

Genetic Algorithms: Learning by Evolution

Annie S. Wu and Ayse S. Yilmaz
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362
aswu@cs.ucf.edu, selen@cs.ucf.edu

1 Introduction

The genetic algorithm (GA) (Holland 1975) is a learning algorithm based on principles from genetics and evolutionary biology. It is one of several methods in the class of algorithms called Evolutionary Algorithms. Where nature evolves organisms that meet the requirements necessary for survival in a particular environment, GAs evolve solutions that meet the requirements necessary for solving specific problems.

A typical GA works with a population of individuals, where each individual represents a potential solution to the problem to be solved. These potential solutions are evaluated and the better solutions are used to create a new population of potential solutions using genetics-inspired operators. Over multiple “generations” the quality of the evolved solutions will improve.

Key features of a GA include the following:

- A *population of individuals* where each individual represents a potential solution to the problem to be solved.¹ The most common representation format is a linear binary string. Nevertheless, a variety of other structures such as rule sets, vectors of floating point values, and ordered integer lists have also been used.
- A *fitness function* which evaluates the utility of each individual as a solution.
- A *selection function* which selects individuals for reproduction based on their fitness. Selection exploits useful information currently existing in a population.
- Idealized *genetic operators* which explore the search space by forming new solutions out of existing ones. Genetic operators define how encoded information is manipulated and changed by a GA. Crossover and mutation are the most commonly used operators.

Figure 1 shows the basic steps of a GA. The initial population may be initialized randomly or with user-defined individuals. The GA then iterates thru an evaluate-select-reproduce

¹For the rest of this chapter, the terms *individual*, *solution*, and *genome* will be used interchangeably to refer to one potential solution in a GA population.

```

procedure GA
{
  initialize population;
  while termination condition not satisfied do
  {
    evaluate current population;
    select parents;
    apply genetic operators to parents to create offspring;
    set current population equal to be the new offspring population;
  }
}

```

Figure 1: Basic steps of a typical genetic algorithm.

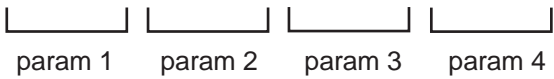
1000101101101111

 param 1 param 2 param 3 param 4

Figure 2: Example of typical binary encoded GA problem representation.

cycle until either a user-defined stopping condition is satisfied or the maximum number of allowed generations is exceeded. Over time, a GA is able to generate better and better solutions to the problem at hand. In the process, a GA must balance the exploration for new solutions via genetic operations and the exploitation of existing good solutions via selection.

1.1 Problem representation

The problem representation refers to how information is encoded as a GA individual. GA individuals are typically linear sequences of characters or values. The most common problem representation format is a binary encoding.

Figure 2 shows a typical binary encoded GA problem representation. The string is divided into multiple fields, in this case four, with each field encoding one component of a solution. One or more bits are used to encode for each component of a solution. Information is encoded on an individual in a location dependent manner, e.g, bits 0 to 3 represent parameter 1; with 0 as the most significant bit (MSB) and 3 as the least significant bit (LSB); bits 4 to 7, parameter 2; and so on.

Although binary encoded problem representation is the most commonly used, many other types of encodings have also been used. Other types of problem representation include, but are not limited to, real valued encodings, multi-character encodings, code, and problem specific encodings (such as rule sets).

1.2 Fitness Function

The fitness function takes as input one individual from a GA population. It evaluates the encoded solution of that individual and returns a numerical fitness value that indicates how good a solution this individual is. Typically, higher fitness values represent better solutions. The design of a fitness function can have a significant impact on the performance of a GA (Jones and Forrest 1995).

1.3 Selection

The selection function drives the exploitation of good solutions found so far by a GA. Given a GA population, a selection function selects those individuals that will pass on their “DNA”, or encoded information, to the next generation population. Selection functions are typically probabilistic with a bias towards selecting highly fit individuals. Individuals with higher fitness are more likely to be selected. Individuals with lower fitness are less likely to be selected. Individuals may be selected more than once or not at all. Note that this description assumes the use of a fitness function that maximizes fitness: higher fitness indicates better solutions. Some adjustments may be needed when using a fitness function that minimizes fitness: lower fitness indicates better solutions.

