AI Project 2: Genetic Algorithm

Roy Howie

February 27, 2017

1 The Genetic Algorithm

The Genetic Algorithm is a stochastic, heuristic-based algorithm which draws its inspiration from the Darwinian principle of natural selection. It aims to generate high-quality but not necessarily maximal solutions to optimization and search problems.

The algorithm requires an initial population and a fitness function which it seeks to maximize. The initial population is usually a collection of individuals which have each been given a random genetic makeup. The algorithm is then able to efficiently search through extremely large domains via three search operators:

Survival. This determines whether a given individual lives on to the next generation.

Recombination. Individuals selected for survival are mated and their genetic compositions mixed.

Mutation. A chromosome is occasionally incorrectly copied when passed onto the next generation.

The application of these three search operators to a population is the act of moving from one generation to the next.

Note that each of these three operators can be customized for whichever domain is under consideration.

For example, the **survival** operator could be changed to always include the top quintile in its selection. The **recombination** operator could randomly select the offsprings' genes from each parent or it could ensure an even split. Instead of remaining constant, the **mutation** rate can be intensified or lessened over time. The possibilities are endless.

Last, the algorithm requires a termination criterion, such as having attained a maximal level of fitness within a population or having run for a given number of generations. This termination criterion must take into account the possibility of **premature degeneration**. This is when a population devolves into a small set of similar individuals and is thus no longer capable of genetic diversity through mating. To overcome this affliction, the mutation rate may be momentarily increased to great effect.

2 Implementation

2.1 Fitness Function

The fitness function we sought to maximize was

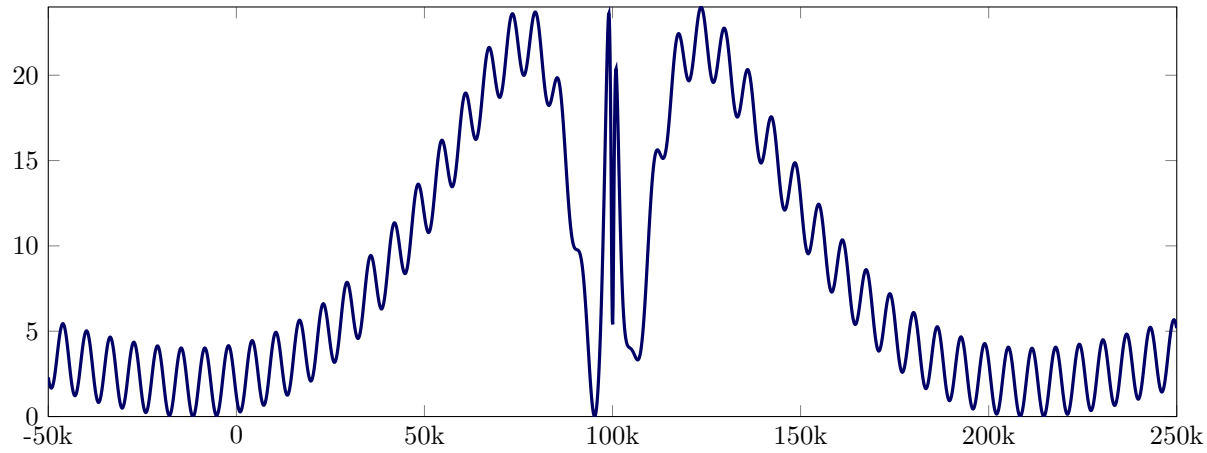
$$f(x) = 12 + 10\cos\left\{\ln\left[0.1 + \left(\frac{x - 10^5}{1000}\right)^2\right]\right\} - 2\sin\left(\frac{x - 10^5}{1000}\right)$$

for all integer x. Note $\inf f = 0$ and $\sup f = 24$.

2.2 Individuals

Individuals are represented via a binary string of 20 bits. Therefore, the range of individuals was all integers between 0 and $2^{20} - 1$.

Figure 1: Graph of Fitness Function



2.3 Search Operators

2.3.1 Survival

Individuals were selected for the next generation using a weighted probability distribution.

For example, denote the current generation as P and let it consist of a list of numbers in the range specified in 2.2. Let F be the corresponding array of fitness values of the members of P and let C be the cumulative sum of F.

