Exploring Genetic Algorithm Efficacy

I. Introduction

Genetic algorithms are a type of artificial intelligence algorithm which has many parallels to biological evolution. The goal of a genetic algorithm is to "evolve" some solution to a problem based on how fit that solution is [4]. Over time, a collection of proposed solutions will converge to an actual solution—provided the genetic algorithm parameters are suitable.

These types of algorithms rely on several parameters to function: using a basic genetic algorithm as an example, it may incorporate differing crossover rates, mutation rates, selection strategies, and crossover types. The selection of appropriate parameters ensures its function, although inappropriate selection may not prove fruitful for results [4].

This report is intended to explore the selection process for these parameters and determine idealized parameters for a simple key decryption genetic algorithm. Many parameters and permutations of parameters are used, and their success is ranked.

II. Background

Key decryption is the process of deciphering a cipher-key based on some text it has encrypted. Provided is some length of a cipher-key as well as some encrypted text that key has encrypted: to approximate a solution, a genetic algorithm is employed to "evolve" random candidates for the key based on its fitness as a cipher-key.

The general procedure for the genetic algorithm employed is as follows:

```
procedure GA:
  initialize random population p[n]
  for 1..m do:
    sort p by fitness
    select e[j] from p
    select t[k] from p
    sort t by fitness
    erase p
    add e[] to p
    for j..p/2 do:
      create chromosome c1, c2
      crossover c1, c2
      mutate c1, c2
      add c1, c2 to p
    endfor
  endfor
endprocedure
```

Where `n` is the population size, `m` is the number of generations, `e[j]` is a collection of elites, `t[k]` is a tournament and `c1`, `c2` are chromosomes.

Chromosomes are individuals within the population and are encoded with genes or alleles. A chromosome of some length contains that many genes. Genes are valid from within a pool of predefined upper and lower bounds: for this decryption algorithm, the valid genes are alphabetic characters (`a-z`) including hyphen (`-`) which comprise the chromosome encoded as a `String` type.

Two other procedures are employed: one each for mutation and crossover. For a genetic algorithm, successive generations after the first need to employ some genetic operators to create new chromosomes [1]: typically, these are crossover (sometimes referred to as recombination) and mutation.

Provided two "parent" chromosomes, crossover is the process of combining portions of each parent to create offspring chromosomes which partially resemble both parents [1]. Three differing crossover operators are used: 1-Point, 2-Point, and Uniform, with an additional option to randomize the operator used for each chromosome [1].

Below are two example parent chromosomes of six genes as well as how each operator creates an offspring:

Figure 1.

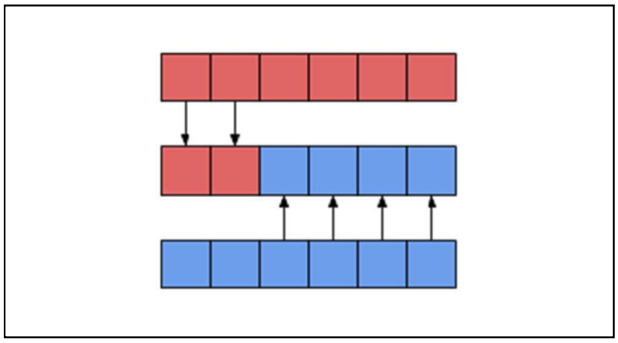

Fig. 1. 1-Point Crossover Operator

For 1-Point crossover, a single pivot point is chosen between `0..c`, where `c` is the chromosome size. When recombining parent chromosomes, genes from one parent are taken prior to the pivot, whereas genes are taken from the other after the pivot [5].

Pseudocode for the 1-Point crossover operator is below:

```
procedure 1P (mother, father):
  -- c is the chromosome length
  pivot := random integer 0..c
  chromosome childA, childB
  for i := 0..c do:
    if i < pivot do:
      childA[i] := mother[i]
      childB[i] := father[i]
    else do:
      childA[i] := father[i]
      childB[i] := mother[i]
    endif
  endfor
  return childA, childB
endprocedure
```

These two children chromosomes will be a combination of both `mother` and `father` based on the randomized pivot.

FIGURE 2.