Common selection methods include the following:

Proportional selection With proportional selection, the probability that an individual will be selected is proportional to its fitness. In its simplest form, the probability that an individual will be selected is calculated as follows. Let $f(i)$ denote the fitness of individual i and

$$f_{sum} = \sum_{i=0}^{pop_size-1} f(i)$$

be the sum of the fitnesses of all individuals in a population. Then the probability that an individual i will be selected, $P_s(i)$, is

$$P_s(i) = \frac{f(i)}{f_{sum}}.$$

Selection typically occurs with replacement. The selection process repeats until pop_size individuals have been selected.

Rank selection Rank selection is similar to proportional selection except that the probability that an individual will be selected is proportional to its rank in the population rather than its fitness. Individuals in a population are sorted and ranked based on their fitness value. Let $r(i)$ denote the rank of individual i . Then $r(i_{most_fit_individual}) = pop_size$ and $r(i_{least_fit_individual}) = 1$. Let

$$r_{sum} = \sum_{i=0}^{pop_size-1} r(i) = \sum_{i=0}^{pop_size-1} i + 1$$

i	Fitness, $f(i)$	Rank, $r(i)$	$P_s(i)$ (Proportional)	$P_s(i)$ (Rank)
0	8	2	$8/132 = 0.06$	$2/10 = 0.2$
1	19	3	$19/132 = 0.14$	$3/10 = 0.3$
2	5	1	$25/132 = 0.04$	$1/10 = 0.1$
3	100	4	$100/132 = 0.76$	$4/10 = 0.4$
$f_{sum} = 132$		$r_{sum} = 10$		

Table 1: Comparison of selection probabilities for proportional and rank selection in a diverse population. $pop_size = 4$.

i	Fitness, $f(i)$	Rank, $r(i)$	$P_s(i)$ (Proportional)	$P_s(i)$ (Rank)
0	80	2	$80/350 = 0.23$	$2/10 = 0.2$
1	95	3	$95/350 = 0.27$	$3/10 = 0.3$
2	75	1	$75/350 = 0.21$	$1/10 = 0.1$
3	100	4	$100/350 = 0.29$	$4/10 = 0.4$
$f_{sum} = 350$		$r_{sum} = 10$		

Table 2: Comparison of selection probabilities for proportional and rank selection in a converged population. $pop_size = 4$.

be the sum of the rankings of all individuals in a population. Then the probability that an individual i will be selected, $P_s(i)$, is

$$P_s(i) = \frac{r(i)}{r_{sum}}.$$

Selection typically occurs with replacement. The selection process repeats until pop_size individuals have been selected.

Behaviorally, rank selection differs from proportional selection in that it tends to exact weaker selection pressure in diverse populations (which tend to occur early in a GA run) and stronger selection pressure on converged populations (which tend to occur later in a GA run).

Tournament selection Tournament selection randomly picks two or more individuals from a population to engage in “competition”. With some probability, the best individual “wins” and is selected. This process repeats until pop_size individuals have been selected. Selection typically occurs with replacement. Mathematically and empirically, tournament selection has been shown to be similar to rank selection (Goldberg and Deb 1991).

Tables 1 and 2 give examples of relatively diverse and converged populations, respectively, and compare the probability that each individual will be selected using proportional and rank selection. When the population is diverse and there is a large spread of fitness values, as shown in Table 1, proportional selection tends to heavily favor those individuals with high

Parents	—>	Offspring
0000000000		0000111111
1111111111		1111000000

Figure 3: One-point crossover using two parents to create two offspring. Crossover point randomly selected to occur after bit 3.

fitness. In the example provided, individual 3 has a disproportionately higher probability of being selected than any other individual in the population. Rank selection minimizes the differences in fitness. When a population is converged and all fitness values are similar, as shown in Table 2, the probability that any individual will be selected using proportional selection is almost identical because all individuals have similar fitness values. Rank selection is better able to distinguish small improvements in fitness in converged populations.

