Machine Learning Based Heuristic Search Algorithms to Solve Birds of a Feather Card Game

Bryon Kucharski, Azad Deihim, Mehmet Ergezer
Wentworth Institute of Technology
550 Huntington Ave, Boston, MA 02115
{kucharskib, deihima, ergezerm}@wit.edu

Abstract

This research was conducted by an interdisciplinary team of two undergraduate students and a faculty to explore solutions to the Birds of a Feather (BoF) Research Challenge. BoF is a newly-designed perfect-information solitaire-type game. The focus of the study was to design and implement different algorithms and evaluate their effectiveness. The team compared the provided depth-first search (DFS) to heuristic algorithms such as Monte Carlo tree search (MCTS), as well as a novel heuristic search algorithm guided by machine learning. Since all of the studied algorithms converge to a solution from a solvable deal, effectiveness of each approach was measured by how quickly a solution was reached, and how many nodes were traversed until a solution was reached. The employed methods have a potential to provide artificial intelligence enthusiasts with a better understanding of BoF and novel ways to solve perfect-information games and puzzles in general. The results indicate that the proposed heuristic search algorithms guided by machine learning provide a significant improvement in terms of number of nodes traversed over the provided DFS algorithm.

1 Introduction

This research was conducted by an interdisciplinary team of two undergraduate students and a faculty to explore solutions to the Birds of a Feather (BoF) Research Challenge proposed by (Neller 2016). BoF is a perfect-information solitaire game, comparable to FreeCell solitaire.

BoF is played with a standard 52 card deck. Each game begins with a starting deal of sixteen random cards organized into a 4-by-4 grid. The player must select an individual card and move it on top of another card in the selected card's row or column provided that one of the following conditions are met: (1) Both cards have the same suit. (2) Both cards have the same rank. (3) Both cards have adjacent ranks. For example, a King of Hearts can be placed on top of a Queen of Clubs since their ranks are adjacent, or that King of Hearts can be placed on top of a Nine of Hearts since they have the same rank. The game concludes when fifteen moves are made resulting in one stack of all sixteen cards.

Newly-discovered perfect-information puzzles offer a plethora of terrain for exploration and allow for fun and creative solutions. The nature of BoF allows for all possible moves to be organized into a decision tree, which can be traversed through with various types of algorithms. Depth-first search (DFS) and breadth-first search (BFS) are two rudimentary approaches to tree traversal that are straightforward to implement and can solve the game if possible. However, there is still room left for improving their performance using auxiliary algorithms. As part of the challenge proposed by Neller, teams were asked to explore potential heuristics to guide the performance of these graph algorithms. This compelled our team to delve into more intelligent solutions such as heuristic-based traversal algorithms. There are several features that can be extracted from any state of a BoF game to be used directly as a heuristic to guide the traversal. This abundance of applicable features facilitated a machine learning approach: suitable features can be used as an input, and the output can be applied as a heuristic - which allows the traversal to be directed by multiple features rather than just one. The team compared the provided depth-first search to heuristic algorithms such as Monte Carlo tree search (MCTS), as well as a novel heuristic search algorithm guided by machine learning.

The applications of machine learning to solving games have been well studied. IBM engineers have applied linear approaches as well as alpha-beta pruning to solve the game of checkers (Samuel 1959), (Samuel 1967). (Yan et al. 2005) proposed an heuristic approach to the Klondike solitaire that solved twice as many games on average than an expert human player. Machine learning based solutions to FreeCell have also been explored. (Chan 2006) compared the performance of multiple algorithms, including heuristic approaches, neural networks, Bayesian learning and decision trees.

Besides machine learning algorithms, randomized search-based techniques are employed to decrease the number of nodes that are evaluated to solve a game. Alpha-beta pruning (Knuth and Moore 1975) has been the forefront for finite, zero-sum two player games with perfect information. (Coulom 2006) introduced a new algorithm, Monte Carlo tree search, which combines Monte Carlo methods with tree search to solve these types of games. MCTS has been widely used in solving various games, including a 9 x 9 computer Go program named Crazy Stone (Coulom 2007), Othello (Nijssen 2007), and Tic Tac Toe (Auger 2011). Google's DeepMind employed a combination of Deep Learning and

MCTS to create AlphaZero, an algorithm that mastered Chess, Shogi, (Silver et al. 2017a), and 18 x 18 Go (Silver et al. 2017b).

