Genetic Algorithms:

Solving Simple Mathematic Equalities

Prepared For: Tony White, COMP 4107

Authors: Eric Adamski, 100873833 and James Roose, 100852693

Date: December 4, 2014

1. Introduction

1.1 Problem Statement:

In algebra, there exist problems in which the right-hand side of the equation is known, however, many values on the left-hand size of the equation are not. These types of equations are known as 'equalities'. An example of such an equation would be the following: a + 2b + 3c + 4d = 30. The solutions to these problems have several applications and are required to solve a wide array of problems across all of mathematics.

1.2 Project Description:

The purpose of this project is to solve the types of simple mathematical equalities that are described above with the use of a genetic algorithm. It will be written in Java using the JGAP library. To keep things simple, the equations that will be accepted must be bound by the following format rules.

- All values in the equation must be positive integers (no floating point number will be accepted)
- The only mathematical operator that will be accepted is addition.
- The right-hand side of the equation must be a single integer.

The reasoning behind these constraints will be discussed in section 2.1 of the document. As long as the given equation meets these expectations, the user may enter any equality they wish. There are often many different solutions to the given problem; the algorithm must only come up with one of them.

1.3 Background/Context:

Evolutionary Computation is a class of Computer Science, which generates solutions to optimization problems using techniques that are inspired by natural evolution. Genetic Algorithms (GA) is a subset of this class. A GA is a search heuristic that is meant to mimic the process of natural selection.

Solutions to the problem are represented in chromosomes, each containing genes that hold the related data. These chromosomes are part of a population of chromosomes, otherwise known as a genotype. The fitness of a chromosome determines its likelihood of creating offspring; therefore higher fitness chromosomes will be more likely to have their genes (and thereby, their solution to the problem) passed on to the next generation. Fitness is meant to be a calculation of how optimal a chromosome's solution is. The population is evolved until either a sufficient solution is found or a predetermined generation number has been met.

Another important aspect of GA is the idea of genetic modifiers. An example of a genetic modifier that we use in our system is crossover, which occurs once a chromosome has been selected for reproduction. The chromosome's offspring will contain data from both itself and its 'partner' chromosome, in which it reproduced with. This is meant to mimic the way reproduction works in nature, where a child's genetic makeup comes from a combination of its parents' genetics. Another genetic modifier used in our system is mutation. After reproduction and crossover have been done, there is a chance for a genetic mutation to occur. This results in a change of a single gene within that child, thereby exploring even further potential solutions.

GA has seen plenty of application in the world of mathematics, as well as other domains such as engineering, computer science, economics, physics and chemistry. It has discovered optimal solutions for problems in many of these domains, an example being its discovery of the tit-for-tat solution to the prisoner's dilemma.

1.4 Result Summary:

In summary, our GA is successful at finding solutions to any given problem that meets the requirements listed in section 1.2. Due to the random nature of GA, the number of evolutions required to find a solution to the problem varies, however, the application of crossover and mutation strongly influence this number. Higher rates of mutation generally decreased the amount of generations required to reach a solution, meanwhile medium rates of crossover contributed to a reduced number of generations as well.

2. Project Analysis

2.1 Implementation:

This section of the document is meant to briefly go over what was implemented in our project and why we made these decisions. The system can be divided into two separate components: the formula intake component and the actual GA component.

The creation of the formula intake component added the ability to allow users to enter any formula they want into the system. We wanted our GA to be useful and able to solve many different algebraic problems. However, this component is also responsible for the limitations the system has in terms of what equations it can solve. Due to the way we parse the user's input string, addition is the only operation currently accepted in the system. Additionally, all numbers in the equations must be whole number. This is because the system expects integers on both the GA and formula intake sides. Lastly, we do not parse whatever comes after the equal sign in the equation; therefore the system will only accept equations with a right-hand side consisting of a single positive integer. We decided to keep things simple because our primary focus was on the GA component of the system.

On the GA side of the project, we had to develop a fitness function, a model for the chromosomes and a configuration object. The configuration we decided on was the default configuration. This configuration is an elitist-ranking selector that clones the best 90% of the population and repeats to fill the rest. It then applies a random-point crossover to 35% of the population before randomly mutating 1 in 12 genes. We experiment with the rates of crossover and mutation in section 2.2 of this document and demonstrate how they affect the number of generations required to come to a solution.

The chromosomes were modeled in a simple and straightforward fashion, in which each gene in the chromosome represented a variable in the equation. Therefore, each chromosome has a number of genes equal to the number of variables in the equation.

Lastly, the fitness of a chromosome is simply demonstrated by how close the left-hand side of the equation, given the variable values in the chromosome's genes, comes to equating to the right-hand side of the equation. It is inversely proportional to the absolute difference between the two sides of the equation. For example, the fitness of an agent with values a = 4, b = 3, c = 2, d = 1 would be 1 / Abs ((1 x 4) + (2 x 3) + (3 x 2) + (4 x 1) – 30 + 1) = 1/11. We add 1 to the denominator to avoid division by zero once a solution has been reached.

2.2 Analysis and Discussion:

2.3 Problems and Improvements:

3. Conclusions

4. References

- Hall, M. (2013, February 18). JGAP Default Initialisation Configuration. Retrieved December 4, 2014, from: http://mathewjhall.wordpress.com/2013/02/18/jgap-default-initialisation/

- Hermawanto, D. Genetic Algorithm for Solving Simple Mathematical Equality Problem. Retrieved December 4, 2014, from: http://arxiv.org/pdf/1308.4675.pdf