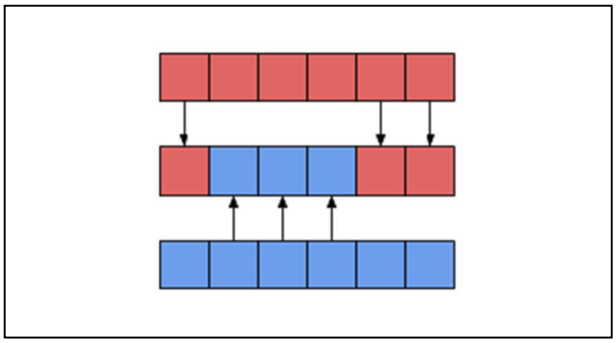

Fig. 2. 2-Point Crossover Operator

For 2-Point crossover, instead two pivot points are chosen. Like 1-Point, each parent chromosome contributes up to a pivot point, where it swaps parents to take genes from [5].

Pseudocode for the 2-Point crossover operator is below:

```
procedure 2P (mother, father):
  -- c is the chromosome length
  pivotA := random integer 1..c
  pivotB := random integer pivotA..c
  chromosome childA, childB
  for i := 0..c do:
    if i < pivotA do:
      childA[i] := mother[i]
      childB[i] := father[i]
    else if i < pivotB do:
      childA[i] := father[i]
      childB[i] := mother[i]
    else do:
      childA[i] := mother[i]
      childB[i] := father[i]
    endif
  endfor
  return childA, childB
endprocedure
```

FIGURE 3.

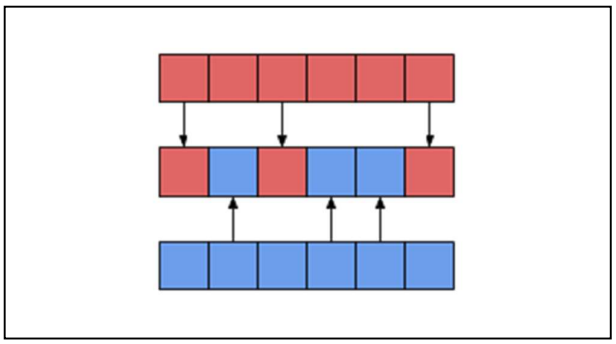

Fig. 3. Uniform Crossover Operator

Uniform crossover, rather than using pivots to determine points where genes are inserted for each parent, provides a randomized chance for either parent to contribute to the new chromosome [5]. Every child chromosome produced this way is a random amalgam of each parent.

Pseudocode for the uniform crossover operator is below:

```
procedure Uniform (mother, father):
  chromosome childA, childB
  -- c is the chromosome length
  for i := 0..c do:
    if random integer 1..2 is 1 do:
      childA[i] := mother[i]
      childB[i] := father[i]
    else do:
      childA[i] := father[i]
      childB[i] := mother[i]
    endif
  endfor
  return childA, childB
endprocedure
```

Conventions continue similarly for 3-Point, 4-Point, et cetera.

In the prior three figures, one child chromosome is used for illustration when two children are created. Using uniform crossover as an example, wherever a gene from the "father" chromosome is transposed into the child, the opposing gene (from the "mother") is transposed into a second child. Crossover produces two children per operation.

A driving procedure determines which type of crossover operator to use as well as the chance of performing crossover:

```
procedure Crossover (mother, father):
  if random double 0..1 < chance do:
    if type is 1P do:
      return 1P (mother, father)
    else if type is 2P do:
      return 2P (mother, father)
    else if type is Uniform:
      return Uniform (mother, father)
    endif
  endif
endprocedure
```

Where chance is the crossover rate. There is another type of crossover employed: random. If this parameter is chosen, which crossover operator used is randomized for each `mother-father` pair.

Mutation, contrastingly, is the process of introducing noise to a chromosome. In other words, genes are transposed verbatim except mutated genes, which for this application are then randomized based on lower and upper bounds (valid gene range) [3].

FIGURE 4.

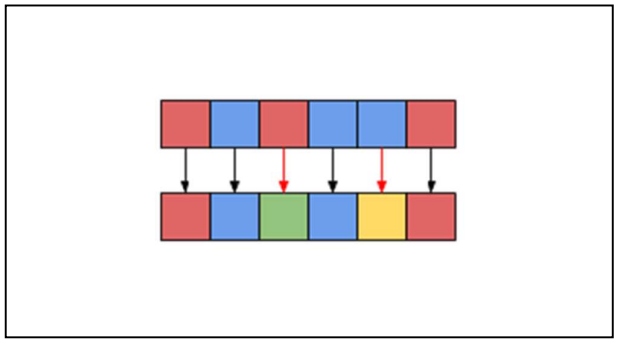

Fig. 4. Uniform Mutation Operator

In the prior figure, two genes have mutated within the chromosome, becoming randomized genes in the process.