MCTS has also been implemented in single player games such as Sudoku (Cazenave 2009) and SameGame (Schadd et al. 2008). In our research, MCTS is applied to Birds of a Feather, a finite, single player game with perfect information.

The paper is organized as follows: Section 2 provides a brief introduction to heuristic search algorithms employed in this research. Section 3 defines the proposed machine learning based heuristic search algorithms. Section 4 introduces the experimental settings and the evaluation criteria. Section 5 discusses the empirical results obtained. Finally, Section 6 states the conclusion and suggests future work.

2 Search Algorithms

This section reviews the provided solution to the BoF challenge and introduces the heuristic search algorithms employed in the proposed solution.

Depth-first search

Depth-first search is an algorithm that allows for simple traversal through graphs. The nature of DFS is that it visits each node once and only once by beginning at the root and exploring as far down the tree as possible before backtracking. DFS can be used to solve Birds of a Feather because all possible moves can be organized into a decision tree. Any solution to a solvable deal of Birds of a Feather must reside in the deepest level of the tree, which makes DFS a desirable method of traversal.

Another algorithm that can be used to solve Birds of a Feather is breadth-first search, which prioritizes finding new paths by searching all the node's children prior to advancing to the next level of the tree. This method of traversal would be undesirable because it would search every node in the top fifteen levels before reaching the final level of the tree where every possible solution must reside.

Linear Regression

Linear regression (LR) is a supervised learning algorithm employed to forecast continuous outcome. It generally requires a structured array of real numbers, x and predicts vector of real numbers, y.

The predicted output, $\hat{y}$ relies on a hypothesis function, $h(x)$.

$$y \approx \hat{y} = h(x).$$

For LR, the hypothesis function, $h(x)$ yields a line that best fits the training data:

$$h(x) = \sum_{i=0}^{n} w_i x_i + b$$

where n is the number of data points in our data-set. We can then design a cost function to minimize the difference between the current prediction and the true label:

$$E_l = \sum_{i=0}^{n} (y_i - (w_i x_i + b))^2$$

To minimize the error, the partial derivative of E_l can be set to 0 with respect to both variables. A closed form solution to the above equation is:

$$w = \frac{\sum_{i=0}^{n}(x_i - x_m)(y_i - y_m)}{\sum_{i=0}^{n}(x_i - x_m)^2}$$

$$b = y_m - w x_m$$

where x_m and y_m are the means of the x and y vectors, respectively (Abu-Mostafa, Magdon-Ismail, and Lin 2012).

The LR model yields a continuous value $\hat{y} \in \mathbb{R}^1$. In our case, y represents solvability of a given game state, and LR predicts if the game is solvable or not. Thus, a higher value for $\hat{y}$ indicates a larger confidence in solving the game, and a lower value suggests a potentially unsolvable game.

Monte Carlo Tree Search

MCTS is a tree search algorithm that computes the best move to make in a given state. The algorithm consists of a selection phase, expansion phase, simulation phase, and a back-propagation phase (Coulom 2006). In the selection phase, the algorithm starts at the root of the tree and traverses down fully expanded nodes until a leaf node that is not fully expanded is encountered. In the expansion phase, a child of the leaf node that has not been visited is simulated n number of times in the simulation phase. The simulation produces an outcome, which is then propagated back to the root node.

3 Machine learning based heuristic search algorithms

BoF Research Challenge provided an implementation of DFS as well as a sample script for generating solvability data. Our team leveraged this file and created addition features to collect data across 10,000 seeds. The data was split into test and train sets, and models were created as part of a heuristic search algorithm. Every iteration of the heuristic, the features were generated for the current node. The probability of solvability is predicted using a linear regression model for every child of the current node, then rearranged from highest probability to lowest probability. The highest probability child is selected first as the move to make, and the heuristic repeats. If a solution is not found, the second highest probability child is selected until the game is solved or all the nodes are exhausted. A priority queue was employed to implement this behavior. The LR solvability predictions were set as the priorities for the nodes and the game would pull the highest priority element to advance.

