Brock University

Department of Computer Science

Real-Time Competitive Evolutionary Computation

Adam Hewgill
Technical Report # CS-02-17
June 2002

To be presented at GECCO-2002, NYC, July 2002.

Brock University
Department of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

Real-Time Competitive Evolutionary Computation

Adam Hewgill

Computer Science Department
Brock University
St. Catharines, ON
ahewgill@hotmail.com

Abstract

In this paper, a system of evolutionary computation will be presented that uses real-time competition to evolve individuals that effectively utilize the environment in which the competition takes place. This system can be used to create computer-controlled enemies for gaming systems with little programmer effort. The evolved individuals will be able to react to changes in the environment in real-time, which is necessary for most user interactive applications.

1 INTRODUCTION

When using evolutionary computation techniques such as Genetic Algorithms (GA) (Holland, 1992) or Genetic Programming (GP) (Koza, 1992) you must calculate the fitness of each individual in the population before a generation can be computed and the evolution can continue. Once the fitness is obtained, the algorithms perform the natural selection steps (crossover, mutation and/or replication) to form the next generation of the population. The fitness scores and the system structure both guide the evolution of the individual chromosomes toward an optimal solution. This continues for a specified number of generations, and once completed, returns the best chromosome encountered over the entire run of the algorithm. Fitness is one of the most important parts of the evolutionary strategy, because it directly relates to what traits are desirable in the population. This correlation is what guides the evolution towards the optimal traits. The fitness is essentially a “score” for the individual represented by the chromosome.

The research presented in this paper uses competition between every individual in the population to produce the fitness value for each chromosome. Previous work by Karl Sims (Sims, 1994) had individuals compete in one-on-one competitions over a resource to gauge their fitness. The approach used here, however, is for every individual to compete against every other individual for a fixed length of time. The evaluation rounds are essentially mass melees during which time each individual scores points by attacking and/or killing other members of the population. The goal of the evolution is to produce an individual that uses the competition environment to its fullest, and is essentially a very good autonomous unit that could be used as a computer-controlled entity. These initial experiments use very simple environments and representations for the individuals in order to test the viability of the real-time competitive model.

The first experiment used a GA to evolve an array of floats [0...1), each element of which was mapped to a characteristic of a 3D fish, which inhabited a boundless virtual aquarium. The elements of the array represented things like: size, speed, color, turning agility, strength, etc. For the evaluation of the chromosomes, the fish would each attack each other, and at the end of a certain number of animation frames the GA would produce the next generation. This application evolved a fish that utilized the environment efficiently due to its superior chromosome. The actual results of this experiment are interesting because they show that the best individual may not always be what is expected even in a simple system.

In the second experiment, a GP was used to evolve "brains" for robots. Each brain was essentially a decision tree that contained if-then nodes and states as leaves. The states were simple actions like: run toward, run away, idle, run randomly, etc. When the competition round was in progress, each individual had their "brains" scanned in order to determine the action that best suited the individual at the current time in the simulation. The environment in which the robots compete is a very simple 2D plane, and uses only a minimal set of rules that robots must follow. Evolution resulted in a very simple but effective strategy for a consistent high score in the simple environment.

2 GENETIC ALGORITHM EXPERIMENT

In the first experiment in mass competition, the initial goal was to create fish that fought with each other; a decision was made to use a GA to evolve the best fighting fish to use in the aquarium.[1]

2.1 AQUARIUM ENVIRONMENT

The virtual aquarium that the fish occupy is a simple boundless 3D area in which the fish may compete. Fish can only turn at a certain rate and cannot simply stop on a dime, turn 180°, and start in a new direction. This provides a simple challenge for the fish and adds to the realism of the simulation. When a fish comes in contact with another fish, both of these fish are attacking each other until they are no longer touching. This is a very simple system but provides some interesting problems for the fish. Firstly, a fish cannot just attack any other fish, because if the attacking fish is weaker then it will die before the bigger fish. Secondly, fish that are smaller do not stand much of a chance, even though they get a speed bonus, because there is no way for them to compete in the overall simulation. See Figure 1 and 2 for screenshots of the system.

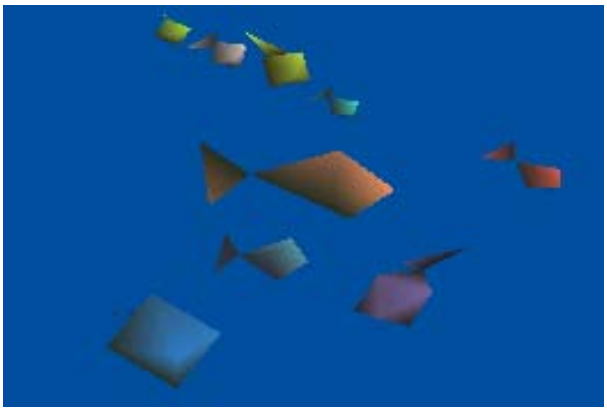

Figure 1: Virtual Aquarium during Competition

2.2 COMPETITION ALGORITHM

The competition of the fish is all based on a function the fish use to select another fish to attack. The only thing the fish do during a frame is to select a fish to attack and move toward it one unit based on their speed. The target selection heuristic is a driving force for evolution of the fish. Since all fish use the same target selection function the weakest fish are killed quickly, causing them to have low fitness scores. The result is that only the strong fish survive, just as evolution dictates.