Table 1: Example Survival Setup

P	0	25000	75000	99999	123548	-
F	1.21	4.18	21.61	5.32	23.99	-
C	0	1.21	5.39	27.00	32.32	56.31

If k individuals are desired for the next generation, then pick k random numbers $\{ r_1, r_2, \cdots, r_k \}$ between 0 and $\max(C) = 56.31$. Next, for each i, find the least j such that $r_i < C[j+1]$ and include the individual at index j of P in the next generation.

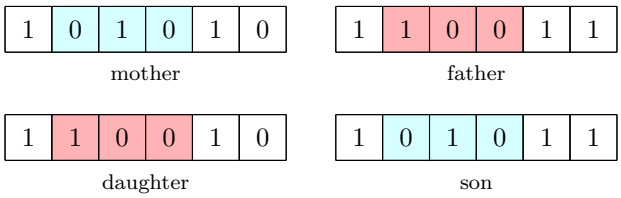
Note that the same individual may be selected multiple times to be included in the next generation, unlike in real life. Indeed, this is the intended behavior: that individuals with higher fitness values shall have a greater chance of surviving and will thus appear in larger number in the generations to come.

2.3.2 Recombination

The 2-point crossover method was used for mating. As individuals were selected for survival, they were put into mother-father pairs. Each mother and father pair was cloned into a daughter and a son, respectively. Then, two random indices $a < b$ were chosen and the binary digits between these indices were then "swapped" between the daughter and son's chromosomes.

For example, suppose the mother could be represented with the sequence of binary digits $m_1 \cdots m_k$ and the father by $f_1 \cdots f_k$. Choose $0 < a < b \leq k$. Then the daughter would be $m_1 \cdots m_{a-1} f_a \cdots f_b m_{b+1} \cdots m_k$ and the son would be $f_1 \cdots f_{a-1} m_a \cdots m_b f_{b+1} \cdots f_k$. This is best seen as illustrated in Figure 2.

Figure 2: Recombination Illustration for $a = 1, b = 3$



2.3.3 Mutation

After mother-father pairs are recombined into daughter-son pairs, each child is given a 1-in-100 chance to mutate. If selected for mutation, each digit in the child's binary representation is given a 1-in-25 chance of flipping, i.e., `BITWISE NOT`.

2.4 Termination Criterion

The algorithm was run until at least one of the following became true:

- ⋆ An individual had attained a fitness value greater than 23.97.
- ⋆ The algorithm had searched for 100,000 generations.
- ⋆ The max fitness level of the past 1000 generations had remained constant.

3 Objectives

- ⋆ Understand how the average and max fitness of a population evolve over time.
- ⋆ Determine how the mutation rate affects a population's fitness over time.
- ⋆ Determine the relationship between population size, number of calls to the fitness function (total number of individuals), and the number of generations needed to find a solution.

