TJHSST Computer Systems Lab
Senior Research Project
Final Research Paper
Applications of Genetic Algorithms
2008-2009

Mary Linnell

June 1, 2009

Abstract

The purpose of this project is to explore the applications of genetic algorithms, an evolutionary computation search technique, to find approximate solutions to optimization problems. This project will focus on computing the minimum point on a three dimensional graph. The goal is to find the minimum point without testing every single point on the graph, a very computational intensive process. This project will explore various population sizes on the efficiency and effectiveness of finding the answer to the problem.

Keywords: genetic algorithm, artificial intelligence, evolutionary algorithm, optimization, OpenGL, C

1 Introduction

Genetic algorithms are used as adaptive algorithms to solve problems of natural evolution. They can be used for a large number of different purposes, not limited to biology, ecology, and genetics. Some problems can be solved using genetic algorithms that would have too large of a search size to solve using other simple techniques, such as depth first search, breadth first search, and A* search. Genetic algorithms can be used effectively to solve optimization

problems, as the population of possible solutions evolves over time to get an answer that is approximate, but very close to the exact solution.

There are many applications for genetic algorithms. Some examples include playing an Othello AI, solving the N queens problem, and optimizing the traveling salesman problem. In fact, any problem that can be characterized by a fitness function can be solved using genetic algorithms. My project focused on solving a graph optimization problem.

In genetic algorithms, the size of the population is a variable. This variable can effect the speed and effectiveness of the genetic algorithm. If the population is too large, then the algorithm will run slowly, and if the population is too small, there will not be enough possible answers in the gene pool to produce accurate results.

Another parameter is the selection size. If the population size is larger, the selection size should be larger, and vice versa. In this project, the selection size was a constant based on a percentage (25%) of the population size.

My project focuses on the effect of population size on the accuracy of the results, while keeping the run time to a reasonable length.

1.1 Scope of Study

The software application will require a good knowledge of OpenGL 3D graphing, as well as general knowledge of the C programming language for genetic algorithms. The research that will be required is how to optimize the genetic algorithm to achieve the best results when running multiple trials.

When run, the program displays wire-mesh graph, a set number of randomly chosen points on the graph (the population), a results panel, a blank chart at the bottom, and several buttons to click on to control the simulation. The buttons include a "Reset" to initialize all variables, a "Step" to cycle through the steps of the genetic algorithm, a "One Iteration" button to run each of the steps of the genetic algorithm, a "Single Trial" to run multiple iterations of the genetic algorithm on the current set of points, and a "Multiple Trials" button that changes the seed after each trial and outputs the results of multiple trials.

1.2 Expected results

The results obtained using genetic algorithms should approximate the exact result obtained using calculus. The results will be analyzed by seeding the