Mutation is represented as the following pseudocode:

```
procedure Mutate (chromosome):
  -- c is the chromosome length
  for i := 0..c do:
    if random double 0..1 < chance do:
      chromosome[i] := random gene
    endif
  endfor
endprocedure
```

Two different mutation operators are employed: uniform and non-uniform. In uniform mutation, the mutation rate remains static throughout each successive generation, whereas with non-uniform mutation, this rate of mutation changes with time [3]. However, the process of mutation is identical: changing genes into randomized ones.

Two parameters for the genetic algorithm directly influence when crossover and mutation occur: the crossover rate and mutation rate respectively. These are given as percentages represented as a `double` between `0.00..1.00`. Were both parameters zero, no mutation or crossover would occur, and were they both one, every chromosome child will have both operators process them. Finding some parameter for both between these two extremes and their effect on performance of the algorithm will be examined later.

The population of each generation is likewise defined as a parameter. The population is represented as a collection of chromosomes and how each population changes with successive generations is paramount in converging to a solution. An example sorted population and how it changes between generations is below:

FIGURE 5.

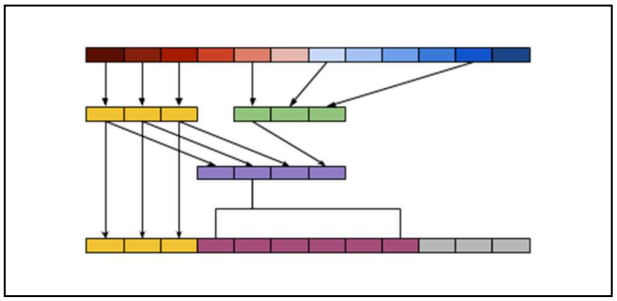

Fig. 5. Population through Generations

In the figure above, the initial population is sorted by fitness (more red means more fit, more blue means less fit). Elites are chosen (yellow) and a tournament is chosen (green). A mating pool (purple) is made from these collections. For this visualization, three elites and a tournament size of three is used for the population of twelve.

Elites are chromosomes which are the most fit of a generation. Based on user parameter, several elite transfer into the mating pool for the next generation. Tournament members, however, are chosen at random, and the most fit of this tournament also transfers into the mating pool.

The previously identified elites are brought over into the next generation verbatim, whereas 90% of the rest of the population is generated through crossover and mutation operators between members of the mating pool. Two parents are chosen at random from within this pool: this process is repeated for each new chromosome. In the above figure, these child chromosomes are colored burgundy.

The final 10% of the population (colored in grey), however, are novel chromosomes introduced into the new population to promote diversity [2]. They may or may not share any commonality with the prior generation and instead are generated randomly.

The procedure for new population initialization per each generation is as follows:

```
procedure GeneratePopulation():
  -- at this point, p is sorted
  for i := 0..j do:
    e[j] := p[i]
  endfor
  for i := j..(j+k) do:
    -- n is p.size
    t[k] := random chromosomes p[i..n]
  endfor
  p.erase
  tWinner := t.sort[0]
  matePool[] := e[] + tWinner
  for i := (j+k)..(0.90 * n/2) do:
    -- m is unique from f
    m := random chromosome in matePool[]
    f := random chromosome in matePool[]
    childA, childB = Crossover(m, f)
    Mutate(childA)
    Mutate(childB)
    add childA, childB to p
  endfor
  for i := (0.90 * n)/2..n do:
    add random chromosome to p
  endfor
  p.sort
endprocedure
```

The above pseudocode is the general basis of the genetic algorithm and ensures the population is continually evolving as in the below figure:

FIGURE 6.