4 Results

Figure 3: Average Number of Generations to Termination over 500 Trials

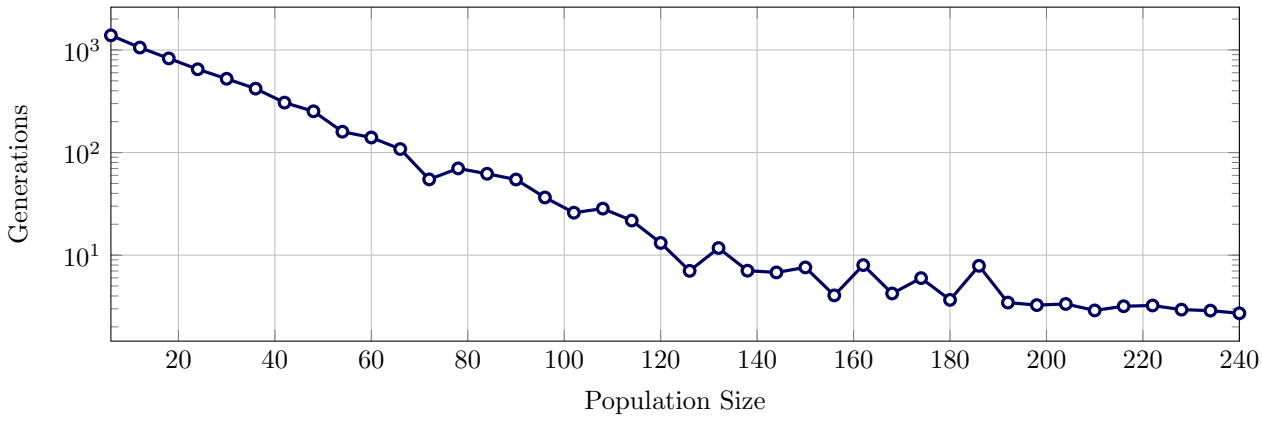


Figure 4: Average Number of Individuals Witnessed over 500 Trials

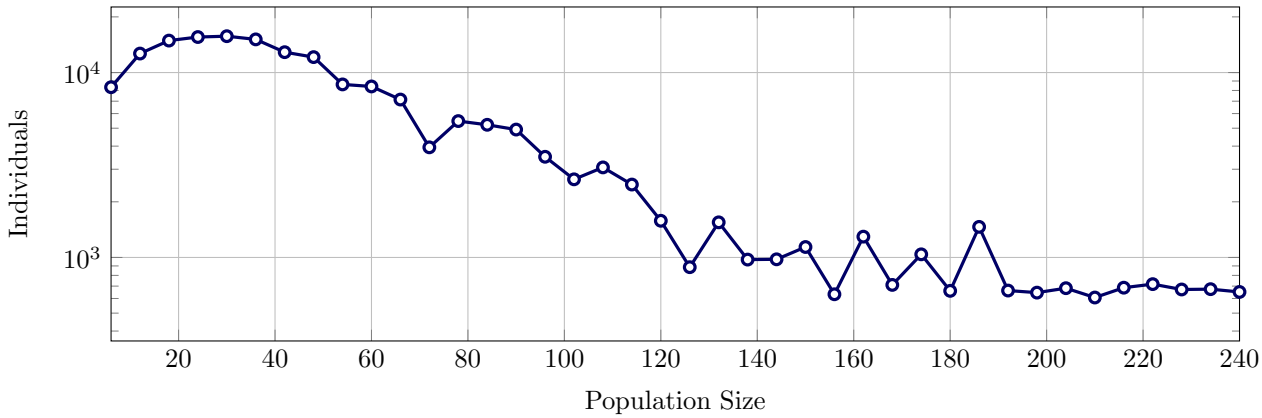


Figure 5: Generations vs. Number of Individuals over 500 Trials

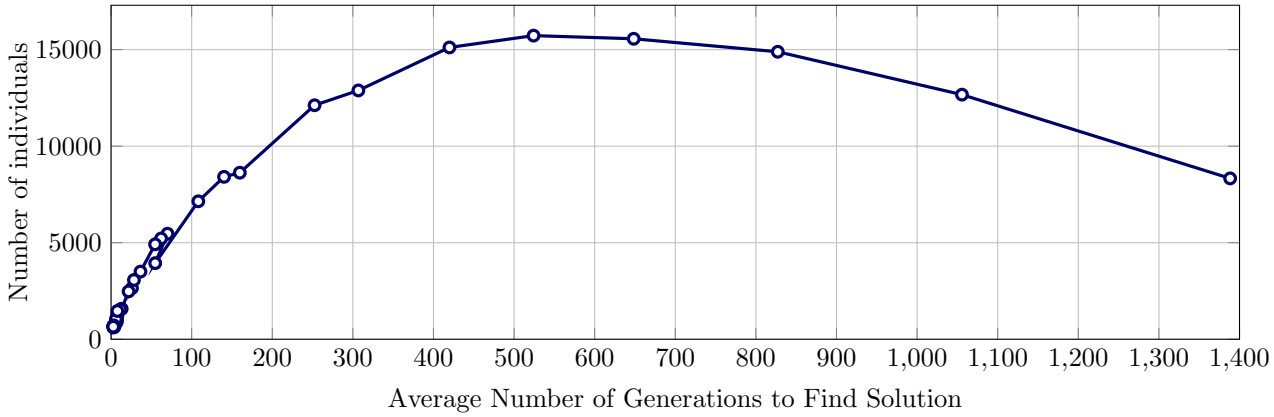


Figure 6: Mean "Average Fitness" for Various Mutation Rates over 500 Trials

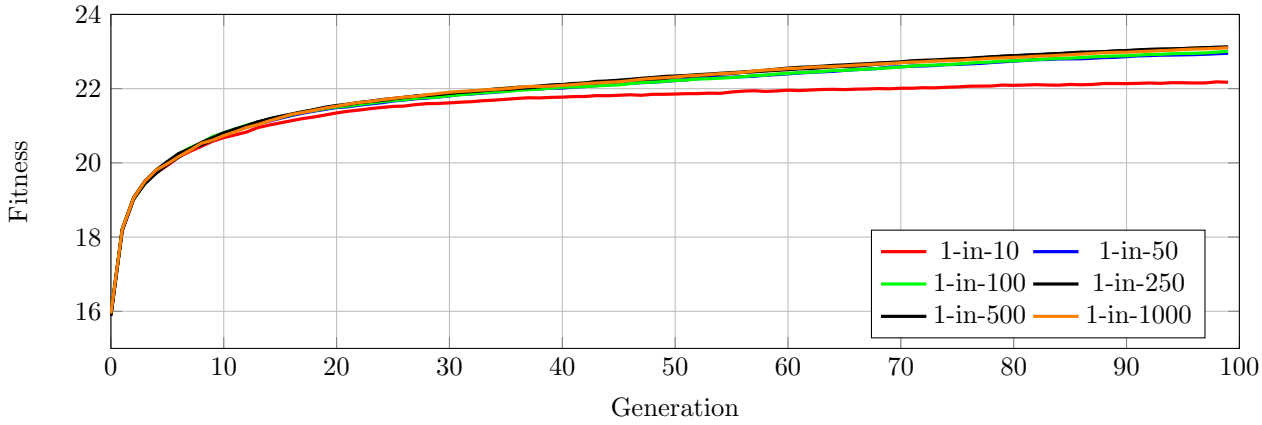
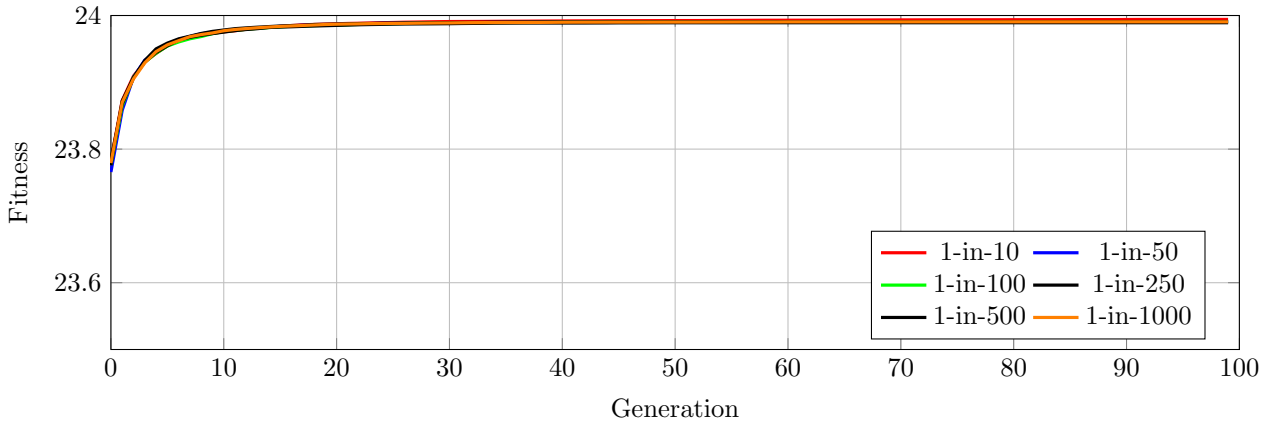


Figure 7: Mean "Max Fitness" for Various Mutation Rates over 500 Trials



5 Discussion

5.1 On the Best Approach

As can be seen in Figures 3 through 7, the ideal implementation of the Genetic Algorithm often consisted of a larger initial population. Indeed, as can be observed in Figure 5, the lower the number of generations needed to find a solution, the fewer individuals observed. This is intuitive and reveals a subtle but important notion: a large initial investment (large population) resulted in a better payoff long-term (less computation).

Furthermore, a mutation rate of less than 1-in-100 was observed to be ideal.

5.2 On Premature Degeneration

Premature degeneration did not occur very often. However, to prevent the algorithm from looping indefinitely, as explained in Section 2.4, we only allowed the algorithm to run for less than 100,000 generations.

In addition, we wanted the max fitness of a population to climb somewhat monotonically over time. Thus, if the max fitness of a population did not increase for 1000 generations in a row, the algorithm was terminated.

Even when premature degeneration did occur, the algorithm had often already found a solution rather close to the one we deemed optimal, i.e., a max fitness level of at least 23.97.