1.4 Genetic Operators

Genetic operators are the engine behind a GA's exploration of a search space. Most genetic operators attempt to generate new information (find new solutions or new partial solutions) without completely destroying all of the existing information. By not completely eliminating existing information, the GA hopes to build upon partial solutions that it has already found, gradually improving the quality or fitness of its evolved solutions. The most commonly used genetic operators are crossover and mutation.

1.4.1 Crossover

Crossover requires two or more parents and exchanges subsections between parents to create one or more offspring. A GA parameter called *crossover rate*, P_x , gives the probability that any two selected parents will undergo crossover. As a result, we expect $P_x * pop_size$ of the offspring to be generated by crossover. If parents do undergo crossover, crossover points are randomly selected, typically with all points having equal probability. If parents do not undergo crossover, they are copied directly into offspring.

Commonly used crossover operators include:

One-point Crossover Given two parent individuals, select one crossover point randomly and exchange the segments to the right of the crossover point. Figure 3 shows an example of one-point crossover.

Two-point Crossover Given two parent individuals, randomly select two crossover points and exchange the middle segment. Figure 4 shows an example of two-point crossover.

Uniform Crossover Given two parents, uniform crossover (Syswerda 1989) exchanges a variable number of segments to create one or two offspring. Initially, each offspring is

Parents	—>	Offspring
0000000000		0011110000
1111111111		1100001111

Figure 4: Two-point crossover using two parents to create two offspring. Crossover points randomly selected to occur after bits 1 and 5.

Parent	—>	Offspring
0000000000		0000001000

Figure 5: Example individual with mutation at bit 7.

assigned one parent as its *current parent*. At each bit, there is a probability p_x that crossover will occur. If a randomly generated number between 0.0 and 1.0 is greater than p_x , the *current parent* remains unchanged. If the randomly generated number is less than p_x , crossover occurs and the *current parent* switches to the other parent. The offspring then receives its bit from the *current parent*. As a result, uniform crossover can generate a variable number of crossover points.

1.4.2 Mutation

Mutation is a single parent operator that changes individual bits on the members of a GA population. Mutation occurs on a per bit basis, with the parameter *mutation rate*, P_m , indicating the probability that each bit of an individual will be mutated. As a result, the expected number of mutations on each individual is $E_m = P_m \times length$ where *length* is the length of an individual. Figure 5 shows an example of a mutation operation.

2 Two Example Applications

We describe two example problems to which GAs have been applied and discuss implementation issues for each problem. It is important to note that the implementation decisions discussed here are examples of what may be done, but are not necessarily the only way in which a GA can be applied to these problems.

For both problems, we will use a basic generational GA with binary encoded individuals. The initial population is randomly generated. The two parts of a GA that must have knowledge about the problem to be solved is the problem representation and the fitness function. How these two components are designed can have a significant impact on GA performance. After the representation and fitness function have been decided and implemented, some experimental tuning of GA parameters may be necessary to optimize performance. Common measures of GA performance include the best fitness achieved in a run and the number of generations required to achieve a given fitness value.

Because a GA is a pseudo-random algorithm, multiple runs of a GA on the same problem will likely produce multiple different results. It is good practice to examine GA performance over multiple runs to draw conclusions. A minimum of 100 runs for each experiment is recommended to ensure a good sampling of behaviors.

2.1 The OneMax Problem

The OneMax problem is a toy problem that has often been used to study GA behavior. The goal of the OneMax problem is to find an individual consisting of all ones. The experiments described here use individuals with a genome length of 150 bits. Thus, the goal of the GA is to find an individual consisting of 150 ones.

2.1.1 Problem Representation

The OneMax problem typically assumes that individuals are fixed-length binary strings.