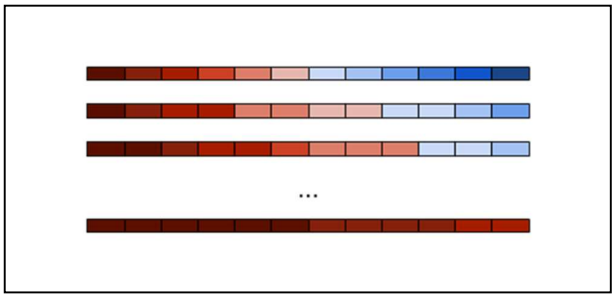

Fig. 6. Population Evolves to Solution

This selection strategy ensures a new population is at least partially comprised of prior most fit chromosomes (thus reducing the chance of regressing onto less fit populations) [2], children of elites and/or most fit random candidate chromosome (promoting diversity between children) and some random chromosomes (ensuring diverse genes are present in future mating pools).

In practice, this selection scheme ensures that over time, the entire population will converge to a solution [1] and the average fitness of the population will increase.

III. Experimental Setup

The provided source code consists of three files: runGeneticDecipher.java, GeneticDecipher.java, and KeyScorePair.java. Execution should be performed in a Linux environment and can be achieved with the below two commands:

```
$ javac runGeneticDecipher.java
$ java runGeneticDecipher <args>
```

Where `<args>` are eleven arguments:

```
<arg1> = population size [0..n]
-- n is any maximum
<arg2> = number of generations [0..n]
<arg3> = chromosome size [1..n]
<arg4> = number of elites [2..m]
-- m is equal to <arg1>/2
<arg5> = tournament size [2..5]
<arg6> = crossover type [1..4]
-- 1P, 2P, Uniform, Random respectively
<arg7> = mutation type [1, 2]
-- Uniform and Non-Uniform respectively
<arg8> = crossover rate [0.00..1.00]
<arg9> = mutation rate [0.00..1.00]
<arg10> = random generator seed
<arg11> = filename of encrypted text
```

Included in the working directory are four text files containing samples of encrypted text. Their filenames and correlated chromosome sizes are below:

```
enc1.txt, chromosome size = 8
enc2.txt, chromosome size = 8
enc3.txt, chromosome size = 26
enc4.txt, chromosome size = 40
```

After a failed execution, these instructions are reminded to the user through the Linux console.

For each experiment, parameters used are as follows:

TABLE I. Default Parameters

Parameter	*Value(s)*
Number of Generations	100
Random Seeds	1, 2, 3, 4, 5

Whereas the other parameters are variable as the efficacy of these parameter changes are directly being examined. For each parameter which is variable, these choices were used:

TABLE II. Tested Parameters

Parameter	*Value(s)*
Crossover Rate	90%, 100%
Mutation Rate	0%, 5%, 10%
Crossover Type	1-Point, 2-Point, Uniform, Random
Mutation Type	Uniform, Non-Uniform
Population Size	50, 500, 5000, 50000
Proportion of Elites (%)	1, 5, 10, 20, 30, 40, 50

However, where any parameter is being tested, the others remain constant:

TABLE III. Variable Parameters

Test	Parameter						
	Pop	*Gen*	*CRate*	*MRate*	*CType*	*MType*	*Elites*
CRate	5000	100	Var.	Var.	Rand	Unif	5%
MRate	5000	100	Var.	Var.	Rand	Unif	5%
CType	5000	100	90%	5%	Var.	Unif	1%
MType	500	100	90%	5%	Rand	Var.	20%
Pop	Var.	100	90%	5%	Rand	Unif	10%
Elites	500	100	90%	10%	Rand	Unif	Var.

The first two tests consist of multiple variations:

TABLE IV. Crossover and Mutation Rate Tests

Test	Parameter	
	Crossover Rate	*Mutation Rate*
1	90%	0%
2	90%	10%
3	100%	0%
4	100%	10%
5	90%	5%

IV. Results

The console will output the parameters used and per each generation, the best chromosome and its fitness as well as the population average fitness.

This section serves as results following data collection and testing performed and summarizes the results for each test.

A. Crossover and Mutation Rates

As previously mentioned, the crossover rate dictates the chance that a child chromosome consists of portions of both parent chromosomes. The mutation rate determines how frequently mutation occurs in a new chromosome.

FIGURE 7.

