Game Strategy Learning using Genetic Algorithms

Nathan Myers

December 13, 2007

Abstract

In this paper I will be examining a subset of Artificial Intelligence known as Genetic Algorithms, and how they can be applied to game strategy learning. I will first examines the general paradigms of AI systems, and then move into the specifics of machine learning and genetic algorithms. I will then use a simple card game with implementation details as an example of how genetic algorithms may be applied to create a both a stand alone strategy generator and a generator that creates strategies tailored to a specific play-style.

1 Introduction

In most games in which one plays against the AI, the AI is a handcrafted set of routines created to play the game optimally. While these types of AIs are typically pretty solid, they are unable to learn or grow overtime, fostering predictability and diminishing the enjoyment that a player may experience when playing against the AI.

Algorithms that are able to generate strategies have a greater potential for producing strategies that are able to adapt to different player's habits and play-styles. While strategies generated this way are unlikely to be as effective as a hand crafted strategy, if even decent results can be achievable algorithmically, it could dramatically increase the replay value of a game.

2 A Brief Review of Artificial Intelligence

An AI, or an agent, is at its core, a mechanism for choosing from many possible solutions an optimal solution to reach what is known as a goal state. To this mean it may employ a number of strategies, most often state based searching. However in nearly all cases a complete state based search to find the optimal solution is unacceptably expensive, if not impossible. So the field of AI is less about the search itself, and more about tactics and schemes designed to prune and direct ones traversal through the states. One mechanism of performing this is creating a complete mapping from states to actions, which is known as a policy[14]

An AI may also contain what is known as a knowledge base, which allows it to collect and 'remember' information about the world. This information enables the AI to make educated guesses about things it currently does not have enough information regarding. Note that this does not necessarily that it learns anything from this data.

3 MinMax

One method of deriving an effective strategy is to use a search method known as Minmax. Minmax operates by searching through all of the states that may be derived from the current state, and identifying alternatively the worst and best possible states that may be chosen until the algorithm reaches the leaves of the state tree. The algorithm then works its way back of the tree determining what the best path to take is base on the worst possible (or best from the perspective of the opponent) move is. This is one of the more basic methods in which a search may be carried out.

4 Machine Learning

Machine Learning is the subset of artificial intelligence that is concerned with the autonomous learning aspect of an agent. This can be done in various ways, one mechanism involves keeping a knowledge base and making some form of reasoning from the knowledge base and from external inputs. The knowledge base can be kept in a number of ways, it can be a set of facts about the world kept in the form of sentences, or it can be kept as a set of policies on how to interact to a set of stimuli.

Within the realm of machine learning, there are two schools of methods of deriving knowledge of the world from external stimuli: inductive and deductive. Inductive learning creates generalized statements derived from observations, for example an external stimulus of "This grass is green" might create the statement "All grass is green". Inductive learning requires no prior knowledge and relies on statistical inference, however it can be easily mislead to over generalize if there exists scarce data. Conversely stands deductive learning which seeks to create general statements that fit the domain theory, for example deductive learning might take the statements, "All men are mortal" and "Socrates is a man" to formulate "Socrates is mortal".

Also within machine learning there exists several other attributes that a learning function may possess. One is whether the algorithm is supervised or unsupervised. Supervised algorithms are given a predefined set of labeled outputs in which it labors to sort inputs into. Unsupervised learning lacks these predefined labeled outputs. In addition, the input gathered by the algorithm may be accurate, or it may be distorted with random noise, and can vary greatly in quantity. Different algorithms are obviously more or less sensitive to both the amount and quality of data in regards to their peers. Also important to the environment is whether or not it exhibits deterministic, or completely predictable, behavior. Learning algorithms tend to function much more slowly in non-deterministic environments.

Also important is whether or not the environment is fully visible to the agent or not. Chess is an example of a fully visible environment, at any point the agent knows the exact locations of all the pieces in play on both sides, as opposed to a partially visible environment such as one a robot mapping the layout of a room would encounter.

