International Conference on Computational Science, ICCS 2013

Co-evolution of Antagonistic Intelligent Agents using Genetic Algorithms

Jhonatan da Rosa\textsuperscript{a}, Murillo T. de Souza\textsuperscript{a}, Luciana de O. Rech\textsuperscript{a}, Leandro Q. Magnabosco\textsuperscript{a}, Lau Cheuk Lung\textsuperscript{a}

\textsuperscript{a}Graduate School in Computer Science - Federal University of Santa Catarina, Florianópolis 476 – 88.040-900, Brazil

Abstract

The aim of this paper is to attest the improvement on strategies of intelligent adaptive agents created using genetic algorithms in electronic games. We present an experiment on the use of genetic algorithms to create intelligent adaptive agents which iterates upon the opponent strategy. A predatory food chain was simulated, containing carnivores, herbivores and plants. This simulation uses the approach of a co-evolved asymmetric antagonistic agent population. Because they use each other as part of their environment, they are also able to learn from exhibited behavior after their evolution. Agents are expected to show a satisfactory evolution, analogous to the learning process of an intelligent being.

Keywords: genetic algorithm; co-evolution; games;

1. Introduction

The human brain has the ability to selectively focus its attention. When something catches the human attention and imagination for a long period, the human brain enters into a differentiated state called flow channel [1]. Game designers try to keep players inside this flow channel when creating a new game. Therefore, games must be designed to always provide a consistent challenge within the reach of the ability of the player. Recreating intelligent behavior is always challenging, however because of the lack of ways to evaluate the results of newer algorithms.

This is the reason why electronic games play an important role as an optimal platform to solve the issue, delivering a good environment for hypothesis testing and the evaluation of different implementations. Because electronic games are naturally fast paced, results are quickly received. The qualities ensure a much needed ease on complexity when performing quality assertions for algorithm adopted strategies.

Many researchers independently studied evolutionary systems under the premise that they could be used as an optimization tool for engineering problems [2][3]. Jon Holland defined the concept of genetic algorithms and started the development of the area with the support of students. In contrast with other techniques of evolutionary programming [4], the objective of Holland was not to obtain algorithms to solve specific problems, but to study

\textit{E-mail address:} jhow@inf.ufsc.br.
\textit{E-mail address:} titon@inf.ufsc.br.
\textit{E-mail address:} luciana.rech@inf.ufsc.br.
\textit{E-mail address:} leandroqm@gmail.com.
\textit{E-mail address:} lau.lung@inf.ufsc.br.
the adaptation phenomenon and to develop ways to assimilate the knowledge of natural adaptation mechanisms and its possible use on new and improved computational systems.

This paper has the same focus Holland had by seeking, not an optimization to a specific problem, but to observe the adaptation of agents evolving from the influence of a genetic algorithm. The purpose of this paper is to create a scenario based on [5], that simulates a simple ecosystem where two antagonistic and asymmetric population of intelligent agents are left to evolve using a genetic algorithm.

2. Genetic Algorithms

Genetic Algorithms are search algorithms based on genetics and the natural selection mechanism. This kind of algorithms solve problems by simulating among individuals the theory of survival of the fittest over consecutive generation, forming an heuristic search algorithm [3]. To use such algorithms, it is necessary to model a data structure that represents a possible solution for a problem, which is called simply Gene. A gene suffers the basic operations of genetic algorithms: reproduction, crossover and mutation [3].

2.1. Genetic Algorithm Scheme

A genetic algorithm engine is responsible for performing basic operations, such as reproduction, crossover and mutation, in populations of individuals. At the reproduction stage, individuals are selected to contribute to one or more children for the next generation.

A fitness function is a type of function used to find the best solution from all feasible solutions in a problem. The aforementioned selection is executed by applying this function. It does summarize if a given design solution is close to achieving its aim. When applied to the chromosome of an individual, a measure of profit, utility or quality is returned. Which measurement is returned depends on what is desired to maximize.

After the definition of which individuals contribute to the next generation, crossover operations take place. At first, selected individuals are matched and an exchange of genetic information between them is enforced. The effect of this operations carries depending on the chromosome model of the selected individuals.