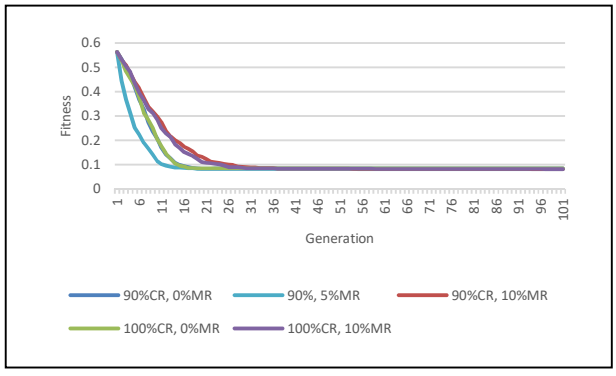

Fig. 7. Crossover and Mutation Rates Test Results (enc3.txt, best)

Using various mutation and crossover rates using `enc3.txt` as encrypted text input parameter provides a visualization of how differing rates for each alters the best fitness per generation. The data suggests that a higher mutation rate (10%) leads to a later convergence but may not converge onto the closest solution. However, mutation is still

an important parameter to consider as a small mutation rate (5%) outperformed no mutation at all.

This is perhaps due to less diversity within the mating pool for each generation: no mutation means fewer novel genes are introduced into the population and instead chromosomes are strictly a function of high fitness elites: were many or all of the elites consisting of incorrect genes yet were able to reproduce, without mutation this means an incorrect solution (despite the high fitness) is transferred to each successive generation.

FIGURE 8.

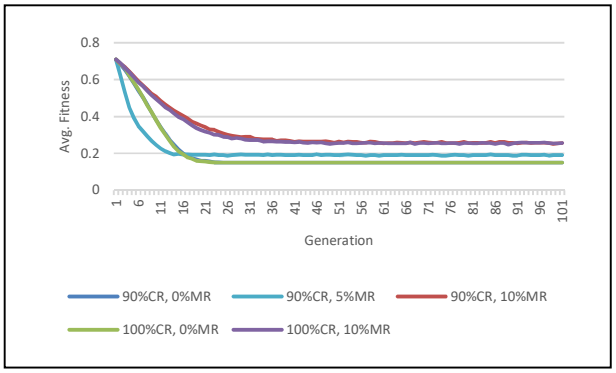

Fig. 8. Crossover and Mutation Rates Test Results (enc3.txt, avg)

Considering the average population fitness, this suggestion is further evidenced. The average population fitness converges to a solution relatively early and while it is not the solution closest to actual, the average population fitness signifies how quickly the best chromosome converges to a solution.

Also suggested is that a higher crossover rate leads to earlier convergence, irrespective of mutation parameter used. The solution converged to is not necessarily the best one for reasons stated previously.

TABLE V. STATISTICAL ANALYSIS USING ENC3.TXT

Statistic	Test				
	90%CR, 0%MR	***90%CR, 5%MR***	***90%CR, 10%MR***	***100%CR, 0%MR***	***100%CR, 10%MR***
Mean (Best)	0.12	0.10	0.13	0.12	0.13
Mean (Avg)	0.20	0.21	0.32	0.20	0.31
Median (Best)	0.08	0.08	0.08	0.08	0.08
Median (Avg)	0.15	0.19	0.26	0.15	0.26
S. Dev (Best)	0.10	0.07	0.11	0.10	0.11
S. Dev (Avg)	0.13	0.08	0.11	0.13	0.11
Max (Best)	0.56	0.56	0.56	0.56	0.56
Max (Avg)	0.71	0.71	0.71	0.71	0.71
Min (Best)	0.08	0.08	0.08	0.08	0.08
Min (Avg)	0.15	0.19	0.25	0.15	0.25

Two statistics are useful in describing the efficacy of parameter choices: the mean best fitness and the standard deviation of average fitness. A smaller mean best fitness denotes an earlier convergence (thus requiring less generations to reach a solution) and a smaller standard deviation of average fitness suggests the overall population is similar (less diverse). Both together suggests a population will quickly approximate a solution and tends to a more fit solution over the entire population, not just the top chromosomes.

To verify prior claims, the same test is performed using `enc4.txt` instead:

FIGURE 9.

