 0-471-91547-5, 1987.