Finally, the individual undergoes the mutation, resulting in new and improved chromosome. In this operation, a small part of the information of the individuals is randomly altered. According to [3], reproduction and crossover are the main operations of genetic algorithms.

2.2. Binary decision Diagrams (BDD)

A Binary Decision Diagram (BDD) is a data structure used to represent a Boolean function. Since each node has two output branches and since one and only one of these is activated for a given input, it follows that for any input exactly half of the branches in a diagram are activated. Moreover, since each node has one and only one active output branch, it follows that from every node there is one and only one active path to an output value of 0 or 1 (one)[6].

![Binary Decision Diagrams](image)

Fig. 1.
A \( n \)-BDD differs from a BDD in the quantity of possible output values. In a BDD there can be only 2 output values, and in a \( n \)-BDD there can be \( n \) outputs. The \( n \)-BDD format was chosen to model the data structure of this paper.

2.3. Chromosomal Representation

All living organisms consist of cells and each cell contains one or more chromosomes. Those are basically chains of Deoxyribonucleic Acid (DNA), which dictates aspects of the organism it is part of. Every chromosome can be conceptually divided into two DNA functional blocks that encode a particular protein. One can say that each gene encodes a characteristic, such as eyes or hair colour. Different possibilities for a single characteristic are called alleles. Each gene is located in a locus, which is a specific position of a chromosome [2].

When working with genetic algorithms, the term chromosome refers typically to a possible solution to a given problem. Chromosomes are usually defined by bits strings of equal sizes [3], but there are some examples of different modelling, as the ones used by [5]. On the classical model, a gene is a block that contains one or more bits inside the chain that encode a particular element towards the solution of a problem. In this kind of model, a bit is analogous to a genetic allele. While the implementation of the engine of a genetic algorithm is reasonably simple and well defined, the chromosome modelling and the evaluation function require a lot more attention, as those are the key factors for the success of the solutions found by the genetic algorithm.

A common obstacle when attempting to encode a problem as a chromosome is that you rarely possess the knowledge of which characteristics of the problem are relevant enough to be considered for the chromosome beforehand and on which loci would be the best fit for them. [2].

Besides the traditional chain of bits, [2] exemplifies encoding use cases using chains where the alphabets are bigger than binary alphabets and tree form encoding, which were used by [7] to model a chromosome representing a complex math function.

2.4. Definition of Schemas

An interesting concept that emerges from the chromosomal representation are the schemas. Defined originally by John Holland as a template that describes a subset of strings with similarities at certain string positions [3]. For a traditional size 4 chromosome, a possible schema is (0*110). That schema is defined by all possible combinations, replacing every ”*” by the values of the alphabet (1 or 0), therefore \{(00110), (01110)\}.

From the notion of schemas, Holland formalized the informal conception of the construction blocks [2]. This notion determines that, while the genetic algorithm explicitly evaluates the population strings aptitude, it is implicitly evaluating the average aptitude to a much greater number of schemas.

2.5. Selection Operation

The selection operation [2], also known as reproduction operation [3], defines which individuals will be selected to iterate to a new generation. The goal here is to emphasize the survival of the fittest individuals characteristics in the future generations, in the hope that their offspring will possess an even higher fitness.

There are several different methods to implement the selection operation. Some of them are:

- **Fitness Proportionate Selection**: Used by John Holland in the original creation of genetic algorithms [2]. One implementation of this method is the Roulette Wheel Selection, where a slice of a wheel proportional to the fitness is allocated for the subjects and the wheel is rotated \( n \) times to select all the \( n \) necessary individuals.