5 Genetic Algorithms

Genetic Algorithms are the subset of machine learning that are modeled after evolution and natural selection. As in evolution, a genetic algorithm starts with set of randomly generated policies, each policy is known as a species. The set of species as a whole is known as a generation. Each generation is then evaluated according to a set of standards known as the fitness function to determine how effective each species is. The higher an individual species scores in relation to its peers, the more likely it is to mate and produce offspring that will enter into the next generation. Thus the more fit parents affect the overall direction and fitness of the next generation more strongly than the weaker parents (if they get to affect it at all).

In order to be effective in a genetic algorithm, one must be able to accurately model the problem space within a genetic structure, and also must be able to produce an algorithm for calculating the relative fitness of a genetic structure. Once these things are provided, the algorithm seeds a template species and begins making random modifications to its components (known as genes). Each generation a subset of the species are selected which are then possibly modified and recombined to form the next generation, this is known as cross-over. This process repeats until either a maximum number of generations is reached, or a target fitness is achieved.[10]

5.1 Components

A genetic algorithm contains two critical components that strongly affect it's overall potential:

1. Species Development - The first hurdle in creating a genetic algorithm for a given problem is to create a template to represent a given policy. All aspects of the policy must be represented as a data structure, which could be a simple string of numbers, or a complex table values. Care must be taken to reduce the size of the species as much as possible by removing redundancy so that the algorithm may converge faster. Moreover, the species must be divided into distinct fields that may be crossed to create new valid fields.

2. Fitness Function - Secondly an accurate fitness function must be developed. The purpose of the fitness function is to evaluate a species and assign it a numeric value based on its performance. It is critical to the algorithm that the fitness function is as accurate as possible, because it is directly responsible for setting the chances that a species has to procreate and graduate in some form to the next generation. If it is not

consistent or accurate, it's poor performance can seriously cripple the potential of the algorithm.

5.2 Performance

Genetic algorithms perform well in situations where these two components are easily maximised. Algorithms whose problem space can be easily represented fully with a small amount of information are able to have more efficient and faster cross-overs (since there is less information to swap) and more prominent mutations, allowing them to evolve more quickly. Focus should also be placed on allowing the species to be as flexible as possible[15].

It should be noted that while in most cases the fitness function implementation is straight forward, in situations where the fitness function is non-deterministic, in which the outcome is somewhat random, several instances of the fitness function should be run per species and averaged in order to even out the randomness and create a more reliable fitness function. Doing so however results in a linear multiplication of the fitness function's time complexity.[15]

5.2.1 Simulated Annealing

Simulated Annealing is a technique that is used to allow an agent to escape local maximas so that it may discover more global maximas. In an example of simulated annealing, instead of always choosing the best move, the agent will always pick a random move. This move is then evaluated to determine whether it improves or worsens the situation. If the move improves the situation, it is always executed; otherwise it is executed with a probability under 1, which decreases exponentially based on how much the move worsens the situation. This is modified by a value T, which at high values, is more forgiving towards bad moves, and as it moves closer to zero makes the algorithm accept only positive moves. T is generated internally based on the number of cycles completed by the algorithm. The theory is that given a high enough T that lowers slowly enough, a given agent will always find the global maxima.[10]

Simulated annealing is represented in genetic algorithms in the form of random mutations. When mutated, a species will undergo a number of random changes to it's genes. Like simulated annealing, this should be set so that a high amount of mutation occurs early on in the evolution, and the tapers off as more generations are completed. Like generating the initial set of species, care must be taken to ensure that the mutations result in a valid species.

6 The Game

The card game chosen plays as follows: Two players each hold an identical hand of cards whose values range from 1 to N, while a third identical set of cards is shuffled and placed face down to be used as a flop. Each round a card is withdrawn from the flop stack to be bid upon by both players simultaneously. The player with the higher bid wins the card, and