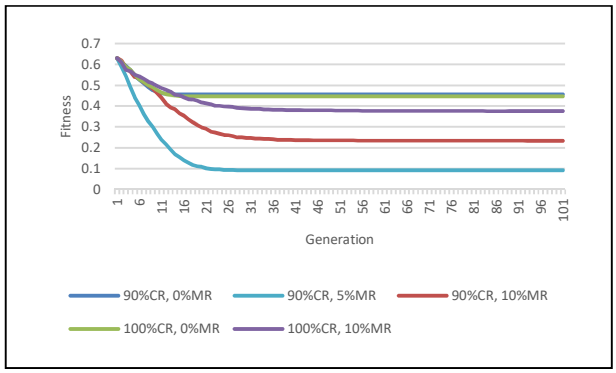

Fig. 9. Crossover and Mutation Rates Test Results (enc4.txt, best)

As before, the minute amount of mutation provides fruitful in finding a solution. However, it is more apparent in these tests. No mutation converges to an incorrect solution early, whereas too much mutation—while providing a better solution—is still far from ideal. This is irrespective of crossover rate.

A small amount of mutation promotes diversity without becoming too diverse. Too much diversity will eventually lead to early convergence as there is too much variation in the population to maintain correct genes within a given chromosome in the population (outside of the elites).

FIGURE 10.

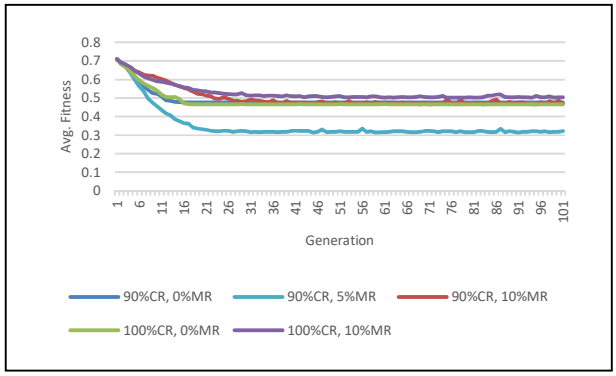

Fig. 10. Crossover and Mutation Rates Test Results (enc4.txt, avg)

Confirming this is the average chromosome fitness within the population converges too early with no or too much mutation, when only a small amount is necessary to provide a dramatically better result.

TABLE VI. STATISTICAL ANALYSIS USING ENC4.TXT

Statistic	Test				
	90%CR, 0%MR	*90%CR, 5%MR*	*90%CR, 10%MR*	*100%CR, 0%MR*	*100%CR, 10%MR*
Mean (Best)	0.46	0.13	0.28	0.46	0.40
Mean (Avg)	0.49	0.35	0.50	0.48	0.53
Median (Best)	0.46	0.09	0.23	0.45	0.38
Median (Avg)	0.48	0.32	0.48	0.47	0.51
S. Dev (Best)	0.03	0.11	0.10	0.03	0.06
S. Dev (Avg)	0.04	0.08	0.06	0.05	0.04
Max (Best)	0.63	0.63	0.63	0.63	0.63
Max (Avg)	0.71	0.71	0.71	0.71	0.71
Min (Best)	0.46	0.09	0.23	0.45	0.38
Min (Avg)	0.48	0.31	0.47	0.47	0.50

As before, a low mean best fitness and a low standard deviation of average fitness means convergence happens later and the average population fitness more resembles the actual solution (and in fact, closer resembles the best fitness).

However, some combinations of crossover and mutation rates yielded differing results with `enc4.txt` versus `enc3.txt`. The chromosome length is likely the impetus to inconsistent results between parameters, yet similar statistics yielding betting solutions is consistent.

While these statistics are not directly controlled and instead determined by the parameters, careful parameter selection should be employed through experimentation for other examples. With both of these examples, however, a 90% crossover and 5% mutation rates is appropriate.

B. *Crossover Type*

Another parameter which has some effect on the genetic algorithm performance is the crossover operator used. As mentioned previously, implemented operators are 1-Point, 2-Point, Uniform, and a randomized operator which selects one of the prior three at random.

By constraining other parameters and only varying the crossover operator used, their effect on performance can be examined. `enc4.txt` is chosen as the chromosome is longer, resulting in additional granularity between generations. The generations in the below is reduced to show better distinction.

FIGURE 11.