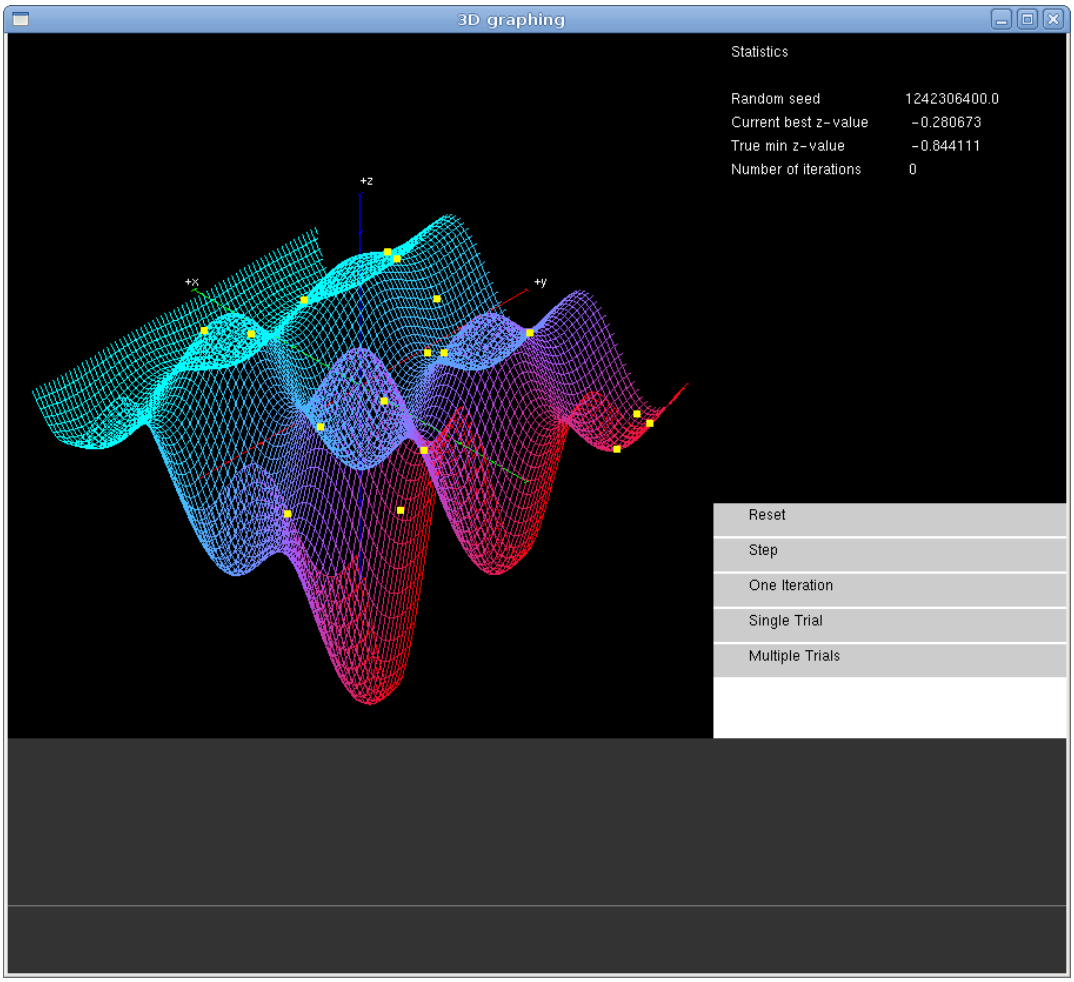

Figure 1: The UI for the genetic algorithm application.

random number generator differently and running multiple trials, and then changing the size of the population and retesting.

The results of a trial can be determined by analyzing the chart displayed at the bottom of the screen. This chart shows the progression of the best z-value (the fitness function) for each iteration of the genetic algorithm.

The results will also be analzyed by averaging 20 runs of the genetic algorithm. The mean will be computed and compared to the true value to arrive at the difference from true value.

2 Theory

2.1 Definition

A genetic algorithm is a theory used to compute approximate solutions in fewer iterations than other search techniques. It is based upon a structured evolutionary biology, with a "population" containing "individuals."

2.2 Set up

The population in this genetic algorithm is composed of points, where each point is an "individual."

2.3 Selection

The fitness function determines which points are the "best." In this case the fitness function will determine the lowest points by evaluating the known function and assigning a value (z value) to each point in the population.

A set number of the population will be killed off in each iteration. The fitness function determines which individuals of the population will be killed off, which will always be the members with the highest fitness function value.

2.4 Crossover

A crossover point was chosen within the population of points. The new individuals in the population were chosen based on an algorithm of duplicating, cutting, and slicing the existing population.

2.5 Mutation

The random mutation helps maintain genetic diversity in the population from one generation to the next. This factor can be important to avoid becoming trapped in a local minimum instead of the absolute minimum.

2.6 Exact Answer Calculation

In Mathematica, we perform the following calculations:

We begin by defining the function as

$f[x\_, y\_] := -(Cos[x * 5.0] + Cos[y * 5.0])/5.0$

Then we call the FindMinimum function.

$FindMinimum[f, \{\{-1, 1\}, \{-1, 1\}\}]$

and Mathematica returns

$\{-0.4, \{-1.25664, -1.25664\}\}$

where -0.4 is the minimum z-value for this function.

3 Procedures and Methodology

I am using C with OpenGL to write my program. The OpenGL is used for the 3D graphing component, and C is used for the genetic algorithm.

The graph of $z = \frac{-(cos(x*8.0)+cos(y*8.0)*(1-x)*(1-y))}{5.0}$ appears on the screen in a wire-mesh of points. Eight randomly-generated yellow points appear on the screen. They consist of the population. See Figure 2.

1. The step button is pressed. The four worst points (as determined by the fitness function) are highlighted in white and enlarged.

2. Step is pressed again. Those four points have been deleted, because this is the selection process.

3. When the step button is pressed the third time, new points appear (in this version of the program, they happen to be the points that were deleted,

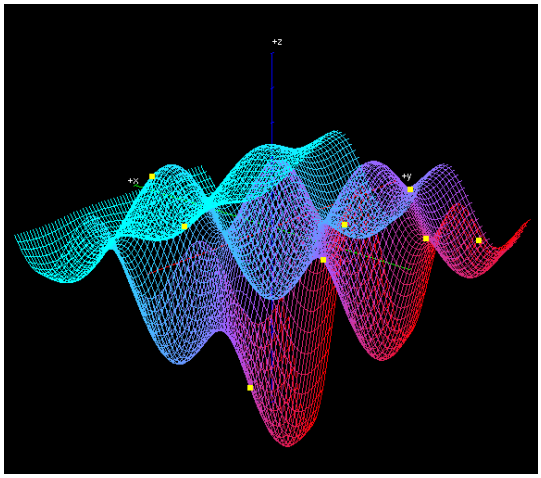

Figure 2: Eight randomly generated points (yellow) on the graph $z = \frac{-(cos(x*8.0)+cos(y*8.0)*(1-x)*(1-y))}{5.0}$.