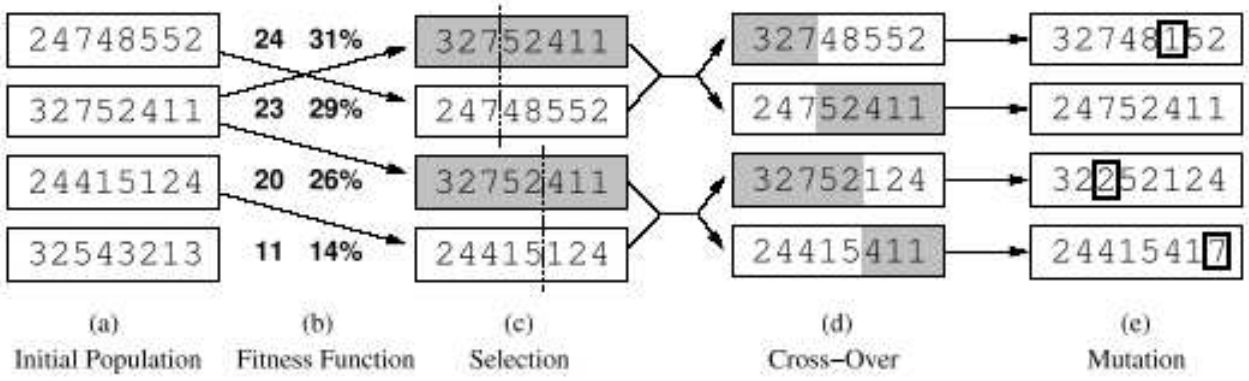

Figure 1: A visual representation of the fitness function, selection, cross-over, and mutation[1]

it's value is added to his/her score, evenly matched bids result in neither player receiving the flop.

This game was chosen both for its simplicity, but also for for the fact that a player is unlikely to dramatically change strategies based on the actions of they're opponent.

This was perceived as a possibly serious flaw in employing a genetic algorithm to adapt to a players habits; a fossilized record of a singular game (known as a replay) does not clue us in directly as to whether actions taken were derived internally by the player or were reactionary to actions taken by their opponent. While it is possible that, given enough replay data on a player, all likely reactions to each action an opponent makes may be taken into account, the sheer size would not only be difficult to gather, but also given the nature of humans to modify and improve strategies over time likely dilute the whole data set.

6.1 Species Definition

Any state in the game can be represented as the combination of:

- The cards remaining in your hand
- The cards remaining in your opponents hand
- The cards remaining in the flop
- The card to be bid on
- The score of the game

The first thing to be done is to strip out tangentially related information. While technically the only required fields are the remaining cards in your hand and the card to be bid on, including only this information results in a very simple species that is only capable of mapping a flop to the card to be played, resulting in a very predictable policy. Therefore the cards remaining in the flop and the opponents cards are also included in the species to give it

greater context. Score information however is purely tangential because regardless of the score we will still have the same goals.

So we end up with a mapping of every legal combination of cards I hold, cards my opponent hold, and what is left in the stack of flop cards to be bid on.

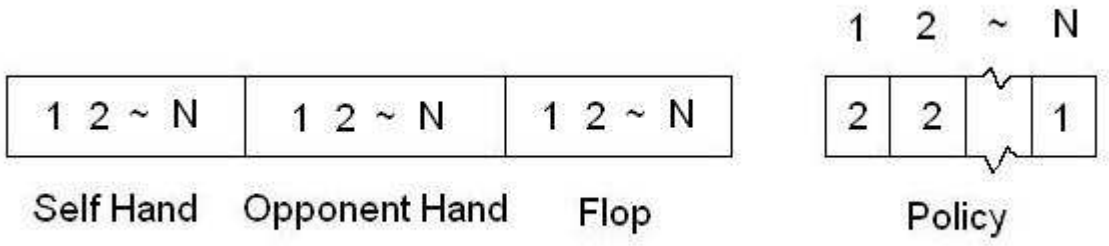

Figure 2: This diagram represents the anatomy of a single gene within a species. Information on the left represents the state information that this gene applies to and is used as a searchable index. The policy field on the right represents the actual genetic information. The position in the field indicate the flop card, and the number in the field corresponds to the card to be played

It is important to note that only the flop card to be bid on contains meaningful data, the rest of the fields merely serve as a unique identifier that we can search against. Indeed it is possible to completely remove this information and calculate algorithmically the index of the desired entry. While doing so would dramatically speed up the execution of the fitness function, removing a factor of N time, doing so unfortunately fell outside of my time constraints.

