Creating a Dominion AI Using Genetic Algorithms

Mok Ming Foong

Abstract

Dominion is a deck-building card game. It allows for complex strategies, has an aspect of randomness in card drawing, and no obvious optimal solution, making a good candidate for a machine learning study. Prior works involve using genetic algorithms in conjunction with an artificial neural network (ANN), but these used many features and do not explore the significance of some features based on a player's understanding of the game. The purpose of this project is to investigate if it is possible to create a strong agent from the game using less features based on prior knowledge. In this project, a genetic algorithm is used to create an agent to play the game. Its chromosomes serve as a guide when the agent makes decisions in the Buy Phase of the game. Candidates in a generation are pit against each other in a round robin, and the number of wins serve as a measure of fitness and are used in determining the next generation. The candidates were also pit against a benchmark candidate to measure their fitness.

Introduction

Dominion is a deck-building card game. While numerous studies have been done on games such as Chess and Go, little emphasis is placed on modern games such as Dominion. Games such as Dominion allow for complex strategies and multiple playstyles, which may serve as a useful testbed for various machine learning approaches. The presence of circularities, where one type of deck has an advantage over another in a circular manner, also means that there is no easily obtainable optimal solution for this problem.

Deck building games usually involve two key decision points in two of the game's key phases.

- Action Phase - Involves solving the problem of how to best play the hand for the current turn
- Buy Phase - Involves solving the problem of how best to build the deck/acquire new cards to optimize it for future turns

For this project, the approach used to create an AI for this game was done using genetic algorithms, but only for the deck building portion of the game (Buy Phase). A chromosome consisted of 3 x 13 positive integers, where 3 integers were used to represent 3 parameters of a specific card in the game. Using these chromosomes as a guide, the agent will aim to build towards a target deck.

Using a fixed card playing strategy across all candidates for the Action Phase, each candidate will play against every other candidates. Based on the number of wins of each candidate, genetic operators are then applied on the chromosomes of these candidates. The fitness of each candidate is also measured against a previously generated candidate from another experiment run.

Related Work

Little prior work has been done on Dominion as a machine learning study. 2 previous work which use both genetic algorithms and an Artificial Neural Network (ANN) to train a dominion AI. 4 of them use an ANN. The complexity of decision making due to the number of card effects in the game and the numerous considerations a real player would have to make is one of the key reasons why a neural network is used. The genetic algorithm serves as a good optimization solution for a problem such as Dominion, where the presence of circularities mean that there exists no single optimal solution, and different candidates are strong or weak depending on their opponent. All prior works usually involves simulating the candidates in a round robin tournament and making use of the number of wins of each candidate to measure fitness. However, each work also applies a different method of calculation for the fitness of a candidate. Many different features and neural network designs are investigated

across all the works and there appears to be no optimal model.

Features

A reduced number of cards were used in this machine learning study for a simpler implementation of the game while still allowing for development of complex strategies. 3 Gold Cards, 3 Victory Cards, and 7 Kingdom Cards were used in the game. Each card has its own cost and card effect (Refer to Appendix)

The 3 Gold Cards used are Copper, Silver and Gold.

The 3 Victory Cards used are - Estate, Duchy, Province.

The 7 Kingdom Cards used are - chapel, village, woodcutter, smithy, market, laboratory, festival. Each has a different card effect.

Each card makes up 3 positive integers in the chromosome, giving 39 parameters in the chromosome for these 13 cards. These parameters correspond to –

- Preference of buying the card over other cards. A higher preference meant that the candidate would buy that card over other cards
- Turn delays before buying of the card. Turn delays are the number of times that the opportunity to buy the card is skipped. This is to prevent the AI from buying cards that are only important at the end of the game, at the beginning (e.g. Victory Cards)
- Card limit in deck. Where possible, the agent should attempt to limit the maximum number of this card to this value.

These parameters are created based on a personal understanding of the important considerations required in the game. The chromosomes were randomly initialized, with constraints set to some of the parameters.

Each candidate was then pit against the other candidates in a round robin, and the distribution of wins was used to determine the next generation.

Methods

The Dominion agent plays the Action and Buy Phase in two different ways. The significantly different decision making process of these two phases make it easier and meaningful for these two phases to be considered separately. The Action Phase will be played according to a fixed set of rules, while the Buy Phase will be played using the chromosome as a guide. The scope of the genetic algorithm is only limited to the Buy Phase for simplicity and ease of implementation.