- **Stochastic Universal Sampling**: Proposed in 1987 by James Baker [2], this method seeks to minimize the negative effect observed when in the roulette method. Instead of spinning the wheel \( n \) times to find \( n \) individuals, fixed sections radially spaced are marked on the wheel. This way, the roulette only needs to spin once and all the individuals needed are selected. Both methods present one big problem: Because of the high variation between the fittest and less fit individuals in the early generations, the fittest ones and their descendants tend to dominate the offspring, preventing the genetic algorithm to explore other possibilities[2]. This issue is known as premature convergence.
• **Sigma Scaling**: This method seeks to keep the selective pressure, which is the degree that highly fitted individuals are selected for reproduction, constant throughout the genetic algorithm execution. The number of times an individual is expected to be selected is a direct function between the aptitude of the individual, the reverse population average and the standard deviation, which is usually represented by a sigma ($\sigma$). This ensures that the beginning, when the standard deviation is higher, less fit individuals will have an opportunity to remain for long enough in the population until the standard deviation is lower, when the selective pressure is allowed to grow [2].

• **Elitism**: Using elitism means that a fixed number of individuals among the fittest will always remain in the population. Some researchers state that the use of elitism improves significantly search performance [2].

• **Ranking Selection**: In this method, the concept of absolute fitness is discarded and relative fitness is used instead. Individuals are ordered in accordance to their fitness from 1 to $n$, where $n$ is the population size. After that, a number is given to each individual, according the previous ordering, representing the number of times that particular individual is expected to be selected, evolving linearly until the last candidate. This method can be quite slow to find individuals with high fitness [2].

• **Torney Selection**: At least two iterations are needed in order to use any of the aforementioned methods. One to calculate the fitness average of each individual and the other to calculate the number of selections that are expected to happen. Besides, in the ranking selection, individual ordering is needed. Those operations can prove to be expensive and diminish the algorithm efficiency. Even tough this method has a similar selective pressure to the ranking selection, this method has the advantage of being more efficient and more prone to parallelism [2]. The method consists of choosing two random individuals and a random number $k$ which is contained between 0 and 1. If $k$ is bigger than an arbitrary number, such as 0.75 for example, the fittest individual is chosen. If $k$ is not bigger than the arbitrary number, the least fit individual is chosen. Both can be picked once again in the future.

• **Steady-state Selection**: In this selection paradigm, only a part of the population iterates each generation. This method is effective when there is a need for incremental learning and when the population must act as a group [2].

3. **Flow Channel**

The focus of the human brain, at any given point, is determined by a combination of unconscious desires and conscious will. When creating games, the main objective should be to create an experience interesting enough to hold the focus of the player for the longest period of time and as intense as possible, allowing the mental state of the subject to enter in what is called the **flow channel**. This is also defined as a **feeling of complete focus on an activity that provides a high level of entertainment and satisfaction** [1].

There are a few key prerogatives in order to achieve the goal of activating the flow channel on a human being, such as:

- Clear Objectives: When the objectives are clear, the subject remains focused more easily
- Constant Challenge: Since they demand concentration
- Direct Feedback: Quick responses help on focus maintenance
- No Distractions: As break the focus and, without focus, there is no flow

The problem is to keep the challenge consistent with the skill of the player. The player might feel frustrated whenever he feels the challenge he is facing can not be beaten. His mind might start searching for easier rewards in other activities, breaking the flow. In other hand, if the challenge is too easy to beat, the player might feel bored and will probably seek more rewarding activities.

This is the reason why the game designers focus is on calibrating challenges difficulty, so the player is hooked for the longest time possible. Also, considering the ability of the player shall increase, the challenges have to be adjusted in accordance to the newly acquired skills of the player.

The challenge to keep the player in the flow channel are represented in the Figure 1(b).
4. Related Works

There are a few papers discussing different implementations of genetic algorithms to solve the classic Traveling Salesman problem, a few more to The Mastermind game. Those are using static environments, but are still interesting in a few ways. Then there are also a few that work on adaptive premises, such as simulations of an ecosystem, some that show the emergence of strategies for persecutions and others that analyses the decision of the algorithms on a flight environment.

It became evident that there is a huge difficulty to manage the increasing complexity of evolving populations. This concern is even worse when dealing with adaptability. Finding an acceptable solution in static environments is really difficult, even though only a single run of the algorithm is needed. Non-static environments are even harder, hence the opponent is constantly changing behavior. A good strategy may lose strength when the environment changes, as illustrated in [8]. This situation would require the search for solutions to be constant. This drawback has a high relevance on the application of genetic algorithms as opponents in games. In order to reach the goal of providing the player an opponent continuously adapted to their new strategies, it is mandatory that the algorithm is always running. This is similar to what can be observed on the implementation of the off-line cities manager on the game FreeCiv [9]. FreeCiv is a turn-based game, which means the algorithm does not need to be extremely responsive. There is no problem if it takes several seconds or a few minutes to define their actions. However, in case of games where response time is important, this is a big issue.