Once a fish dies, it is dead for the rest of the evaluation round and can no longer gain fitness. The competition rounds end after a specified number of frames of animation, and then the GA carries on to the next generation.

The target selection function uses the properties of the other fish to give a score to each fish. Once every fish is scored the highest valued fish is selected to attack. The score is based on: distance, size, speed, relative size, relative speed, relative strength and color. The closer a fish is the more attractive if is since less distance must be traveled to catch it. Relative size, speed, and relative strength both compare the selecting fish's size, speed and strength to the fish that is being scored. If the other fish is bigger, faster or stronger then it will score low. Bright coloring is given a higher score since it is very attractive to other fish. So fish with bright colors are more likely to be selected.

2.3 FITNESS EVALUATION

Each fish is given one point for every frame that it is alive and the rest of the evaluation comes from killing other fish. One hundred points is issued to a fish that kills another fish.

The total of frame points and kill points is used as the fitness score of an individual fish during the evolution step of the GA.

2.4 CHROMOSOME REPRESENTATION

Each fish is created from a chromosome comprised of an array of floating point numbers in the range [0...1). There are eleven numbers in the chromosome, which are used to build a fish. The attributes of the fish are: speed, size, attack strength, hit points, turning agility, RGB characteristics of the ambient and diffuse properties of the material used when lighting the fish in OpenGL. All of these characteristics are used to define what a fish can do and how attractive it is to other fish.

2.5 GENETIC OPERATORS

The crossover, mutation and replication operators used are very standard. There was no research done towards which types of crossover and mutation operators were best and only the crossover rate and mutation rate were changed while trying to improve the results of the simulation.

“Two Point” crossover is used to create the two children from the parents' chromosomes. This is a very simple way to produce children that are a mixture of their two parents and quite different from their sibling. It was found that a crossover rate of 1.0 produced better fish than when the rate was lower.

[1] The experiment (executable on windows) is available for download at http://www.cosc.brocku.ca/Offerings/3P98/programming/GAfish/GAfish.zip

The mutation is a simple swap of two floating-point values in a chromosome that is applied after crossover. Two distinct indices into the array are selected and the values at these positions are swapped. A mutation rate of 0.2 was used. When new chromosomes are created using crossover or replication, there is a 20% chance that the chromosome will become mutated.

The mutation operator that is currently being used is not optimal and could easily be improved. An example of a simple improvement would be to randomly choose an index into the chromosome and replace the value with a new randomly generated float in the range [0...1). This mutation would not only produce the desired "small" change to the chromosome, but would introduce new random numbers into the population which are normally static after the initial population creation.

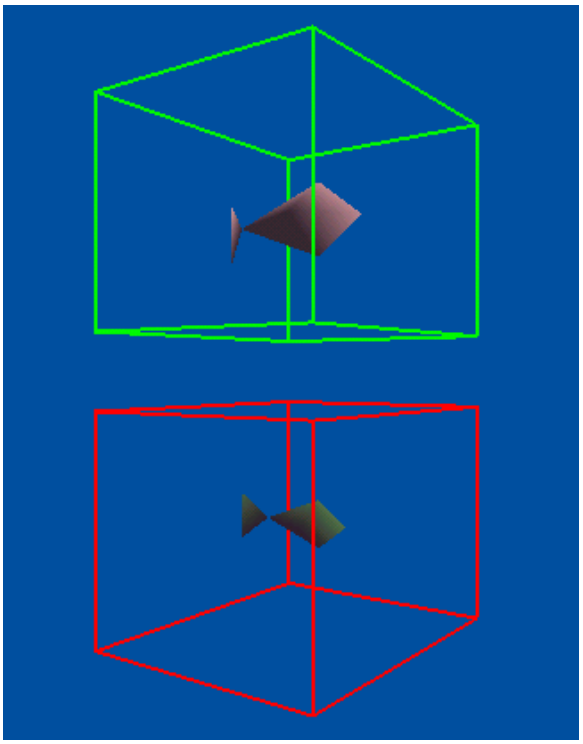

Figure 2: Best (Top) and Worst Fish Of Generation

2.6 RESULTS

The results for this experiment were very interesting and were quite unexpected. The fish did learn to adapt to the competition environment but not in a manner that would be obvious to a programmer. Numerous experiments were run and the best chromosomes from each were analyzed for similarities. When looking at the averages for each gene in the chromosome it turned out that size, speed, strength and hit points were high while turning agility was low and color was in the middle. Size and speed were not as high as strength and hit points and in a few instances were low. Color was not very consistent but when averaged showed that a medium color was desirable.