Evolution of search

We started the algorithm development with a higher number of features and a more complicated prediction model. We trained various models to find a trade-off between the number of features generated and model sophistication. We managed to reduce the number of features and model complexity with minimal compromise in algorithm performance. Below, we discuss some of our findings during this research.

Table 1 lists the train and test accuracy scores of various machine learning models on the same feature set. These models are covered in an introductory AI elective taught at the team's university (Ergezer, Kucharski, and Carpenter 2018) and can be implemented with ease using SciKit (Pedregosa et al. 2011).

Model	Train / Test Accuracy (%)
Logistic regression	75.48 / 75.41
KNN	71.17 / 70.10
Decision trees	74.81 / 74.75
Random forest	73.90 / 73.97
AdaBoost	74.30 / 74.23
Gaussian naive Bayes	71.48 / 71.54
Neural networks	75.36 / 75.17

Table 1: Train and test accuracy scores of various machine learning models using the six BoF features

Some of these models' performances, such as the neural networks (NN), are sensitive to their tuning parameters. In order to achieve a reasonably high performance, we conducted a separate search through their tuning parameters. For the case of NN, we swept through four α values: $[0.0001, 0.01, 1, 3]$ and four sets of hidden layers, each with 100 neurons for each α value: $[1, 3, 6, 12]$. The tabulated results are the best of these 16 candidates. Future work can involve employing evolutionary algorithms to optimize the set of features as well the model parameters (Neller et al. 2010) and (Ergezer, Simon, and Du 2009).

Since none of the tested 16 NN architectures outperformed regression, we settled for a linear model to minimize the model complexity. We decided to add more flexibility to our model by transforming the existing features. We transformed the six final features into a new feature matrix consisting of third order polynomial combinations of these features. Thus, the number of features expected by the model increased from six to 83 with minimal added cost of time and computational complexity.

Table 2 lists the results of adding the polynomial features to the logistic regression model. The first column represents the range of cards left in the game for the train and test data. We developed a different model for each range given in this column. The table allows us to evaluate the consequence of extending the features as a function of number of cards left in the game.

Cards	Features	Train / Test Accuracy (%)	Features	Train / Test Accuracy (%)
0 - 16	6	73.60 / 73.47	83	76.66 / 76.58
8 - 16	6	75.78 / 75.95	83	76.11 / 76.04
16	6	99.63 / 99.84	83	99.63 / 99.84

Table 2: Train and test accuracy scores of models with various game states before and after adding polynomial features

Based on Table 2, we observe that in the most general case, when the model consists of all possible number of cards, transforming the number of features with the third degree polynomial increases the train and test accuracy scores by about three percent. On the other hand, in the most specific case, when the model considers only a new game, extending the number of features does not benefit the model accuracy. None of the cases' performance deteriorates due to transformed features. Based on these results, we decided to transform our features and expanded input to all models.

The Proposed Algorithm

A variety of algorithms using the heuristic and DFS were compared in our research (Kucharski, Deihim, and Ergezer 2018). A heuristic only model, labeled as LR, uses data collected across every number of card possibility (1 to 16) to train a single model. Multiple novel algorithms which combined both linear regression and DFS were also evaluated. LR8+DFS is an algorithm that uses a trained model when the number of cards is eight or above, and a DFS when the number of cards is below eight. LR16+DFS does the same, but the model is trained and used on inputs with only 16 cards. Finally, the algorithm titled LR+DFS combines four LR multiple models with DFS. One model is trained on data with 16 and 15 cards, one model with 14 and 13 cards, one with 12 and 11 cards, and another with ten and nine cards. DFS is employed when the number of cards is below nine. When running the algorithm, the model used to predict the probability of each child is selected based on the number of cards present in the game.