The work done in [10] provides an alternative to manage the complexity of the increasing population. According to their experiments, even running on a single machine using threads to simulate parallelism, the algorithm efficiency obtained the performance similar to the conventional implementation. There would probably be a big performance gain when using this technique on a computational grid.

In the experiments done in [7], strong parallels can be drawn with the objectives of this work. Both mentioned experiments implement evolution on changing environments by the use of two antagonistic populations that evolve simultaneously. Even though it is not the focus of these papers, the ideas implemented on [11] and [5], show great scenarios to observe the emergence of behaviors and strategies that can be considered credible from an intelligent agent. In [12] and [13], it is shown the ability to create agents for complex tasks, even when there is little knowledge about environment, suggesting that it is possible to automate most of the steps on the process of creating artificial opponents.

Unlike the aforementioned studies, the aim of this work is to analyze the evolution of agents in real time and in dynamic environments. Despite concerns about the level of complexity being too great, which would invalidating the proposal, the work of [10] suggests that it is possible to get around the problem of efficiency. As for effectiveness, [8] shows that co-evolution schemes are prone to create agents that alter their behavior according to the actions of other agents, corroborating with our proposition, that this paradigm allows to generate more interesting opponents in electronic games.

5. Performance Analysis

5.1. Scenario

The scenario for the implementation was inspired by [5]. A bi-dimensional grid, two antagonistic populations, herbivores that feed on plants and carnivores that feed on herbivores, are co-evolved by a genetic algorithm. The only purpose for the plant population is to serve as food for the herbivores. The possible actions for the agents are:

- Move: to move randomly in any direction.
- Approach: to approach of an individual of the same species.

For the Carnivores there is a special action called Hunt, which is to pursue an herbivorous. And for the Herbivores there is a special action called Runaway, which is to move in opposite direction of a carnivorous in the field of vision. Each individual has a field of vision of 3 squares to each direction and 100 energy, which is decremented by 2 when an action is taken by the individual. An individual can Feed when there is another individual of lower position in the food chain in the same position. When an individual feeds, its energy is
increased by 30. Once the energy of an individual reaches a value above 100, a new individual of the same species is created, with half of energy and the same gene. There are three sensors for an individual that are used as base for decisions:

- Carnivores Visible: indicates if there is at least one carnivorous visible in the field of vision.
- Herbivores Visible: indicates if there is at least one herbivorous visible in the field of vision.
- Plants Visible: indicates if there is at least one plant visible in the field of vision.

The system is turn-based and each individual can make only one action per turn. When there is no individual alive or the turn counter reaches a determined value, the system stops the simulation and processes the genetic algorithm operations to create a new generation for the next simulation.

5.2. Gene Model

The gene was modeled in a flattened tree, represented by a \( n \)-BDD. The Figure 2 presents a possible gene for one individual and a possible decision made by it. The red line represents the sequence it chose in accordance to information provided by the sensors.

![Gene Representation](image)

6. Graphical Interface

The Figure 3 shows the graphical interface developed to visualize the simulation. The field where agents can move is represented by the grid on the left. The herbivores are represented by the green circles, carnivores by the red circles and plants by the brown circles. Each cell represents a possible position that an individual can occupy. Still on Figure 3, an average of the evaluation function is represented by black lines while the maximum is represented by red lines for the last 15 generations of herbivores, respectively on the upper and lower graphs. The interface has controls to accelerate the simulation, controlling the maximum steps to be executed by second, as well as a button to pause the simulation. There are also information displayed about the current generation and simulation step on the bottom.
7. Tests Balancing and Results

7.1. Selection Methods

In order to get better results, several tests were performed with various methods of selection for carnivores and herbivores. Those are:

- Roulette Wheel
- Roulette Elitism \((n = 2, 10\% \text{ of population})\)
- Tournament Elitism \((n = 2, k = 4)\)
- Tournament with Roulette Wheel \((k = 4, 20\% \text{ of population})\)
- Ranked Roulette
- Truncated Roulette \((\text{best 50}\% \text{ of population})\)

All selection methods cited above were tested using different methods for the populations. It was observed that more aggressive methods lead to better results, causing the population to excel in relation to another. In a scenario where population \(A\) uses an Truncated Roulette and \(B\) uses Ranked Roulette, population \(A\) is more likely to prevail.