2.1.2 Fitness Function

In this example, the goal of the GA is to maximize fitness.

We would like to maximize the number of ones on an individual. The fitness function counts the number of ones on an individual and returns that value as its fitness. As we use individuals of length 150 in these experiments, the optimum fitness value is 150.

2.1.3 Parameter Settings

We use the following GA parameter settings in our experiments:

Population size	200
Crossover rate	0.8
Crossover type	Two-point
Mutation rate	0.01 per bit
Selection method	Fitness proportional
Maximum number of generations	200

2.1.4 Experimental Results

We perform 100 runs of the GA on the OneMax problem and summarize the results here. Figure 6 plots the best and average fitness achieved in each generation from one example run. The best fitness (of a generation) increases quickly at the beginning of a run and gradually levels off later in the run. The average fitness of a generation follows slightly behind the best

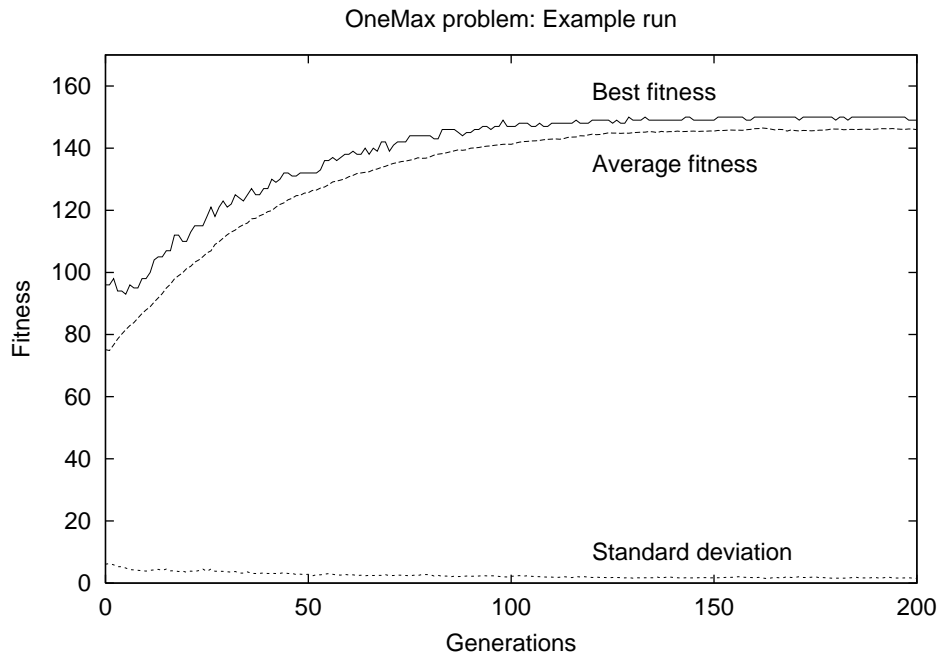


Figure 6: GA performance on OneMax problem from one example run.