The combination of heuristic search and DFS was implemented because the benefit of the heuristic models diminished with a lower amount of cards. Tables 3 and 4 list the correspondence between solvability and accuracy, where solvability is the percentage of training data that is solvable. Higher solvability percentage correlates to a higher accuracy for models with high amount of cards, and a lower solvability corresponds to lower accuracy. The use of DFS for situations with fewer cards may mitigate inaccuracies in cases with lower number of cards and allow for the algorithm to find a solution quicker. Furthermore, combining multiple models may prevent underfitting since there is less data about nodes with a higher number of cards. Separating their models may reduce the need to create a more sophisticated model with a higher computational cost. Multiple models also allow for higher accuracy regardless of the number of cards in a node.

The following six features were generated to be utilized with the heuristic search algorithms:

– Number of Valid Moves - number of adjacent ranks or same suits in the same row or column
– Number of Pairs - number of adjacent ranks or same suits, not necessarily in the same row or column
– Suit Most - number of times the most frequent suit appears
– Rank Most - number of times the most frequent rank appears
– Ratio of number of moves per number of cards left
– Ratio of number of pairs per number of cards left

Cards	Solvability (%)	Train / Test Accuracy (%)
16 and 15	95.21	95.33 / 94.85
14 and 13	83.97	85.10 / 84.70
12 and 11	59.53	72.84 / 73.02
10 and 9	32.71	73.21 / 73.61

Table 3: Accuracy of models for LR+DFS algorithms

Cards	Algorithm	Solvability (%)	Train / Test Accuracy (%)
All Cards	LR	46.29	76.66 / 76.58
8+ Cards	LR8+DFS	52.32	76.11 /76.04
16 Cards	LR16+DFS	99.68	99.63 / 99.84

Table 4: Accuracy of models for single model algorithms

MCTS Solution to BoF

Besides the LR, performance of a randomized tree search was also evaluated. The MCTS algorithm has a selection, expansion, simulation, and back-propagation phase as discussed in Section 2. For BoF, the selection phases follows a tree policy using the upper confidence bound (UCT) introduced by (Kocsis and Szepesvári 2006). The next node to traverse is selected by maximizing

$$UCT(v) = \frac{q_v}{n_v} + c\sqrt{\frac{\log N_v}{n_v}}$$

where q_v is the number of wins subtracted from number of losses for current node v, N_v is the number of times the current node v has been visited, n_v is the number of times each child node of v has been visited, and c is a constant to control exploration versus exploitation.

During the simulation phase, a random policy is used where random children are selected from node v until a terminal state, where an outcome of 1 was assigned for a win, or the goal node was found, and -1 was assigned for a loss, or no more children were available to select. At the end of each simulation, the values of q and N are back-propagated for every child until the root of the tree is reached.

Given a new state s, this tree policy, random policy, and back-propagation are repeated for 160,000 simulations to select the ideal move to make. Selecting the ideal action to take is repeated until the game is in a terminal state. For BoF, the probability of randomly selecting children until a win is low requires a relatively large number of simulations to find the best child node.

4 Empirical evaluations

To test the performance of the proposed algorithms, 50 seeds were randomly selected. These seeds were used to control the randomness of the deals generated. On occasion, there are outlier seeds that have very few paths to a solution, making search time extremely long - using the same seeds maintains uniformity across each algorithm. The same seeds were used to test each algorithm, allowing each algorithm to solve the same 50 deals. If each algorithm were to be tested on different seeds, the performance of each may be compromised by the number of outlier seeds each algorithm is tested on.

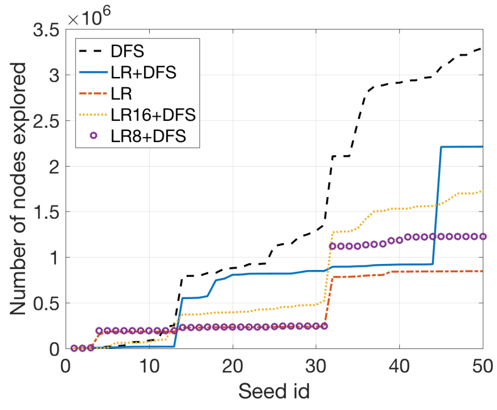

Figure 1: Running total of number of nodes explored by each algorithm for the 50 random seeds. DFS is depth-first search, LR+DFS is four LR models for greater than eight cards and DFS for the rest, LR is solely one LR model for all game states without a DFS, LR16+DFS is one LR model for when the game starts and DFS for rest of the moves, LR8+DFS is one LR model for greater than eight cards and DFS for less.

