FreeCiv Learner: A Machine Learning Project Utilizing Genetic Algorithms

Felix Arnold, Bryan Horvat, Albert Sacks
Department of Computer Science
Georgia Institute of Technology
Atlanta, GA 30318
farnold3@gatech.edu bhorvat3@gatech.edu gtg913h@gatech.edu

Abstract

Genetic Algorithm techniques were applied to the complex strategic computer game FreeCiv in order to find competitive parameter setting for the AI player. The particular aspect of the game we looked at was the city placement. Several experiments with different tunings of the Genetic Algorithm were performed. It was found that the performance of the AI player could be improved with generation using this technique. The outcome was greatly influence by the size of the population. The Player with the newly found parameters even outperformed the AI with the regular parameter settings.

1 Introduction

Computer games are an extremely large part of pop culture, and one of the most prominent, and hardest, areas of study in the gaming world deal with artificially intelligent game agents. One particular game, called Civilization, deals with an agent, or player, controlling the actions of several sub-agents, or units, with the goal of creating networks of cities and armies in a small world, in order to defeat the civilizations of enemy agents. The complexity of the game requires artificially intelligent agents to be extremely well developed and complex, themselves. The task of creating an artificial agent in a game such as this is extremely difficult, and it would be advantageous to allow the agent, itself, to learn intelligent strategies, rather than require a human to learn and program the entirety of the agent's capabilities him/herself. The goal of this paper is to use machine learning techniques to allow an artificial agent to learn an optimal solution for one specific aspect of the game, namely city placement. Several other studies have been conducted using similar techniques and the same game such as [1],[2] and [3].

City placement, although it makes up only a very small fraction of the game, is extremely important, and a good city placement algorithm can greatly influence the outcome of the game. Cities are the central component of a civilization, and are responsible for producing armies, food, technology, and several other important resources, all of which are affected by the location of the city. A well-placed city will allow for more production of resources and trade, allow for quicker production of armies, and allow for better defense from enemy armies. Thus, it would be of benefit to learn an optimal city-placement solution.

FreeCiv is an extremely complicated game in which multiple factors have various effects on many other factors. Therefore, in order to devise a quality machine learner, the game must be simplified by removing as much as possible; specifically, everything but what is to be learned. For this reason, the strategies to be learned must be simplified as well. For this project, the game must be simplified around the specific task of learning optimal city-placement. Since city-placement is performed at the beginning of the game, only the first few moves of the game will be played. This, by itself, will cut out almost all other aspects of the game which could affect the resulting fitness function.

2 City Placement

The FreeCiv world is divided into a grid of squares, or tiles. Each tile is an indivisible unit of area on the map. The game begins with each civilization having a small number of units, or "people." One of these types of people is a settler. The settlers' job is to explore the map and build cities. The job of the city placement program is to control the movement of each settler and to find a tile on which the settler will begin building a city. Once built, a city can expand beyond the tile on which it started. Therefore, if a city placement algorithm is to be successful, it must take into account not a single tile, but a grid of tiles.

The FreeCiv program has a well working set of algorithms to control the AI players in a game. Included with these algorithms is a suitable city-placement algorithm. Therefore, since the purpose of this project is to use machine learning techniques to learn optimal parameters, and not to devise an artificial intelligence mechanism itself, the city placement algorithm that the FreeCiv software provides will be used, and certain parameter values within that algorithm will be modified and their optimal values will be learned via a genetic algorithm. Each tile on the map has several values, such as food supply, defensive characteristics, etc. For each tile in question, and the tiles in the surrounding vicinity, the city-placement algorithm extracts this information, and performs some basic calculations to predict other attributes, such as potential for city growth, potential for trade, etc. These values are all combined in such a way as to give the tile in question a potential value for building a city. If this value is high enough, and is the best in the area, then a city is built. If not, then searching for potential city locations continues until a city is finally built.

3 Optimization Parameters

Thirteen parameters are to be optimized. These are equivalent to the attributes of the input to the function whose optimal value we are trying to find. The parameters to be optimized are as follows:

Perfection:

The "perfection" attribute is a parameter that determines the amount of perfection required by the optimal solution. If a higher perfection is required, then the algorithm will take longer to search for a more optimal city location. The total result found for a certain location is multiplied by the perfection parameter. For a certain land value, a lower parameter value requires a higher optimum result, and so a lower parameter value means that more perfection is required.