After the averages were examined the standard deviation of the chromosomes from the average was analyzed. These standard deviations were the most interesting result because they showed that three of the attributes were more important than the rest. Strength, hit points and turning radius all had standard deviations around 10% where the rest of the genes ranged from 19% to 29%. The fish discovered that these genes were more important to have within the optimal ranges.

The results were found while using a population size of 100 over 100 generations where each competition round lasted approximately 20 seconds. The averages and standard deviations were calculated using 10 runs with the previously mentioned parameters.

3 GENETIC PROGRAMMING EXPERIMENT

The next experiment was designed to improve upon the results of the previous system by making some modifications to the competition algorithm until better results were found. The overall complexity of the environment was also simplified to give the search a better chance at finding an optimal solution. A GP was used to simulate "brains" for each individual robot in order to allow for different personality traits. This was a large improvement over the previous system where all fish used the same target selection algorithm. The final results for this experiment were very good.

3.1 ENVIRONMENT OVERVIEW

The robots must follow the rules of the system when acting on what their "brains" decide to do. Each robot has a fixed amount of energy to use during the simulation. Movement, attacking, and being attacked all decrease the energy of the robot until it can no longer do anything. The robot can rest and gain back energy, but it cannot do anything else during this time and is open to attack by other robots. Robots can only fire a certain maximum distance and then have to wait for several frames before their weapons are ready to use again.

When a robot is killed, it is brought back to life and randomly positioned in the environment so it can continue competing during the evaluation round. This rule helps to avoid the “unlucky” factor where a good fish might be killed early and not have a chance to prove itself.

If a robot is not moving when it is shot by another robot, the shot inflicts full damage. If the robot is moving then only half damage is done. This rule exists to make motion a desirable trait and mimics real life since it's harder to hit moving targets.

3.2 FITNESS EVALUATION

During the evaluation round each robot in the population collects points by competing with other robots in the environment. There are two ways to get points: attacking and killing. In order to balance out another problem with the previous experiment points were given for attacking another robot. These points add a better score gradient to the population. Each robot also gains points for each kill it makes.

3.3 CHROMOSOME REPRESENTATION

Each individual robot used a chromosome of the GP for its "brain". The chromosome was a tree made up of nodes and leaves which is a typical setup for a GP chromosome. The nodes of the tree were "if-then statements" which asked various yes/no questions. The leaves of the tree were states that the robot would go into. For every frame of the animation, each robot would be asked what state it is going into.

The "if-then statements" implemented were: if-tired, if-enemy-close and if-state-is. The first was true if the robot's power level was lower than ten percent of its maximum power and false otherwise. The second was true if another robot was within a certain distance and false otherwise. The third was more complicated because it was a three-position if statement. The first value was the state to compare the robots current state against, the second was the true branch and the third the false branch. This third "if-then statement" added a great deal of complexity to the trees and improved the results of the experiment.

The states that made up the leaves were: idle, rest, fire, runaway, runrand and runto. The first state would cause the robot to do nothing at all, which meant not moving, firing nor recharging. The second state was the same as idle, except the robot would recharge, gaining back some of its lost energy. The third state was the attack state, which caused the robot to try to shoot at its closest enemy if it was in range. The last three states are movement states, all of which cost the robot energy and move it either away from the closest enemy, toward a random location or toward the closest enemy, respectively.

3.4 GENETIC OPERATORS

All of the GP genetic operations and tree evaluation were done using the *lil-gp* GP system (Punch-Zongker, 1995). All of the nodes and leaves were specified and written, then given to the *lil-gp* system to run.

3.5 RESULTS

Initially, the best robot "brain" was simply: constantly rest until another robot came by to shoot at. Once the movement damage rule was added this result changed. Instead the results tended towards robots that moved, even though the resting robots still scored highly.

A typical best individual for the experiment was a very good utilization of the system and its simple rules. The robot would run toward its closest enemy and then fire at it until it died. It would then continue on to the next closest enemy in an endless pattern. The actual "brains" were, in most cases, too complex to analyze so the observed results are described after watching the individual in competition. Figure 3 is a representation of a typical individuals "brain", which has been simplified to be slightly more human readable.

```
(if-enemy-close
  (if-state-is
    (if-state-is RUNTO REST RUNTO)
    IDLE
    (if-state-is
      (if-tired RUNTO FIRE)
      IDLE
      (if-state-is REST FIRE RUNTO)))
  (if-state-is
    FIRE
    (if-tired IDLE REST)
    (if-state-is IDLE RUNRAND RUNAWAY)))
```

Figure 3: "Brain" of Best Individual

4 GENERAL COMPETITION ALGORITHM

In general, the competition of individuals in the population works well to evolve better results for an evolutionary system. This has been seen constantly in nature and has been shown in evolutionary systems by the works of previous researchers (Angeline-Pollack, 1993).