In order to compare the algorithms, they must all be evaluated using the same metrics. As the algorithms searched through each seed for a solution, the number of nodes visited and the amount of time taken to find a solution were tracked. These were the metrics used to determine the effectiveness of each algorithm. The median, mean, and standard deviation were then calculated to better understand how each algorithm performed on a seed-by-seed basis. The ideal algorithm should be able to find a solution with minimal traversal and relatively fast computation time.

Simulations were executed on a server with 32 2.0 GHz Intel Xeon CPU E5-2640 v2 processors. Data generation was performed using Java and model creation and deployment were accomplished with Python.

5 Experimental results

Figure 1 illustrates the running total of the number of nodes explored by each algorithm for the 50 random seeds. The smaller number of nodes explored, the better. All four of the tested heuristic algorithms consider less number of nodes than the provided DFS to solve the game when possible. Employing LR with a single model for all number of cards and without DFS finishes 50 games with the least number of nodes exploration. This indicates the success of the proposed heuristics: sorting by solvability prediction. Special attention should be paid to LR16+DFS where we deploy one LR model for the first move when the game starts and DFS for all of the other moves. LR sorts the nodes at the beginning of the game from most to least likely to be solvable. Based on Figure 1, running this heuristic once is sufficient to boost the performance of DFS.

Table 5 presents the median, mean and standard deviation of the number of nodes explored by each algorithm, respectively. The median, mean and standard deviation of number of nodes explored by all of the proposed algorithms were less than DFS. The largest decrease in the median node count was observed between LR only and DFS as 88.60%. The largest decrease in the mean node count was also observed between LR only and DFS and is 74.37%.

Algorithm	Median	Mean	Std
DFS	1.43e+04	6.71e+04	1.43e+05
LR+DFS	1.79e+03	4.51e+04	1.97e+05
LR	1.63e+02	1.72e+04	8.01e+04
LR16+DFS	5.39e+03	3.53e+04	1.09e+05
LR8+DFS	4.44e+02	2.50e+04	1.27e+05

Table 5: Statistics on number of nodes explored for 50 random seeds by the tested algorithms.

Table 6 presents the median, mean and standard deviation of the time required by each algorithm to solve 50 random BoF seeds, respectively. The median time consumed by all of the proposed algorithms except LR was less than DFS. However, mean and standard deviation time exhausted by most of the heuristic algorithms was greater than DFS. This table illustrates that even though heuristics algorithms visit less number of solutions, they consume more time to play the game. Further analysis is needed to understand if this is due to the time required by the LR model for prediction or sorting the LR predictions with a priority queue. Furthermore, the code was implemented in a higher-level language to prototype the heuristics rapidly and to emphasize ease of development and collaboration among research members.

Algorithm	Median (s)	Mean (s)	Std ($\pm$s)
DFS	1.10	4.82	10.12
LR+DFS	0.75	42.17	229.34
LR	1.23	47.06	219.38
LR16+DFS	0.68	2.91	8.09
LR8+DFS	0.85	25.58	164.06

Table 6: Statistics on time to solve 50 random BoF seeds by the tested algorithms.

A simulation of MCTS was performed across 30 random seeds over a span of 11 hours and 35 minutes, or an average of 23.16 minutes per seed. The goal node was found for every seed in 15 node selections, or a total node count of 450. MCTS introduces a compromise between computation time and number of nodes to the goal. While the computation time was high compared to the other tested heuristic algorithms, the node count was significantly lower. Thus, a combination of MCTS with a model-based heuristic may result in a better trade-off between time and accuracy. Future work can combine both approaches such that the heuristic algorithm executed first until a certain number of cards is reached, then switch to MCTS for the rest of the game. Since there is a larger state space with more child nodes at the beginning of the game, running a heuristic first would provide a quicker way to traverse through the tree to start. MCTS should provide a strong finish to reach the end in the shortest number of nodes and will not take as long to run since there are not as many simulations towards the end of the game. In contrast, the opposite approach can be taken to execute MCTS first to provide a confident path to start, and then a heuristic may take over to quickly reach the goal node. This approach would take longer to run MCTS at the beginning of the game, but may result in higher accuracy.