7.2. Fitness function

Different fitness functions were tested, and the main among them were: Steps survived by individual. Steps survived by individual summed with the remaining power of the individual. Steps survived by gene (including children generated after reproduction by excess energy).

Of these three main models, the third options was adopted because it has a better reflection on the survival of a particular strategy. This is a valid assumption since the gene may represent more than one individual in the simulation, which is something that occurs when an individual has enough energy to break up into two new individuals.

7.3. Balancing

Some changes were made to the proposed scenario in order to improve game balance. Initially, both carnivores and herbivores generated new plants when starved. In this case, herbivores had great performance, but the carnivores could not survive for long, regardless of the method of selection chosen. The system was then changed so that herbivores did not generate more plants. Despite of diminishing above advantages, herbivores were still able to avoid carnivores until most of them died, leaving a scenario with only a few predators and lots of food.
Trying to increase the supply of food for carnivores, cannibalism between carnivores was enabled. If a carnivore \( A \) was near another carnivore \( B \) with life less than or equal to a given value, the carnivore \( A \) would try to eat \( B \), and the energy of the carnivore \( B \) would add to \( A \). In this test, carnivores started to compete with each other in a great deal. As the objective is to observe the evolution of antagonistic populations, this change was disregarded. An acceptable balance was reached by changing the amount of energy provided by plants to 30, the amount spent each turn to 2 and allowing only herbivores to generate new plants when they starve.

7.4. Decision Making Process

An individual determines his actions from a decision tree, whose inputs are derived from the individual visual sensors. Considering the carnivorous represented in the center of Figure 4(a), highlighted with a blue circle for easy identification, sensors indicate the presence of carnivores, herbivores and plants in the vicinity. Figure 5 shows one possible decision tree for a carnivore. The would-be traversed path through the tree if it was the carnivore highlighted in Figure 4(a) is indicated in red.

After deciding the action to be performed in accordance with the sensors, the action will be processed according to weights assigned to visible individuals based on its distance and the action being performed. Figure 4(b) shows the weights assigned to each individual in the field of view of the carnivore outlined in Figure 4(a).
The individual will pursue the point with the greatest weight. If this point is one of the points around the individual, it shall occupy this position, if not, it will move to the nearest position from the target. In the example, the carnivore will move as close as possible to herbivore with weight 5. The grid after the movement of the carnivore is shown in Figure 4(c).

7.5. Results

The algorithm took about 30 seconds to simulate 15 generations. The maximum time per step limit is removed when running on commodity computers. Despite performing adequately compared to other genetic algorithm implementations, it is still too slow to be used in real time. It is worth noting that the algorithm was not extensively optimized. Regarding the behavior of individuals, as noted by Figure 4(a), some alternated peaks of population were evident. In Figure 6, one can observe that, when the population of herbivores has a peak, the population of carnivorous drops (e.g. the generation 5). For the simulation used to generate the graph in Figure 6, Elitism Roulette selection methods for carnivores and Ranked Roulette for herbivores. The graph shows that the population of carnivores, on average, is kept alive longer than the population of herbivores, which is due to the fact that carnivores are not eaten by other individuals.