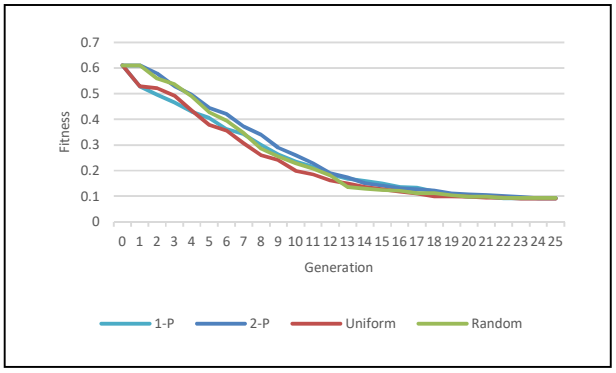

Fig. 11. Crossover Type Test Results (enc4.txt, best)

As found in the prior figure, there is little difference in operator used. Minute changes in timing of convergence or early fitness aside, every crossover operator converges to the same solution. Thus, it is appropriate to use any operator (or, randomized, as implemented) as the effect on final fitness is negligible if non-existent. Convergence may occur early but with enough generations, the result is the same.

C. *Mutation Type*

Two mutation operators are present with one of them consisting of two variations. By keeping other parameters consistent, all three can be tested.

FIGURE 12.

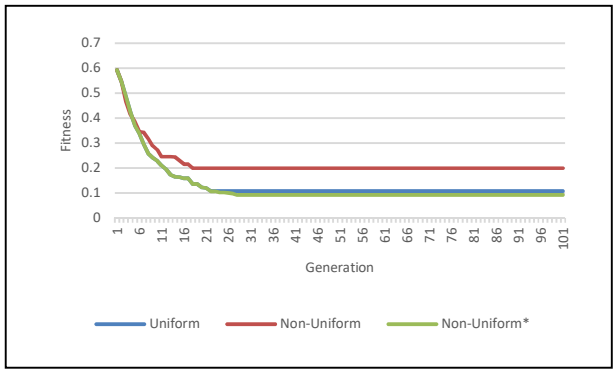

Fig. 12. Mutation Type Test Results (enc3.txt, best)

Uniform mutation means the mutation rate does not change with time (generations), whereas non-uniform increases the mutation rate with time. The third operator is non-uniform mutation, but with a later activation time: instead of increasing mutation rate between generations `0..maxGen`, the mutation rate increases between `(maxGen/3)..maxGen`.

A uniform mutation rate tends to a solution quicker than non-uniform, however reintroducing diversity into the population by "activating" non-uniform mutation later than at program execution yields a slightly better result.

A lower mutation rate initially and mutation added later in the process of finding a solution may introduce novel genes to the population, genes that may otherwise be unavailable depending on elite selection and crossover parameters used. This is proven by analysis of the population average fitness with the varying mutation operators.

FIGURE 13.

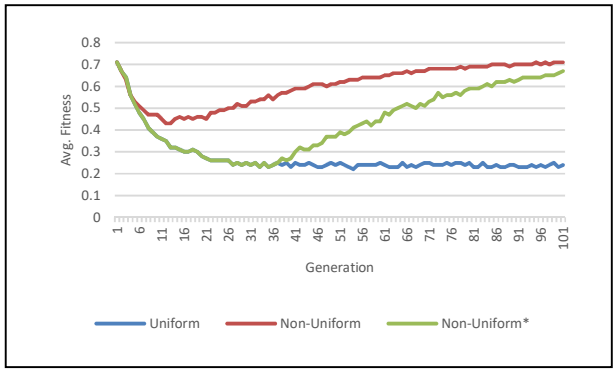

Fig. 13. Mutation Type Test Results (enc3.txt, avg)

D. Population Size

Size of the population can also have an impact on performance, specifically whether a correct solution is converged upon or the population is stuck in a local extremum.

FIGURE 14.

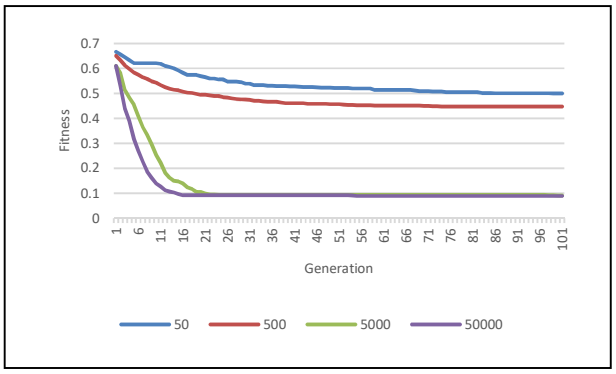

Fig. 14. Population Size Test Results (enc4.txt, avg)

Using `enc4.txt` as an example, a population size of 50 and 500 were inadequate parameters as there is stagnation in early generations leading to early convergence to a (wrong) solution. Contrastingly, higher populations show a dramatic change in fitness.