6.2 The Fitness Function

At time of writing, the fitness function employed is to play a species against a simple static policy that always matches the flops value on the bid. Here I define a static policy to be one that is a straight forward mapping of flop card to card to be played. The score for the whole game is represented as

$$Score = YourScore - OpponentScore \tag{1}$$

As the static policy will always match the flop, the highest card is unattainable, but by throwing off on it, it is possible to win every other card in the flop by playing one higher than its value. Using this strategy, the optimal score achievable is:

$$N - \sum_{i=1}^{N-1} i^1 \tag{2}$$

[1]It is notable that for deck sizes under 4, this static policy is unbeatable.

Under these circumstances, the genetic algorithm will evolve around being effective when playing against a foe whom employs a strategy of always matching the flop. However training against such a novice strategy will most likely produce a fairly novice policy. Therefore the next logical step is to use a more complex foe to use in our fitness function. For this we can use any given species' generation peers to provide a more rounded challenge. Unfortunately this increases the time complexity of the fitness function significantly, multiplying it by the size of the generation, plus an additional search through the state tree for the opponent.

6.3 Time Complexity

The algorithm presented[2] has a time complexity as follows, where the first polynomial represents the fitness function, and the second the cross-over function. N represents deck size.

$$\bigcirc \left(N!\right)^3 \tag{3}$$

However it is particularly worthwhile to note that since the fitness function and cross-over function do not depend on anything else, they belong to a class of problems that are known as 'embarrassingly parallel'. Which is to say that it can be paralleled very easily and to a high degree.

6.4 Results

Test runs on the ACLs using a generation size of 50, run for 100 generations with no upper score limit. Tests using a deck size of 4 completed in approximately 20 minutes, incrementing the deck size to 5 increased this time to 2 hours and 15 minutes, tests using a deck size of 6 yield a wall time of 15 and a half hours. However after enabling Optimization, Wall time decreased by a factor of 10. After looking over the test results I noticed a few interesting aspects that applied across test sizes:

- Algorithm hits the maximum very quickly, many times within the initial generation. This suggests to me that either I'm using a gratuitously large generation size, or I am testing on too small a test size.
- Algorithm is very 'bouncy'. Meaning that although it achieves the maximum score, it often times regresses out of the maximum, sometimes significantly. Since mutation is disabled in these builds, it must instead be the result of an improperly weighted mating ratio that does not give high scoring species enough of an edge. In any-case I believe it would be prudent to add in future builds the ability to save the all time highest scoring species in any generation, with priority given to the species occurring in the highest generation number.

[2]This refers to the static policy fitness function variation

6.5 Planned Additions

Admittedly, my implementation is somewhat simplistic in its current state. However there are a number of features that I would like to add in the future given the time. These features are sorted according to priority.

- Command-line Parameters for Evolution Settings - Adding command-line parameters for simulated annealing rate, percentage to mutate, and number of cross-over crosses will allow me to greatly enhance my ability to tweak and optimize the performance of my algorithm.
- Modularized Fitness Function - A swappable fitness function would allow me to easily redefine the meaning of the evolution easily. Possible Variations include playing species against their peers, or playing against a predefined set of static policies (player replay data).
- Algorithmic Access to Policy - This will allow me to dramatically reduce the time complexity from the fitness function to the cross-over function. The following would be the new time complexity, where N is the deck size.

$$\bigcirc(N) \tag{4}$$

- Implement Variable Threading - As genetic algorithms are embarrassingly parallel, this algorithm would greatly benefit from the added support for threading.
- Save/Load Functionality - The ability to save a species at the end of a simulation, or to load a species either to be used as the fitness function, or to be played against interactively.
- Interactive Play - A mode that would allow the player to play against a species, with the option of saving your game to a composite file containing all other previous games for the given profile. This could then be used as the fitness function in future runs.
- Web Integration - Allow a player to play against the AI more easily and more often, building a more complete profile for the player, plus allowing me to show off my senior project to friends and family.

In addition to these features, there are few more experimental additions I would like to consider.

- Family Grouping - Family Grouping is a feature in the Neural Networking algorithm NEAT[3], that allows it to categorize species into distinct families which affects the cross-over stage by creating a strong affinity for mating with other species of it's own family.

[3]Neural Networking algorithms are a subdivision of genetic algorithms that simulate specifically how the brain works. Policies are represented as a series of nodes in a directed graph that are meant to mimic neurons in the brain

This is done by keeping historical track of each specie's development to quickly categorize them, rather than employing a much slower topological analysis.[15] Unfortunately this method does not translate well into my particular problem space.