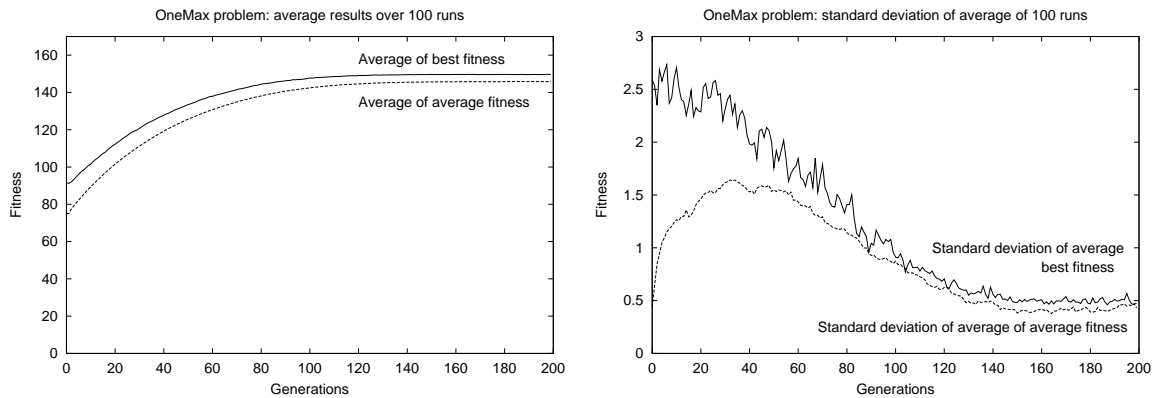


Figure 7: Average GA performance on OneMax problem over 100 runs and standard deviation.

fitness. The standard deviation decreases throughout the run, indicating that the fitness of the members of a population converge and become more similar as a run progresses. Figure 7 shows the average behavior of the GA over 100 runs. The average number of generations required to find the first optimal solution is 117.25 with a standard deviation of 9.57.

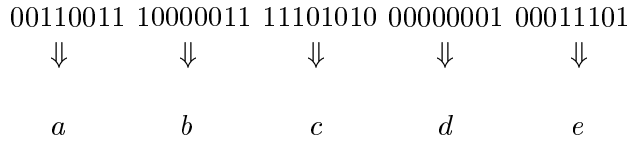


Figure 8: Problem representation for symbolic regression problem.

2.2 Symbolic Regression

Suppose we are given a set of data points in terms of (x, y) coordinates and the equation

$$y = f(x) = a x^2 + b x + c + d \cos(x) + e \sin(x).$$

We would like to find values for the coefficients $a, b, c, d,$ and e such that $f(x)$ most closely approximates the given data points.

2.2.1 Problem Representation

Figure 8 shows our selected problem representation. We want to find five numerical values. Each individual consists of five groups of bits, each encoding the value for one coefficient, $G(n), n \in \{a, b, c, d, e\}$. As each group consists of $l(n) = 8$ bits, we can represent $2^{l(n)} = 2^8 = 256$ values for each coefficient. Given the maximum, $V_{max}(n)$, and minimum, $V_{min}(n)$, values for each coefficient, we calculate the encoded value of each coefficient as follows. Let $B(n)$ be the binary value of $G(n)$. The encoded value $V(n)$ is calculated using

$$V(n) = \frac{B(n)}{2^{l(n)}} \times (V_{max} - V_{min}) + V_{min}$$

2.2.2 Fitness Function

In this example, the goal of the GA is to maximize fitness.

We would like to find coefficients $a, b, c, d,$ and e that results in an equation $f(x)$ such that, given any value of x , return a value $f(x)$ that is as close to y as compared to the data point coordinate (x, y) .

The fitness function calculates sum of the differences between $f(x_i)$ and y_i for all $i : i = 0, 1, \dots, n$ data points:

$$f = \sum_{i=0}^n |f(x_i) - y_i|$$

As the goal of the fitness function is the minimize the differences, in this fitness function, smaller fitness values indicate better individuals. An optimum individual would match the target data points perfectly, resulting in a fitness value of zero.

Population Size:200						
	fitness	a	b	c	d	e
<i>f</i>		2	250	-22	- 252	248
run 1	1103.6	2	252	-73	-203	129
run 2	1714.3	2	254	-108	-157	-26
run 3	1741.5	2	253	-94	-176	-2
run 4	1792.9	2	252	13	-236	65
run 5	2425.7	2	250	-145	-135	229

Table 3: Fitness values of the best individuals of the best 5 runs among 30 and the a, b, c, d, e coefficients found for 3 different functions showing the population size effects

2.2.3 Parameter Settings

We use the following GA parameter settings in our experiments:

Population size	200
Crossover rate	1.0
Crossover type	One-point
Mutation rate	0.01 per bit
Selection method	Tournament, size=5
Maximum number of generations	200

2.2.4 Experimental Results

We use the following function to generate data points for our experiment:

$$f(x) = 2x^2 + 250x - 22 - 252 \cos(x) + 248 \sin(x).$$

As a result, an optimum solution would have the following coefficient values:
 $a = 2, b = 250, c = -22, d = 252, e = 248$.