5.3 On Improvements

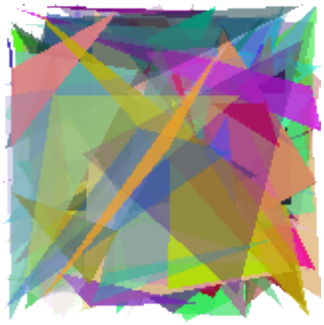
The algorithm was implemented in JavaScript. While the `v8 JavaScript Engine` developed under the Chromium Project and used by `node.js` makes JavaScript rather fast, it is often still no match for `C++`. Furthermore, maximizing a function is not necessarily computationally expensive. Indeed, as will be seen in Section 6, a `C++` implemenation would have allowed us to natively hook into the `gd` Graphics Library and more quickly process images.

6 Bonus: Genetic Programming & Hill-Climbing Search

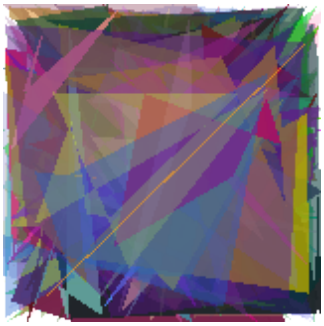
As a bonus, we decided to mix hill-climbing search and genetic programming to see if the computer could "paint" someone. Namely, the author's girlfriend.

Figure 8: Progression of Pictures

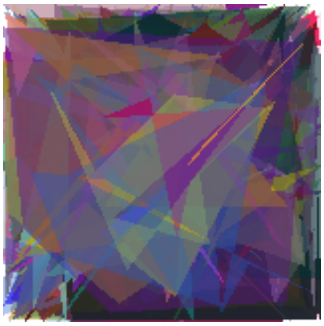
goal



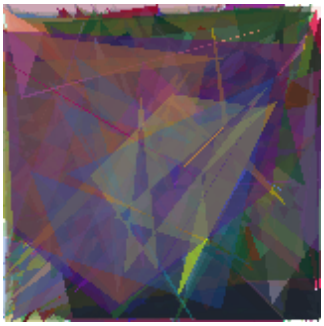
0



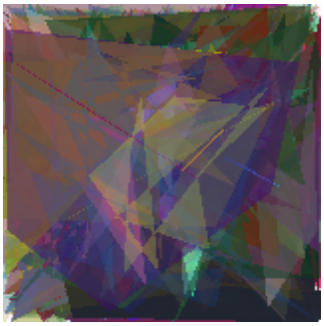
1021



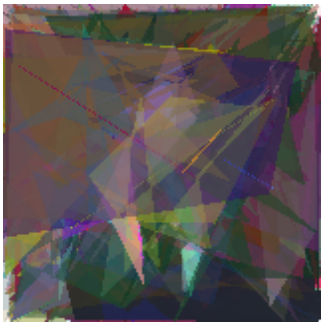
3505



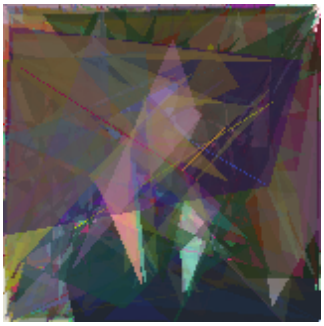
11177



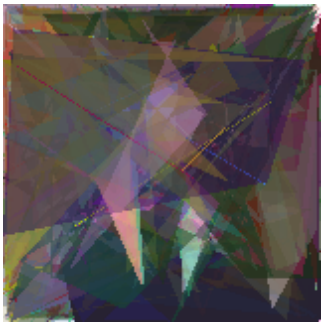
25363



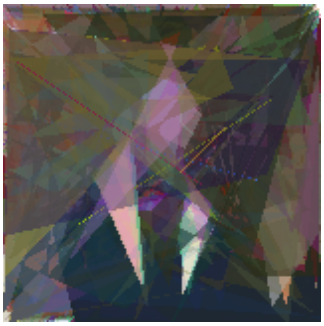
42007



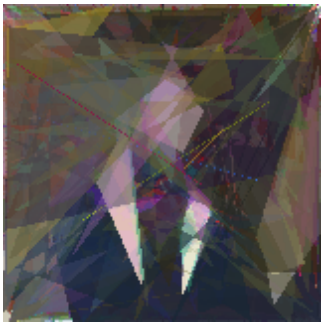
75623



100043



300800



500265

900144