Action Phase

For the Action Phase, the agent plays the game according to a set of rules. The fixed strategy consists of these rules –

- Cards in the hands are played in the order that would maximize the expected Gold value in their turn. Permutations of all possible card playing orders are calculated, and the order which results in the highest expected Gold value is used. This is to ensure that the agent is able to consistently buy/acquire better cards.
- Expected Gold value is calculated based on the current Gold in hand and the average Gold per card in the deck

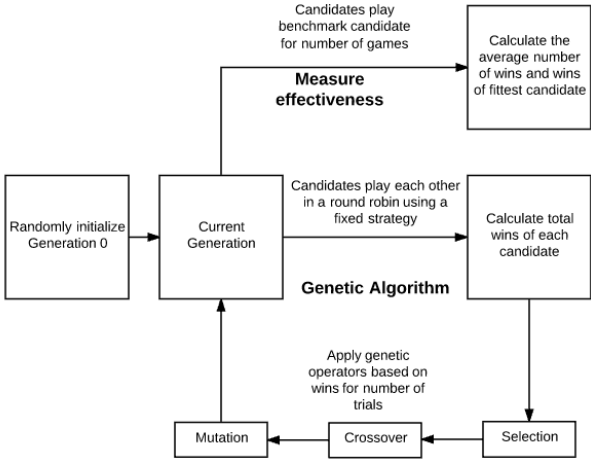
Under these conditions, it is possible that the simulation will take unnecessarily long due to inefficient decision making by the agent. A turn limit of 100 per game is set. This is more than double the turns required by a human player, and is sufficient for the simulation.

Buy Phase

For the Buy Phase, the parameters in the chromosome are used as follows.

- Cards would be bought/added to the deck based on a preference score in the chromosome. Should scores be equal, the card bought is randomly selected.
- The buying of a card is constrained by the card limit and turn delay parameter.
- The turn delay parameter is reduced by 1 when the opportunity to buy the card is skipped, down till a minimum value of 0. Only when it reaches 0, will the agent buy the card.

Using these rules, the genetic algorithm is carried out as follows.



Pipeline of genetic algorithm

Initialization

For the 3 types of parameters mentioned above,

- Preference of buying cards was initialized with integers from 0 to 5
- Turn delays was initialized at 0.
- Card limit was initialized from 0 to 6

These values are based on a personal understanding of what a general good range of values might be in the game. This was done to speed up the growth of the candidates, and would have little effect to the convergence of the solution given their small values relative to the change in the mutation phase.

Evaluation

All candidates then played each other candidate in 3 simulated matches in a round robin manner using a fixed strategy.

Each candidate then has its number of wins across all matches calculated. This value will be used with the genetic operators.

Since the number of wins of a candidate is significantly affected by the fitness of other candidates, there needs to be another metric such that the effectiveness of the algorithm can be measured. For this, the fittest candidate of another run of the experiment will be used as a benchmark.

Each candidate will play against the benchmark in 50 simulated games. The number of wins will be used as a measure of effectiveness.

Selection

The two candidates with the highest number of wins are kept as elites, and will not be altered by any of the operators below.

Crossover

Each candidate will be generated via uniform crossover of two randomly selected parents.

To ensure that fitter parents are used in the crossover, a sample space with duplicate parents is created. The number of duplicated candidates is proportionate to the number of wins the candidate has. The two parents are then selected from this sample space to form the next generation.

Mutation

Then, 20% of the new generation will be mutated. For mutation, candidates will be randomly selected, and have a random parameter in the chromosome changed by a random value of 1 or -1. All random selections are done such that each

outcome has an equal chance of being selected.

Experimental Setup

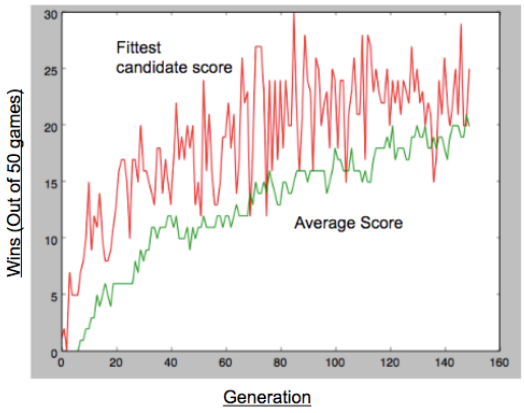
The experiment was carried out with 35 candidates across 150 generations. There is a high number of generations compared to candidates. This is a workaround for the low number of candidates due to the slow performance of the algorithm, as many games have to be simulated among all candidates in a round robin. These values were chosen after several rounds of testing, and gave a good balance of attaining convergence, learning and performance.