Table 3 gives the best solution generated by the best five runs out of 30 runs. The optimum values for the coefficients a, b, c, d, e are given in bold characters at the top of the table. Each row below lists the fitness and coefficients of the best solution found by one run. Figure 9 shows the plots of each of the solutions from Table 3 and how they compare with the the target data points. This plot shows that all five evolved solutions are very close to the optimum solution.

Figure 10 plots the best fitness of each generation and the average fitness of each generation averaged over 30 runs. Figure 11 plots the best fitness of each generation and the average fitness of each generation in a closer look for the first 20 generations. It also shows the standard deviations corresponding to the average of Best and Average fitness values.

On average, the population converges to the optimum solution after only 10 generations and remains unchanged for the rest of a run. Increasing mutation rate or the population size may introduce more diversity which would in turn discourage premature convergence.

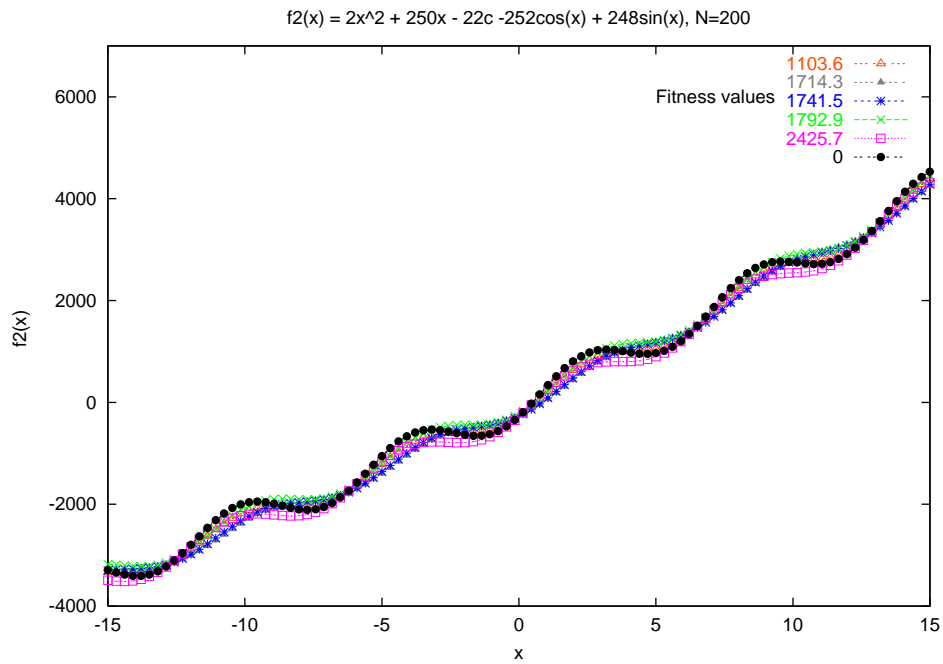


Figure 9: Plots of the best functions found along with the original function indicated with fitness of 0.

