Evolutionary Computing - Assignment 1

Ratih Ngestrini (6047939) and Steven Both (4138015)

March 10, 2017

1 Introduction

Genetic algorithms contains many variables such as population size, number of generation, selection, recombination, and mutation operators that are called as GA parameters. These parameters are explored and set up so that it converges efficiently to the best solution among many possible solutions.

The main purposes of this report is find out how applying different parameters for population sizes, crossover types and fitness function will affect the outcome of a Genetic Algorithm. To do this, we will divide the problem into multiple smaller research questions:

- What population size results in the best outcome?
- What crossover type results in the best outcome?
- What type of fitness fuction results in the best outcome:
  - Counting Ones Function or Trap Function?
  - Uniformly Scaled or Linearly Scaled?
  - Deceptive or Non-deceptive?
  - Randomly linked or Tightly linked?

2 Methodology

A program is built using C# to obtain the experimental results of Genetic Algorithms on several fitness functions with the same selection operator ($N + N$-selection) and string length ($\ell = 100$). The parameters to be varied for each experiment are type of fitness function, population size (N), and crossover operator (uniform and 2-point crossover). The program performed 25 independent runs to get results statistics for each parameter setting (see Appendix).

The steps of Genetic Algorithms are:

1. Initialization

   At first, initial population with N individuals will be created and serves as first generation. Individual is represented in random binary strings of 0's and 1's with length ℓ. Then each individual in the population is evaluated by calculating the fitness value according to the type of function.

2. Recombination (crossover)

   Crossover operator is applied on population of each generation (called parent population) to produce offspring population. There are two types of crossover operators: uniform crossover (UX) and 2-point crossover ($2X$). Recombination begins by taking two adjacent individuals from parent population and applying the crossover operator to create two offspring. The crossover process can be described as follows:

- *UX*: A Boolean variable is created to generate random value, true of false, with probability 0.5. Each pair of individual bits from two parents is evaluated and they will be swapped if the variable value is true.
- *2X*: Two crossover points are created by generating two random integer between 1 - 100. The first crossover point is lowest one, while another will be the second crossover point. The binary string from beginning of individual to the first crossover point is then copied from first parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent.

Offspring population with same size as parent population will then be produced and fitness value of each offspring is also calculated.

3. Creation of next generation population (selection)

   In this step, the parent and offspring population ($N+N$) are combined and sorted based on fitness value of each individual, then the best N individuals are selected and used as next generation population. After that, this new population is shuffled randomly to be prepared for recombination process in next iteration.

4. Stop condition

   Recombination and selection process will run repeatedly, and continue to produce new candidate solution which is the fittest individual from each generation. In each iteration, fitness value of individuals are evaluated. Genetic algorithm stops if the highest fitness value of the offspring population is less than or equal to the lowest fitness value of the parent population (previous generation) because it means that next iteration processes will not produce better individual than the parent population and tend to converge to same solution.

Test functions used in experiment to calculate fitness value of each individual in population are Counting Ones Functions (Uniformly Scaled and Linearly Scaled Counting Ones Function) and Trap Functions (tightly and randomly linked Deceptive and Non-deceptive Trap Functions). For each parameter setting, program then calculate statistics of 25 runs to be observed in the next section which includes:

- Success is the number of successful run achieving global optimum
- Gen (First Hit) is the generation where the global optimum enters the population for the first time
- Gen (Converge) is the generation when the stopping condition described above is triggered
- Fct Evals is the number of function evaluations during a run from the first generation (initial population) until Gen (Converge) generation
- CPU time is the running time corresponding with Gen (Converge) generations

3 Results and observations

The results of our experiment are presented in the tables in the Appendix. Each table describes the behavior of our Genetic Algorithms convergence by applying a particular fitness function with different parameters for population size and crossover type. We will discuss the results of each of these parameters in the following sections.

3.1 Population size

In the results there is a clear difference between the different population sizes. This is clearly visible for the Trap Functions, but somewhat less visible for the Counting Ones Functions, so we will discuss them separately.

First the Counting Ones Functions. The outcomes of the four population sizes is not really different for *UX*. In almost every experiment the global optimum was found, except for the combination of a population of 50 and the Linearly Scaled. For these parameters less than halve of the experiments resulted in success. Apparently this small population size is not enough for the algorithm to converge to the global optimum every time. After the population size of 250 is reached, the success rate does not change anymore and rests at 100% for the larger population sizes.

For *2X* the population sizes of 50, 100 and 250 were to small for the algorithm to find the global optimum. Only for a population of 500 the algorithm was able to find the optimum at least once. Here, the Uniformly Scaled Function (8 out of 25) performed better than the Linearly Scaled Function (1 out of 25).

For the Trap Functions the difference was very clear. The population sizes of 50 and 100 resulted in none of our experiments in the global optimum, for both *2X* and *UX*. Only a population size of 500 for the two Non-deceptive Trap Functions was enough to succeed almost all experiments, but this only applies to *UX*.

We also conducted tests for significance, using ANOVA (F test) with significance level 5%, to compare the mean of the convergence generation for different population sizes. The results show that there is a statistically significant difference in the mean of convergence generation for all of fitness functions and crossover types, except for the Uniformly Scaled Counting Ones Function applying *UX*.

3.2 Crossover type

For almost all population sizes and fitness functions, *UX* succeeds more often than or equally often as *2X*. The only exception being a population of 500 for the tightly linked Deceptive Trap Function. Against our expectations, *2X* was able to find the global optimum twice out of 25 experiments.