Result Is Enough:

The "Result Is Enough" attribute is a threshold value on the minimum acceptable predicted value of the solution. After the value of a certain tile is determined, it is tested against this threshold parameter. If it exceeds the threshold, then a city is placed on the tile. If not, then no city is placed, and the search continues. Since the fitness function of the city takes into account the total amount of resources generated throughout the entirety of the game, it is advantageous to build a city as early as possible. Also, since it is obvious that the more cities a civilization contains, the better are its chances of survival, so the fitness function sums the qualities of all of the cities owned by the player. Therefore, a good city placement strategy must also take into account speed of city placement, and so learning this parameter is akin to finding a good tradeoff between the desire to finding a good tile and the desire to not waste valuable turns searching for a location. This has much of the same effect on the city-placement algorithm as does the perfection parameter, and so, the optimum values of these two parameters are highly dependent on each other. Therefore, a more efficient algorithm may be to learn a ratio of the two.

How many turns to look after finding a good enough result:

If a good tile for placing a city is found, it is most likely the case that there are other valuable tiles nearby. Therefore it may be worthwhile to continue searching for a few turns to be sure that the optimum tile has been found. This parameter sets the number of turns to continue searching after finding a good tile. If this parameter is set too low, then a nearby tile, which is better than the currently best-found tile, may be missed. And if the parameter is set too high, then much time will be wasted, and the algorithm will take too many turns to find a good location. Therefore, it is advantageous to learn an optimal value for this parameter.

Growth Priority:

The growth priority parameter is used to prevent possible overcrowding in a city.

Growth Potential De-emphasis:

This parameter emphasizes how important it is that a city have potential to grow. The smaller this value is, the more important it is that the city-placement algorithm find a tile which allows for a large city, and, thus, the higher a potential tile's value will be if it has space for large cities. By learning this parameter, the learning algorithm, in a way, learns the short-term importance of a city's size. It is expected, however, that there are no short-term advantages or disadvantages of a city having the potential to grow large over time. Since most of the effects of a city's size are most likely to be long-term effects, it is expected that this is attribute's value contributes little to the overall fitness function.

Defense Emphasis:

Each tile is given a certain defense bonus, proportional to the ease of defending a city against attack, at that location. The defense emphasis parameter indicates the importance of building a city in a location that is easy to defend. The higher the defense emphasis, the more the tile's predicted value will be affected by its defense bonus. Since the number of turns per game is no more than 55, it is expected that, although a city may be attacked, it won't be attacked very many times, nor will the strength of a possible attack be very large. Therefore, it is expected that this parameter will have an effect on the ultimate fitness of a city…although not a very large effect.

Naval Emphasis:

The naval emphasis parameter specifies the importance of locating a city adjacent to an ocean. Placing a city near an ocean has many benefits, such as increased irrigation, and the inability of land troops to attack a city on all sides. However, it may not be beneficial to place too much of an importance on finding a good location near an ocean, as it may take too long to find such a location, and valuable production may be lost if a city is built too late. Therefore, the genetic algorithm attempts to learn the optimum importance to place on finding a location near an ocean.

Building cost reduction importance:

A location may be very good location. However, if a boat is required to be built in order to place the city, then its importance must go down. This parameter determines the cost to place on the value of a tile per the amount of cost required to build the city.

Food importance:

This parameter emphasizes the amount of importance to place on the amount of food production that a tile will afford a city. The predicted amount of food a city will be able to produce, at a certain tile, is determined, multiplied by this parameter, and then added to the total value of the tile.

Science importance:

This parameter emphasizes the amount of importance to place on the amount of scientific development and trade that a tile will afford a city. The predicted amount of science a city will be able to develop, as well as the amount of trade a city will be able to perform, at a certain tile, is determined, multiplied by this parameter, and then added to the total value of the tile. The amount of corruption within a city that may result is also linked to its science and trade. Therefore, the city-placement program also determines the amount of corruption possible at a certain tile, multiplies it with this parameter, and subtracts it from the total value of the tile.