Despite each iteration increasing the population by a factor of ten, the fitness does not follow this trend and instead there is a dramatic change in fitness between 500 and 5,000, with smaller changes between 50 and 500, and 5,000 and 50,000.

This suggests there is both a minimum and a maximum boundary which the population size parameter should be chosen within: too high of a population may make no difference in final fitness, whereas too small of a population may result in early convergence to an incorrect solution.

E. Proportion of Elites

Lastly, the number of elites as a parameter has a large effect on the success of the algorithm. As the proportion of elites to the overall population relates to the overall diversity of the population, it may influence the number of novel genes per generation, not unlike altering the mutation rate.

FIGURE 15.

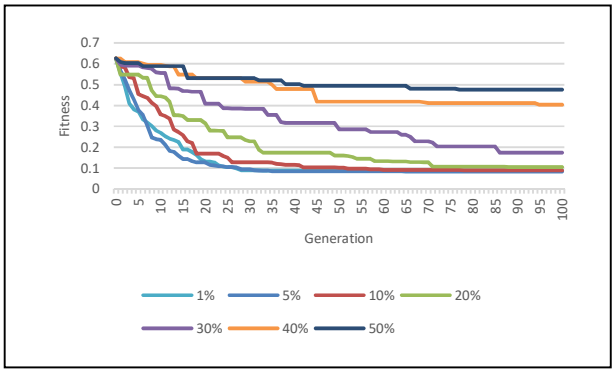

Fig. 15. Proportion of Elites Test Results (enc3.txt, avg)

Too many elites in the population leads to not enough diversity and recombination of parent chromosomes in child chromosomes. Effectively, too much of the population is identical leading to stagnation.

However, less elites promotes the most diversity in the population and ensures very few chromosomes are identical: while no elites would prove to have a fully unique population, this can lead to regression (fitness reverting to worse over time rather than improve or remain with each generation).

V. Discussion & Conclusion

Among the parameters tested, some had minute but non-negligible effect to the genetic algorithm efficacy and some had larger effects.

By comparing different crossover and mutation rates, it was found that a high crossover rate lead to better solution and using less generations to reach convergence, whereas a low but non-zero mutation rate gave best fitness as diverse genes become present in a population that would otherwise stagnate.

Between crossover and mutation operators used, the crossover type made negligible difference to efficacy of the algorithm while uniform mutation lead to the most consistent results. A delayed non-uniform mutation proved somewhat fruitful in reintroducing diversity at a later generation but the performance difference versus a uniform mutation rate was negligible.

The final two tests, testing the size of the population and the proportion of the population designated elite had a large impact on performance.

A lower and upper bound on valid population choices varies between chromosome size and encrypted text used and is a large deciding factor in efficiency. Unfortunately, this is situation specific and no "one size fits all" approach applies, although higher is typically better at the risk of diminishing returns. Unequivocally, however, the proportion of elites in the population should remain low to achieve best fitness.

Concerning both test subjects `enc3.txt` and `enc4.txt`, the below parameters are found to be ideal:

TABLE VII. IDEAL PARAMETERS

Parameter	***Value***
Population Size	5000
Number of Generations	100
Elite Proportion	1%
Crossover Rate	90%
Crossover Type	Negligible difference
Mutation Rate	5%
Mutation Type	Uniform

The above parameters provided the best scenario for both timing of convergence and accuracy of solution.

References

[1] Fogel, David. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 3rd ed., Piscataway, NJ: IEEE Press.

[2] Holland, John. *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press.

[3] Kramer, Oliver. *Self-Adaptive Heuristics for Evolutionary Computation*. Dortmung, Germany: Springer.

[4] Schmitt, Lothar. *"Theory of Genetic Algorithms". Theoretical Computer Science*. 259 (1-2): 1-61.

[5] Spears, William. *A Study of Crossover Operators in Genetic Programming*. Washington, DC: US Naval Research Center