![Fig. 6. Performance of the species during simulation](image)

The Y-axis of the graph represents the fitness of the individual, which is the number of steps it has survived. Considering that the initial energy of each individual is 100 and that each step the individual takes two points of energy, the individual can last up to 49 steps before it feeds or dies. The average population of carnivores is mostly in the range between 50 to 60 steps, while the maximum of each generation is mostly between 60 and 80 steps. This shows that, although there is a lot of internal competition, some carnivores are having success in hunting their prey. The fitness of the herbivores population varies between 20 and 40 steps. In some generations, individuals managed to survive over 120 steps, with no significant increase in the average population. This is due to the fact that only a few herbivores can escape the initial attacks of carnivores. However, when individuals from both populations begin to starve, herbivores that ate at least one plant have a huge advantage. The number of predators is severely decreased, carnivores who have not eaten are already dead from starvation, less competition for food, some herbivores have died of starvation, and larger amounts of food in the form of plants generated by dead herbivores can all be accounted as advantages for the remaining herbivores. Carnivores were running an Elitism algorithm, which means that the genes of the best individuals of each generation are kept. It was expected that the fitness of carnivores would only increased over the generations. However, as the environment in which the population is inserted changes, this effect is not observed.

8. Conclusions

In this implementation, a scenario was created to allow the simultaneous evolution of two different agents, using one another as an integral part of the environment, consisting on a scheme of co-evolution. A food chain
was established, where the populations of each agent alternately dominated the other, demonstrating continuous adaptation to new strategies developed by their opponents. In the electronic games area, creating opponents that can adapt to the behavior of the player is a major challenge, since it plays a key role in the observed level of enjoyment of the player. The use of evolutionary techniques can thus serve for the creation of intelligent agents that continuously adapt to the level of the player, maximizing the entertainment provided. The biggest challenge is to achieving this is the amount of processing required to achieve significant progress in the quality of the agents, as seen in the existing literature on the subject. Note that, disregarding this impediment, genetic algorithms seem to be able to produce quality sufficient for the proposed objective.

9. Future Works

9.1. Improved Gene Modeling

In order to improve the quality of the results and also to generate more complex behaviors, the Gene Modeling should be improved. New entries must be added to the gene of individuals, such as if the individual is hungry or not.

9.2. Changes in the Scenario

With the creation of new scenarios, different types of behavior could and should emerge according to the differences in dynamics between individuals. One possible change would be to include new types of individuals, forming a food predatory chain and possibly cyclical.

References

[1] J. Schell, The art of game design a book of lenses, 1st Edition, Elsevier/Morgan Kaufmann, 2008.
[2] M. Mitchell, An Introduction to Genetic Algorithms (Complex Adaptive Systems), third printing Edition, A Bradford Book, 1998.
[3] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, 1st Edition, Addison-Wesley Professional, 1989.
[4] A. E. EIBEN, J. E. SMITH, Introduction to Evolutionary Computing, Vol. 1, Springer, 2003.
[5] K. Moriwaki, N. Inuzuka, M. Yatnada, H. Seki, H. Itoh, A Genetic Method for Evolutionary Agents in a Competitive Environment, Springer London, 1998.
[6] S. Akers, Binary decision diagrams, Computers, IEEE Transactions on C-27 (6) (1978) 509 –516.
[7] J. R. Koza, Evolution and co-evolution of computer programs to control independently-acting agents, in: From animals to animats : proceedings of the first international conference on Simulation of Adaptative Behavior (SAB), MIT Press, 1991.
[8] M. Ebner, R. Watson, J. Alexander, Coevolutionary dynamics of interacting species, in: Applications of Evolutionary Computation, Vol. 6024, Springer Berlin Heidelberg, 2010, pp. 1–10.
[9] I. Watson, A. D., Y. Chuyang, W. Pan, G. Chen, Optimization in strategy games: Using genetic algorithms to optimize city development in freeCiv (2010).
[10] V. Mole, Algoritmos genéticos - uma abordagem paralela baseada em populações cooperantes (2002).
[11] A. Fernandez, C. Cotta, R. Ceballos, Generating emergent team strategies in football simulation videogames via genetic algorithms (2002).
[12] G. Kendall, K. Spoerer, Scripting the game of lemmings with a genetic algorithm, in: Evolutionary Computation, 2004. CEC2004. Congress on, Vol. 1, 2004, pp. 117 – 124 Vol.1.
[13] E. Lopez, J. Swafford, M. O Neill, A. Brabazon, Evolving a Ms. Pacman Controller Using Gramatical Evolution, 2010.