In the round robin, 3 games are played between the candidates, and 50 games are played against the benchmark candidate. The benchmark candidate was generated from an experiment run with all parameters kept the same, but with only 50 generations. A low number of games are performed to speed up the algorithm. It is possible that with a larger number of candidates, less games need to be played between each pair of candidates.

In the genetic algorithm, 2 out of the 35 candidates are kept as elites, while the mutation rate is set at 0.2. Crossover is uniform. The value for elitism is arbitrary, with a constraint set at less than 10% of the number of candidates in a generation. The mutation rate is arbitrary.

Results

The algorithm has mostly converged after 150 generations, resulting in candidates that can achieve 20 wins out of 50 against the benchmark on average, and have intersected with the fittest candidate. The fittest candidate is also able to beat the benchmark about 50% of the time. It is possible that running more generations might result in an increase in the performance of the fittest candidate, but these results are sufficient to show the algorithm's effectiveness.



Score of fittest candidate and generation average when facing benchmark candidate

Discussion

It appears that the algorithm is performing well, with the fittest candidate being able to beat the benchmark for the first time after 54 generations despite only 35 candidates per population.

There is a large variance for the scores against the benchmark candidate. Apart from the random nature of card draws, this is likely due to the fact that only 50 games were played against the benchmark candidate. It is likely that a smoother curve will appear should more games be played, as seen from the average score curve.

Due to the low number of candidates in a generation, it is possible that there are genotypes that are unaccounted for even though the candidates have been through 150 generations. In addition, it is also possible that the solution converges to a local maxima given that the sample space for the crossover was biased towards candidates with higher wins.

Due to the benchmark being generated with the same algorithm as the experiment, it is possible that the fittest candidate from the experiment and the benchmark candidate are solutions that revolve about the same local maxima, and may not be effective at all relative to the global maxima. There should ultimately be a benchmark that is generated separately and based on a commonly

known good strategy.

Conclusion

It appears that the genetic algorithm is a good method for creating an agent to play Dominion, as seen from the experiment and prior work. From the experiment, it seems that basing the model only on few key features is sufficient to create an agent that can play Dominion. However, as mentioned in the discussion, a different benchmark should be used to measure the effectiveness of the algorithm. One suggestion is to implement the "Big Money" strategy, which is a well-known beginner strategy that is able to perform well with the cards used in the experiment. Multiple benchmarks can also be implemented to investigate for the presence and effect of circularities.

In addition, with more computational resources, there should be a significant increase in the number of candidates in a generation. This should allow the algorithm to better converge to a global maxima. Also, the round robin can be changed to reduce the number of games. One possible implementation is to only pit the candidates against 10% of the other candidates which are randomly sampled.

Lastly, since optimize plays for the current turn (Action Phase) is an equally important part of the game, there should definitely be another agent created to handle the decision making in the Action Phase, or both phases. More cards can be added into the experiment to allow for a closer representation of the game as well.

References

1. Fynbo, R. B., & Nellemann, C. S. (2010). Developing an agent for dominion using modern ai-approaches. *M. Sc. IT, Media Technology and Games (MTG-T) Center for Computer Games Research*.
2. Glimsdal, S. (2015, April). AIs for Dominion Using Monte-Carlo Tree Search. In *Current Approaches in Applied Artificial Intelligence: 28th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2015, Seoul, South Korea, June 10-12, 2015, Proceedings* (Vol. 9101, p. 43). Springer.
3. Jansen, J. V., & Tollisen, R. (2014). An AI for dominion based on Monte-Carlo methods.
4. Mahlmann, T., Togelius, J., & Yannakakis, G. N. (2012, June). Evolving card sets towards balancing dominion. In *2012 IEEE Congress on Evolutionary Computation* (pp. 1-8). IEEE.
5. Dominion Strategy. (n.d.). Retrieved December 16, 2016, from https://dominionstrategy.com