Shield importance:

This parameter emphasizes the amount of importance to place on the amount of shields that a tile will afford a city. The predicted amount of shields a city will have, at a certain tile, is determined, multiplied by this parameter, and then added to the total value of the tile. The amount of waste that a city may produce is also linked to its shields. Therefore, the city-placement program also determines the amount of possible waste a city may produce at a certain tile, multiplies it with this parameter, and subtracts it from the total value of the tile

Starvation threshold:

The starvation threshold is the minimum amount of food a city may produce without its residents being expected to starve. If a city's expected food production does not exceed this threshold, then it is expected that a city will not be able to survive at this tile, the value of the tile is set to 0, and no city is built here. This parameter is important to learn, as it will not allow a city to be placed at an otherwise valuable tile if a threshold amount of food cannot be produced at the tile. Therefore, if this parameter is set too low, then some cities may starve, and if this parameter is set too high, some good tile locations may be overlooked.

Minimum shield parameter:

This parameter is much like the previous parameter, in that it sets a threshold under which no city will be built. The shield threshold is the minimum amount of shields a tile will afford without its residents being too vulnerable to attack. If a city's expected shields do not exceed this threshold, then it is expected that a city will not be able to survive at this tile, the value of the tile is set to 0, and no city is built here. This parameter is important to learn, as it will not allow a city to be placed at an otherwise valuable tile if a threshold amount of shields cannot be acquired at the tile. Therefore, if this parameter is set too low, then some cities may not survive an initial attack, and if this parameter is set too high, some good tile locations may be overlooked.

4 Fitness Function

At the end of each game, which lasts 55 turns, the fitness function is used to determine the fitness value of the specific parameter settings. The fitness function is a linear combination of several resources generated by all of the cities owned by the player. The specific algorithm used to determine the fitness of a city is an adaptation of an algorithm in the FreeCiv AI software, which determines the quality of a particular city. The main difference between the two is that the FreeCiv algorithm uses some of the specific parameters that are being learned, to determine the quality of a city. Using the learning parameters in the fitness function would defeat the purpose of learning the parameters, and therefore, they were removed from the algorithm. The fitness function iterates through all cities owned by the player, and sums up the individual fitness of each city as follows. Each city produces a certain amount of food, trade (in the forms of science and luxury), and taxes each turn. Each city also has a certain amount of shields. These resources are all tallied up and added to the fitness of the city. However, the city's shields produce waste and pollution, and the city's trade causes corruption. The total waste and pollution created by the city, and the corruption in the city, are tallied up and subtracted from the total fitness of the city. This produces the final fitness value of the city. It can be argued that certain elements are more important to a city's survival than are others. However, as these relative levels of importance are subject to opinion, we have given equal weight to each element.

5 Genetic Algorithm

A genetic algorithm is used to optimize the parameters of the city placement algorithm. At the beginning of the program, a set of "parent" instances are generated, whose attributes are all randomly set. Since the original finesses of the parents are unknown, they are all set to a constant value. Different numbers of instances were generated throughout the experiments, but for clarity, assume for the rest of this section that the number is 50. Then 96 children were generated from the 50 parents. To ensure that the fittest parents don't get erased, the four fittest parents were copied to the children. This technique is called elitism and it makes the algorithm to converge faster [4] [5]. After creating the children, FreeCiv was run, setting the parameters in the game to each child in succession and saving each child's fitness value. Each game was run for 55 turns. After each generation of games, the 50 fittest children were found and copied back into the parents, and the procedure was repeated.

Generating the children from the parents worked as follows. Two parents were randomly selected, based on a probability proportional to the cube of their fitness values [6]. From these two parents, two children were created using a crossover function. The crossover point is between the eighth and the ninth parameters. This seemed to be a good crossover point, as the ninth through thirteenth parameters seem to be closely related to each other, and the first through eighth parameters seem to be more sporadic. The point of crossover, however, is highly subjective, and the project would have benefited greatly if more effort had been put into researching this. After crossover, random mutation was performed on the children, based on a small probability. Random mutation was performed by setting one random attribute to a new, randomly selected value. These steps were performed 48 times, producing 96 children.