Figure 3: The chart that graphs the result of a single trial.

but in the future they will be new points chosen based upon a breeding algorithm).

4. The fourth time the step button is pressed, the points stay the same but become the permanent new population.

When the step button is continued to be pressed, the cycle repeats according to the steps above. The "Single Trial" button automates pressing the step button by calling the step function multiple times and displaying the results. The "Multiple Trials" button runs a series of 10 complete, single trials and displays the quantitative results in the terminal.

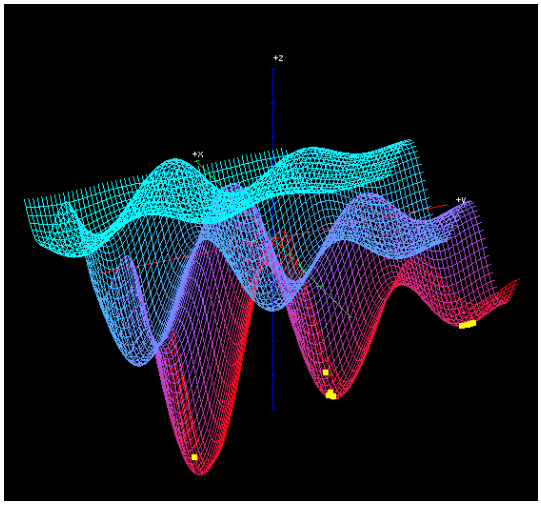

Figure 4: The population tends to converge to the local minima.

4 Local minima

One issue in finding the minimum of the graph is the complications of local minima. The graph that I chose (Figure 2) has lots of local minima. At first, the population converges to the local minima in addition to the true minimum (Figure 4). After a long time, however, the population finds the true minimum and converges there (Figure 5). The random mutation helps the genetic algorithm find the true minimum by exploring other parts of the graph that the population would not normally converge to.

5 Results

The equation of the tested graph was: $z = \frac{-(cos(x*8.0)+cos(y*8.0)*(1-x)*(1-y))}{5.0}$. This graph consists of multiple local minima and maxima, to try to make the optimization problem harder to solve.

Population size: 8
True z-value: -0.84411
Average result: -0.52009
Difference: 0.32403

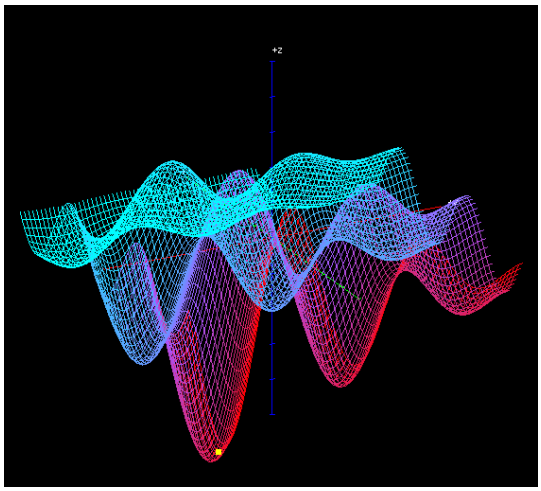

Figure 5: The population eventually finds the true minimum.