- Genetic Calibration - An interesting feature, this would create a genetic algorithm that would attempt to optimize the settings for my actual genetic algorithm. While such an algorithm would be significantly simpler[4] there exist two problems:

    1. Optimizing this wrapper genetic algorithm could prove to be just as difficult as optimizing the underlying genetic algorithm, somewhat defeating the whole purpose of the function.
    2. More importantly, the time complexity associated with such an algorithm would be astronomical.

7 Conclusion

These tests show that genetic algorithms are effective at generating counter strategies in simple strategic games. Unfortunately the time complexity attached makes genetic algorithms currently unsuitable for larger more complex games. However as machines become more and more parallel we may find that the embarrassingly parallel nature of genetic algorithms will bring them back into feasible territory for complex games.

References

[1]

[2] Robert St. Amant and R. Michael Young, *Links: artificial intelligence and interactive entertainment*, Intelligence **12** (2001), no. 2, 17–19.

[3] Renato Reder Cazangi, Fernando J. Von Zuben, and Maurício F. Figueiredo, *Autonomous navigation system applied to collective robotics with ant-inspired communication*, (2005), 121–128.

[4] Vincent Conitzer and Tuomas Sandholm, *Communication complexity as a lower bound for learning in games*, (2004), 24.

[5] Yoav Freund and Robert E. Schapire, *Experiments with a new boosting algorithm*, (1996), 148–156.

[6] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore, *Reinforcement learning: A survey*, Journal of Artificial Intelligence Research **4** (1996), 237–285.

[4]The species template would merely be each of the performance relevant settings represented in a field, and the fitness function would be implemented as a run of the genetic algorithm, reporting back both the maximum score attained, and optionally the generation at which the algorithm plateaus at

[7] Dimitrios Kalles and Panagiotis Kanellopoulos, *On verifying game designs and playing strategies using reinforcement learning*, (2001), 6–11.

[8] John E. Laird, *Using a computer game to develop advanced AI*, Computer **34** (2001), no. 7, 70–75.

[9] Nicolas Lassabe, Stéphane Sanchez, Hervée Luga, and Yves Duthen, *Genetically programmed strategies for chess endgame*, (2006), 831–838.

[10] Daniel Livingstone, *Turing's test and believable ai in games*, Comput. Entertain. **4** (2006), no. 1, 6.

[11] J. J McDowell, Paul L. Soto, Jesse Dallery, and Saule Kulubekova, *A computational theory of adaptive behavior based on an evolutionary reinforcement mechanism*, (2006), 175–182.

[12] Risto Miikkulainen, *Evolving neural networks*, (2007), 3415–3434.

[13] Asadollah Norouzi, S. Mohammad Mohammadzadeh Ziabary, Morteza Mousakhani, and S. Mohammad Reza Shoaei, *Semi-human instinctive artificial intelligence (shi-ai)*, (2006), 153–164.

[14] Asim Roy, *Artificial neural networks: a science in trouble*, SIGKDD Explor. Newsl. **1** (2000), no. 2, 33–38.

[15] Peter Norvig Stuart Russell, *Artificial intelligence: A modern approach*, Prentice Hall, 1995.

[16] Matthew E. Taylor, Shimon Whiteson, and Peter Stone, *Comparing evolutionary and temporal difference methods in a reinforcement learning domain*, (2006), 1321–1328.

[17] Geoffrey G. Towell and Jude W. Shavlik, *Knowledge-based artificial neural networks*, Artificial Intelligence **70** (1994), no. 1-2, 119–165.

[18] Fang Wang and Eric Mckenzie, *A multi-agent based evolutionary artificial neural network for general navigation in unknown environments*, (1999), 154–159.

[19] Shimon Whiteson and Peter Stone, *Evolutionary function approximation for reinforcement learning*, J. Mach. Learn. Res. **7** (2006), 877–917.

[20] Du Zhang and J.J.P. Tsai, *Machine learning and software engineering*, Tools with Artificial Intelligence, 2002. (ICTAI 2002). Proceedings. 14th IEEE International Conference on, 2002, pp. 22–29.