6 Results

The results from the first few runs were very strange, indicating our algorithm actually got worse over time. This was the result of a bug with the creation of children, and was fixed. With that fixed, we ran it through some examples of games based on saved game files, to minimize randomness (there's still a little bit). We tried varying the number of children per generation and the number of generations to see which affected improvement more. Note that the run time is proportional to the product of the number of children and the number of generations. What we discovered was that doubling (approximately) the number of children was far more effective than doubling the number of generations for the same cost.

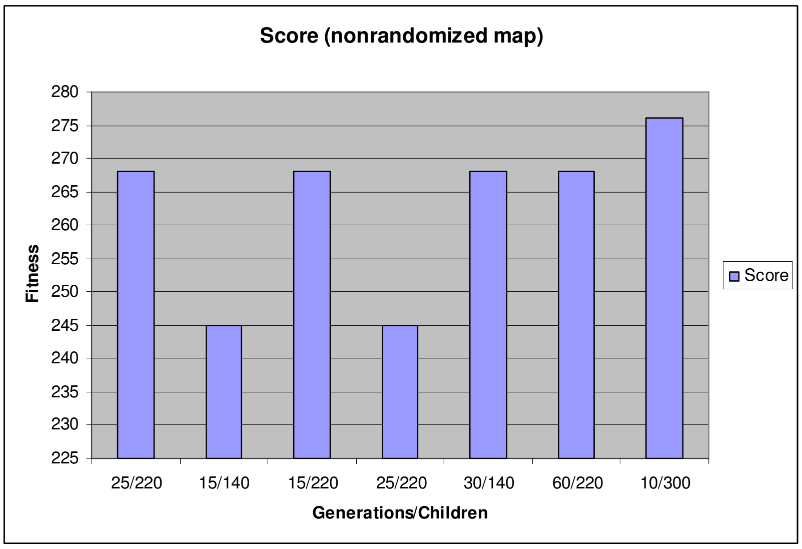

Figure 1: Comparison of score vs. total generations and children per generation, listed by trial number

As shown on the chart, many of the runs ended with similar results. This is due to the fact that, due to the use of a relatively small number of saved games during our initial work, it is quite likely that one or more children will hit upon the argmax of the function easily. More interesting is the fact that children are more important than generations. The most likely explanation for this is that the more initial variance in the algorithms, the better, since mutation is very unlikely. The highest value of the first generation for the >200 children set was consistently higher than the highest value of the first generation in the 140 children set. Indeed the first generation of the >200 children set outperformed the final generation of the 140 children set. The use of saved games greatly decreased variance, but probably caused tremendous overfitting. The fact that we had to limit the number of turns played to 55 in order to make the simulation take a reasonable amount of time also influenced the maximum fitness value as well.

After we confirmed our algorithm worked with a saved and nonrandomized game, we modified it to operate over randomized games. Since the map was now fully random, the amount of resources extracted is very variable and thus the fitness values are very variable. We decided to utilize the average performance of five runs per child per generation in order to smooth over some of the randomness. Once again performance would improve quite a bit from the first generation to the last (one trial with only 10 generations had the maximum fitness jump from 224 to 276 and average for all children jumped from 131 to 193). With randomized games, it becomes clear that generations are far more important than numbers of children now. Even with only 30 children, the gains seem strong.