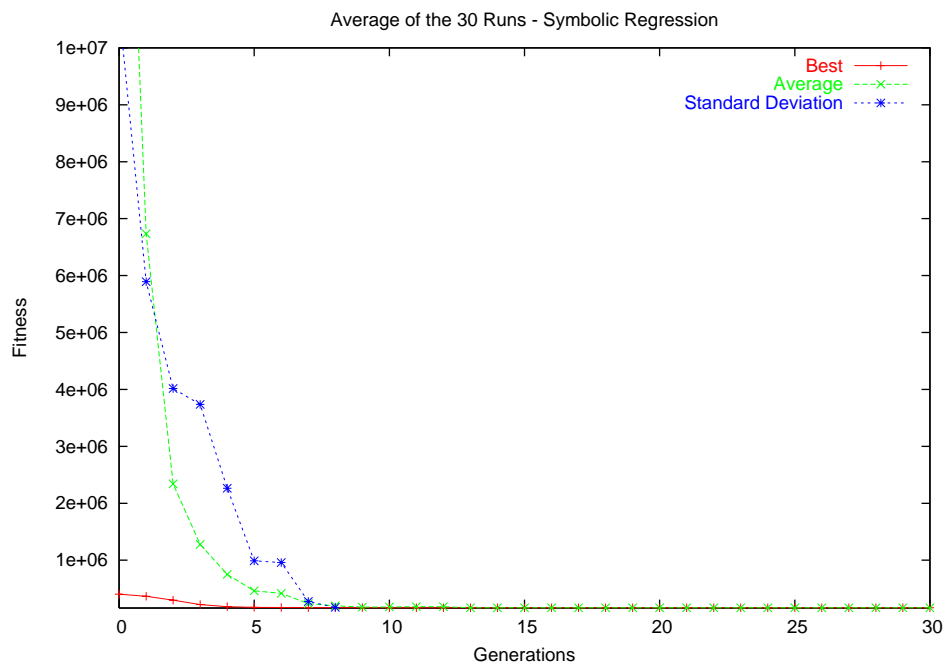


Figure 10: Best fitness, average fitness and standard deviation corresponding to each generation averaged over 30 runs.

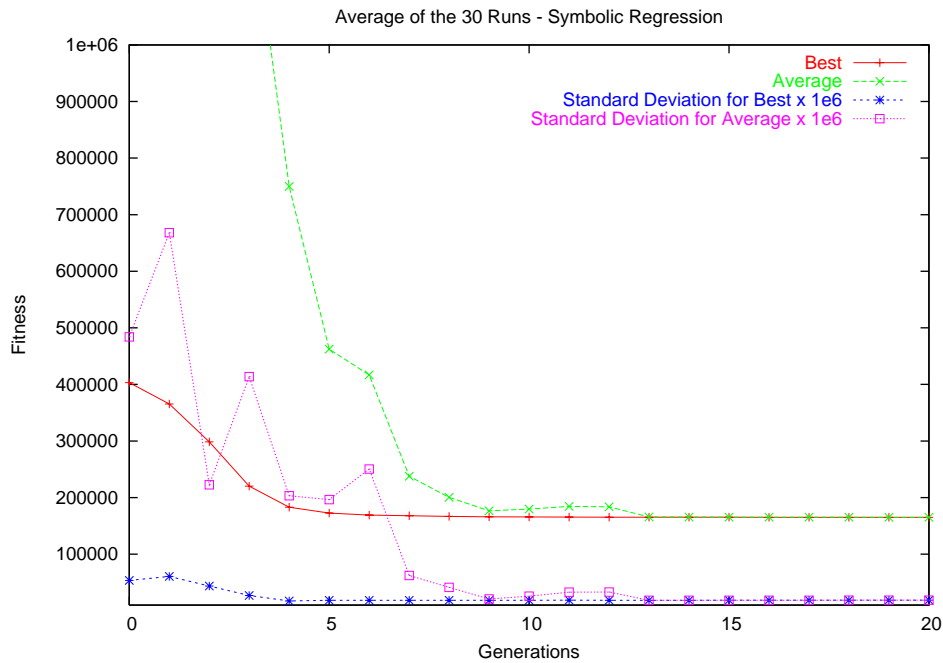


Figure 11: Best fitness, average fitness and standard deviation corresponding to each generation averaged over 30 runs.

References

Goldberg, D. E. and K. Deb (1991). A comparative analysis of selection schemes used in genetic algorithms. In G. J. E. Rawlins (Ed.), *Foundations of Genetic Algorithms*, pp. 69–93. Morgan Kaufman.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Jones, T. and S. Forrest (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proc. of the 6th International Conference on Genetic Algorithms*.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proc. of the 3rd International Conference on Genetic Algorithms*, pp. 2–9.