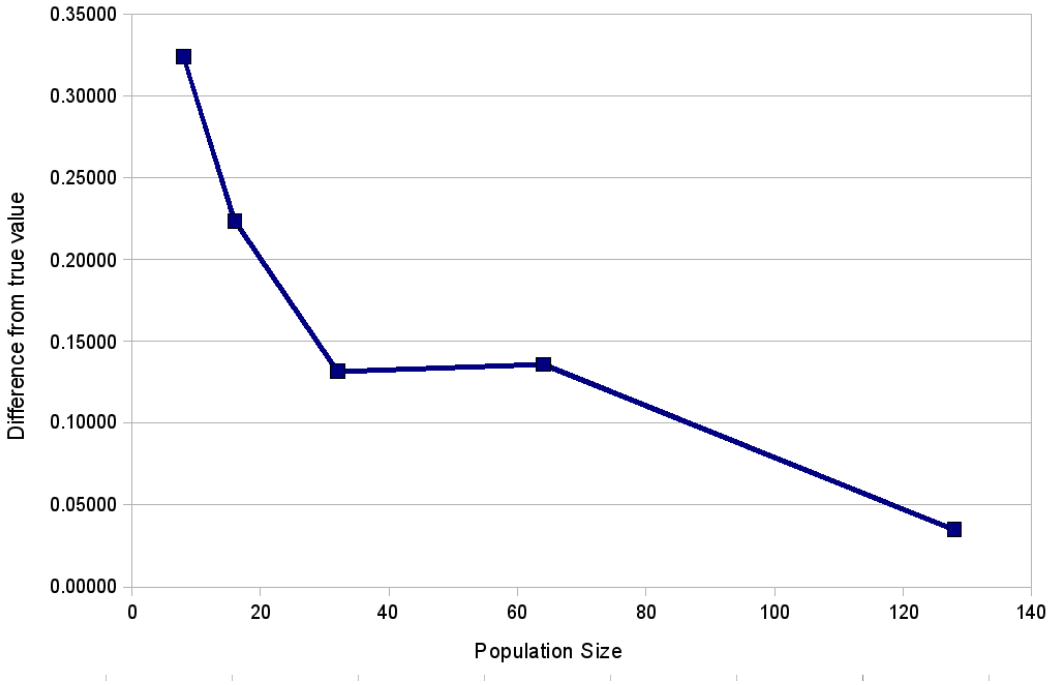

Figure 6: Results in chart form.

Population size: 16
True z-value: -0.84411
Average result: -0.62066
Difference: 0.22345

Population size: 32
True z-value: -0.84411
Average result: -0.71237
Difference: 0.13174

Population size: 64
True z-value: -0.84411
Average result: -0.70805
Difference: 0.13607

See Figure 6 for chart of results.

6 Conclusion

As the population size increased, the accuracy of the genetic algorithm improved. The difference between the true value and obtained value decreased when the population size increased. At first, the trade-off is really good: a small increase in population size yields a large increase in accuracy. As the population size increases, however, a large change is required in the population size to obtain a small change in accuracy.

7 Acknowledgements

I would like to thank Mr. Latimer for guidance and assistance throughout this Computer Systems Techlab project.

References

[1] Ahrens, B. 2005. Genetic algorithm optimization of superresolution parameters. In *Proceedings of the 2005 Conference on Genetic and Evo-*

lutionary Computation (Washington DC, USA, June 25 - 29, 2005). H. Beyer, Ed. GECCO '05. ACM, New York, NY, 2083-2088. DOI= http://doi.acm.org/10.1145/1068009.1068354

[2] Coello, C. A. 2000. An updated survey of GA-based multiobjective optimization techniques. *ACM Comput. Surv.* 32, 2 (Jun. 2000), 109-143. DOI= http://doi.acm.org/10.1145/358923.358929

[3] Wainwright, R. L. 1994. A family of genetic algorithm packages on a workstation for solving combinatorial optimization problems. *SIGICE Bull.* 19, 3 (Feb. 1994), 30-36. DOI= http://doi.acm.org/10.1145/182063.182071

[4] Aranha, C. C. and Iba, H. 2008. A tree-based GA representation for the portfolio optimization problem. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* (Atlanta, GA, USA, July 12 - 16, 2008). M. Keijzer, Ed. GECCO '08. ACM, New York, NY, 873-880. DOI= http://doi.acm.org/10.1145/1389095.1389267

[5] Yu, T. and Lin, W. 2008. Optimal sampling of genetic algorithms on polynomial regression. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* (Atlanta, GA, USA, July 12 - 16, 2008). M. Keijzer, Ed. GECCO '08. ACM, New York, NY, 1089-1096. DOI= http://doi.acm.org/10.1145/1389095.1389294

[6] Lobo, F. G. and Lima, C. F. 2005. A review of adaptive population sizing schemes in genetic algorithms. In *Proceedings of the 2005 Workshops on Genetic and Evolutionary Computation* (Washington, D.C., June 25 - 26, 2005). GECCO '05. ACM, New York, NY, 228-234. DOI= http://doi.acm.org/10.1145/1102256.1102310

[7] Chu, P. H. and Dudley, S. A. 1993. The effect of population structure on the rate of convergence of genetic algorithms. In *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing: States of the Art and Practice* (Indianapolis, Indiana, United States, February 14 - 16, 1993). E. Deaton, K. M. George, H. Berghel, and G. Hedrick, Eds. SAC '93. ACM, New York, NY, 147-151. DOI= http://doi.acm.org/10.1145/162754.162846