6 Conclusions

In this paper, we presented various solutions to solve the BoF Research Challenge. We were successful in designing and implementing algorithms such as Monte Carlo tree search, a heuristic search algorithm using linear regression, and hybrid search algorithms that incorporate linear regression and depth-first search. Through extensive testing, we were then able to compare the efficiency of each algorithm by studying the time taken to reach a potential solution as well as the number of nodes visited before a solution was reached.

The simulations performed indicate that the solely heuristic algorithm, on average, is 74% more efficient than DFS in terms of nodes visited at the cost of computation time. This increase in time can be attributed to two factors: (1) the implementation of our heuristics using the built-in data structures in Python, and (2) the time to compute the features and predict using our machine learning models. MCTS is shown to find the solution with the shortest possible number of nodes when provided with enough simulation. But this comes at an extremely high computational cost. LR16+DFS is presented as the recommended approach since it minimizes the computation cost by executing the machine learning model only once at the beginning of the game, yet it still reduces the number of nodes visited by DFS to find a solution.

In future work, a speedup in computation time may be achieved by optimizing the sorting of the predicted probabilities as priorities or by using a different programming language. Other MCTS variations or combining MCTS with a heuristic could be explored to lower the computation time without sacrificing as much efficiency in node count. In addition to optimizing the implementation of the proposed algorithms, our research could be expanded by employing an evolutionary algorithm to design the machine learning models where the tuning parameters are selected based on minimizing a cost function that considers the number of nodes visited and the corresponding time complexity.

References

Abu-Mostafa, Y. S.; Magdon-Ismail, M.; and Lin, H.-T. 2012. *Learning from data*, volume 4. AMLBook New York, NY, USA:.

Auger, D. 2011. Multiple tree for partially observable monte-carlo tree search. In *European Conference on the Applications of Evolutionary Computation*, 53–62. Springer.

Cazenave, T. 2009. Nested monte-carlo search. In *IJCAI*, volume 9, 456–461.

Chan, C. 2006. Helping human play freecell.

Coulom, R. 2006. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, 72–83. Springer.

Coulom, R. 2007. Monte-carlo tree search in crazy stone,".

Ergezer, M.; Kucharski, B.; and Carpenter, A. 2018. Curriculum design for a multidisciplinary embedded artificial intelligence course. In *49th ACM technical symposium on computer science education*.

Ergezer, M.; Simon, D.; and Du, D. 2009. Oppositional biogeography-based optimization. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, 1009–1014. IEEE.

Knuth, D. E., and Moore, R. W. 1975. An analysis of alpha-beta pruning. *Artificial intelligence* 6(4):293–326.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293. Springer.

Kucharski, B.; Deihim, A.; and Ergezer, M. 2018. Birdsoffeatherpython. https://github.com/bryonkucharski/BirdsOfFeatherPython.

Neller, T. W.; Fisher, A.; Choga, M. T.; Lalvani, S. M.; and McCarty, K. D. 2010. Rook jumping maze design considerations. In *International Conference on Computers and Games*, 188–198. Springer.

Neller, T. W. 2016. AI education: birds of a feather. *AI Matters* 2(4):7–8.

Nijssen, J. 2007. Playing othello using monte carlo.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. 2011. Scikit-learn: Machine learning in python. *Journal of machine learning research* 12(Oct):2825–2830.

Samuel, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of research and development* 3(3):210–229.

Samuel, A. L. 1967. Some studies in machine learning using the game of checkers. ii-recent progress. *IBM Journal of research and development* 11(6):601–617.

Schadd, M. P.; Winands, M. H.; Van Den Herik, H. J.; Chaslot, G. M.-B.; and Uiterwijk, J. W. 2008. Single-player monte-carlo tree search. In *International Conference on Computers and Games*, 1–12. Springer.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017a. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017b. Mastering the game of go without human knowledge. *Nature* 550(7676):354.

Yan, X.; Diaconis, P.; Rusmevichientong, P.; and Roy, B. V. 2005. Solitaire: Man versus machine. In *Advances in Neural Information Processing Systems*, 1553–1560.