When the whole population competes in a mass melee there are certain considerations that must be taken into account so that the evolution will generate good results. When these things are properly setup, the resulting best individual utilizes the competition environment very efficiently and when placed in competition with a randomly generated population, will quickly dominate.

Firstly, when an individual dies, it must be respawned so that it has an opportunity to gain points throughout the competition round. This will prevent the statistical clobbering of deviant chromosomes and allow the population to improve via evolution instead of remaining average.

Secondly, the scoring gradient must be set up well. It is not efficient to have individuals competing over a kill and have the "lucky" one who gets the killing blow have all the points. It is much better to give all the robots a portion of the points of the kill depending on the amount they contributed to the attack. This will cause the scores to be nicely distributed instead of very granular. In an evolutionary system, the better the fitness gradient is, the better the evolution will perform.

Another very important aspect of the competition is the building blocks from which the individuals build their "intelligence". It is a well-known fact in evolutionary computation that the smallest adequate set of building blocks is the most effective. If the components being used are so specific that they exactly specify the final behavior desired, then the system will not evolve anything interesting and will give disappointing results. In other words, primitives can negatively bias the search. It is quite possible that what the biased primitives will find a solution that is much worse than the best solution, since the best solution might not always be logical to human understanding. This has been found in other systems where computer generated heuristics are better than the best possible human created solution (Prieditis, 1993).

Lastly, we define an interface as the connections the individuals are able to use to interact with the competition environment. If the interface to the system is too complex or too simple, then the competing individuals will not have a very good chance of finding an optimal way of using the system to its full advantage.

In the case of a simple interface, the individual might not be able to adapt to certain very important features of the system since it cannot explore them during the competition. An example of the simplicity problem would be target selection. If there is a primitive for target selection, such as pick closest, then there is no way for the individual to pick a different target which is not as close but is much weaker; which would be desirable.

If, instead, the interface is too complex then the search space for the evolution is greatly magnified, and the chance of an individual finding the important features of the interface and utilizing them efficiently is very low, leading to disappointing results.

It is also important that the evolved individuals compete in real-time. This is due to the fact that it is necessary for the individuals to react to the changing environment in which they compete. When these best individuals are used in the system against/with a user they must react in real-time.

5 FUTURE WORK

It would be an interesting test to see how complex the rules of the universe could become and still have the competition evolve a strong best individual. The ultimate goal being to see how well the competition could evolve robots for a multiplayer first person shooter that has very complex 3D rules and interaction methods for the individuals. Alternatively, any system that needs to have a computer-controlled user could simply evolve one quickly and effectively.

If this were to work well, then this system could be used to create enemies and computer controlled users for complex gaming or other systems to save on valuable programmer time.

6 CONCLUSIONS

Real-time competitive evolutionary computation is an effective process for evolving individuals who utilize an environment very well. It is useful for creating computer-controlled opponents for gaming systems or for other applications. Overall, the evolutionary system could help save programmers time for more important aspects of the application, like usability and feature implementation.

There are a few pitfalls that must be avoided but once they are successfully circumvented, the system's results will be good and quite consistent.

The evolved individuals operate in real-time and can respond to user changes immediately. This real-time aspect is a must for applications like gaming system.

Acknowledgments

Thanks to GECCO Student Travel Award Program for the funding assistance supplied to bring this research to the 2002 Genetic and Evolutionary Computation Conference.

Conference entry fee and other travel expenses paid for through NSERC grant 138467.

References

Holland JH. (1992). *Adaptation in natural and artificial systems*. Cambridge, MA: MIT Press

Koza JR. (1992) *Genetic programming: on the programming of computers by means of natural selection.* Cambridge, MA: MIT Press.

Sims K. (1994) Evolving 3D Morphology and Behavior by Competition. In *Artificial Life IV Proceedings, ed.* R. Brooks & P. Maes: MIT Press, 28-39.

Punch B., Zongker D. (1995) lil-gp genetic programming system http://garage.cps.msu.edu/software/lil-gp/lilgp-index.html

Angeline PJ., Pollack JB. (1993). Competitive Environments Evolve Better Solutions for Complex Tasks. In *Proceedings of the 5th International Conference on Genetic Algorithms*, ed. by S. Forrest, Morgan Kaufmann 264-270.

Prieditis AE. (1993) Machine discovery of effective admissible heuristics. In *Machine Learning Vol.12* :117-141.