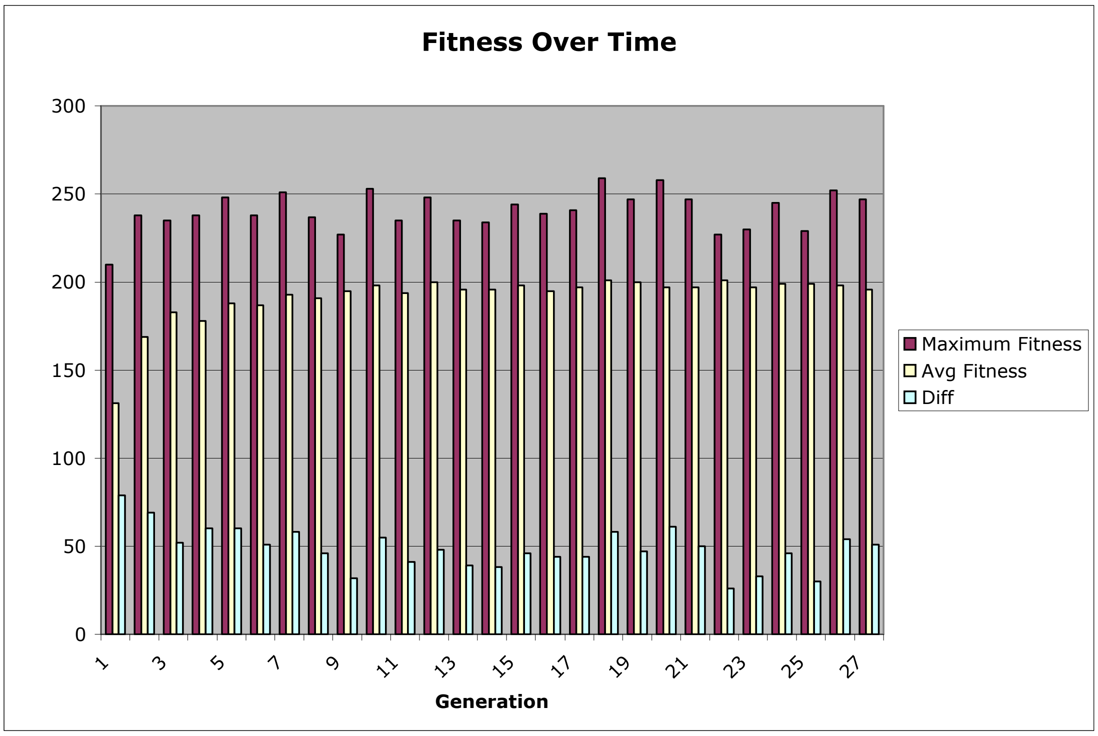

Figure 3: Scores for the algorithm running on a fully randomized game, shown by generation

The randomness means that the gains are not monotonic, but there is a noticeable upward trend for the first few generations. After approximately 7 generations, the gains lessen and the maximum fitness oscillates around a level near 240. The average fitness on the other hand stays locked near 200 with much less variation. As a point of comparison, the default AI maxes out at 216 fitness rating and averages 196. This suggests our algorithm is better, at least by a bit.

7 Conclusion

The test results suggest that FreeCiv is sufficiently random and that learning a truly optimal policy is difficult, but that it is relatively easy to learn a decent policy. The results also suggest that a good policy for one map will also perform decently on another map, suggesting that optimal values for the parameters in the algorithm are global to the game, and work for all types of terrain.

Although much has been done in the way of comparing and finding the optimal parameters of the genetic algorithms (number of children, generations, etc.) and in learning optimal values for decision-making parameters in the city placement algorithm (food priority, defense priority, etc.), not much has been done in the way of running this algorithm in a full game to see if it truly is a desirable algorithm. Future work would include running this algorithm in several full games. This would require considerably more time and CPU capacity than we have at the moment.

Acknowledgement

We would like to thank Dr. Goel for his assistance and insights into this field. The material on this subject that he provided was particularly helpful.

References

[1]Hierarchical Judgement Composition: Revisiting the Structural Credit Assignment Problem, J. Jones, A. Goel / Proceedings of the AAAI Workshop on Challenges in Game AI, 2004

[2] Using Model-Based Reflection to Guide Reinforcement Learning, P. Ulam, A. Goel, J. Jones, W. Murdock – IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games. Edinburgh, 2005

[3] A Strategic Game Playing Agent for FreeCiv, Masters Thesis Project by Philip A. Houk

[4] Elitism-based compact genetic algorithms, Chang Wook Ahn; Ramakrishna, R.S.; Evolutionary Computation, IEEE Transactions on, Volume 7, Issue 4, Aug. 2003 Page(s):367 – 385

[5] Convergence models of genetic algorithm selection schemes, D Thierens, DE Goldberg - Lecture Notes in Computer Science, 1994

[6] A Comparative Analysis of Selection Schemes Used in Genetic Algorithms, DE Goldberg, K Deb - Urbana