That *UX* outperforms *2X* is very obvious. For the Counting Ones Functions *UX* finds the global optimum almost every experiment, while *2X* finds it only a couple of time, only under the best of circumstances. This is equally true for the Trap Functions, but the difference is less outspoken, because *UX* also performs worse here.

In *2X*, convergence occurs too early before the population reaches the global optimum. In terms of convergence performance, *UX* executes more number of function evaluation since it creates more generations than *2X*, hence the running time will take longer than *2X*.

3.3 Fitness function

We will discuss the differences in the results of the various fitness function in the same division as used for the research questions.

3.3.1 Counting Ones Function or Trap Function

It is clearly shown in the tables that Counting Ones Function has higher success rate compared to Trap Function. For the performance Counting Ones also is faster than Trap Function.

3.3.2 Uniformly Scaled or Linearly Scaled

Overall uniformly Scaled and Linearly Scaled have almost same success rate and performance, but if we see carefullly with population size 100, all experiments in Uniformly Scaled have found the global optimum. For the performance Uniformly Scaled runs faster than Linearly Scaled with the same parameter settings.

3.3.3 Deceptive or Non-deceptive

Non-deceptive achieves more success rate compared to Deceptive especially with *UX* and higher population size (250 dan 500) and there is no big difference in performance between two functions both in number of function evaluation and CPU time.

3.3.4 Randomly linked or Tightly linked

The difference between these two link types is not very clear. *2X* performed slightly better on the tightly linked than on the randomly linked Trap Functions, mainly because for the randomly linked variant it never reached the global optimum, while it succeeded a total of 7 times using a tightly linked function. For *UX* there is no big difference. It succeeded 33 out of 200 times for randomly linked and 31 out of 200 times for tightly linked. All successes were produced with the two highest population sizes.

Also there is not much difference between the convergence generations or the number of function evaluations. In CPU time however, there is a difference, with randomly linked taking more time to converge. Probably this is because of the extra indexing time as a result of the randomly linking.

4 Conclusion

For the experiments that we performed we can conclude that a larger population size (in our case 500) will always result in a better outcome; that is a higher success rate. This is especially true for the Trap Functions. For the Counting Ones Functions a nuance can be placed, namely that a population size of 250 is already large enough to reach the same, best outcome. The only difference is that a smaller population size requires less computing time and less function evaluations and is therefore preferred over a larger population size with the same outcome.

Using our data, it is fairly easy to decide which crossover type results in the best outcome. Because almost everywhere *UX* resulted in a better outcome than *2X*, it is safe to conclude that *UX* is by far superior.

The distinctions we made for the fitness functions performed not all very different for this problem. The Counting Ones Functions performed better than the Trap Functions, especially under the less optimal parameter settings. But this makes sense, because this problem of a optimizing a binary string is a fairly easy problem. The simple Counting Ones Functions are pretty much created to solve problems like this. The more complex Trap Functions probably are just to complicated for such a simple problem.

For the two variants of the Counting Ones Function, namely the Uniformly Scaled and the Linearly Scaled, Uniformly Scaled resulted in a better outcome. The weight that is added in the Linearly Scaled Function just doesn't pay off, because in the algorithm a string with only one 1 with a very large weight can have a higher fitness than a string with several light ones. In the selection process a string that has a higher fitness (but might lie further away from the global optimum) will preferred over the other. So the data coincides with the expected result.

The fitness functions where the difference was the clearest, were the Deceptive and the Non-deceptive Trap Functions. Except for one occasion, the Deceptive Functions never resulted in the global optimum, while the Non-deceptive Functions succeeded multiple times. So the Non-deceptive Functions are very much preferred over the Deceptive ones. This is not surprisingly, because these Non-deceptive Functions were designed to counter the mistake that Deceptive functions are bound to make. That is, that on the level of lower-order schema fitness averages, it prefers a string like "0000" over "1111", which is the oposite of what we are trying to achieve.

Finaly, the difference between a randomly linked Trap Function and tightly linked ones. Against our own expectations, there was not really a distinct difference between these two types of fitness functions. In the case of the outcome, they were in fact rather similar. The only difference being that randomly linked functions tend to cost more computing time, making the tightly linked functions somewhat faster. So, concidering this, a thightly linked Trap Function has slightly the upper hand.

So, in short, for this problem we dealt with, our Genetic Algorithm results in the best outcome for a combination of a Uniformly Scaled Counting Ones Function, *UX* and a population size of 250.

5 Appendix

Uniformly Scaled Counting Ones Function						
Crossover	PopSize	Successes	Gen. (First Hit)	Gen. (Converge)	Fct Evals	CPU time (ms)
2X	50	0/25	NA (NA)	10.80 (2.14)	591 (107.04)	12.96 (4.32)
	100	0/25	NA (NA)	13.72 (3.26)	1473 (325.99)	33.60 (8.88)
	250	0/25	NA (NA)	30.56 (5.55)	7891 (1388.34)	172.24 (35.36)
	500	8/25	37.5 (4.31)	42.20 (4.03)	21601 (2015.56)	442.96 (41.12)
UX	50	24/25	26.42 (1.38)	31.48 (0.96)	1625 (48.13)	35.52 (2.43)
	100	25/25	24.40 (1.5)	30.28 (0.74)	3129 (73.71)	70.20 (3.45)
	250	25/25	23.68 (0.95)	29.88 (0.44)	7721 (109.92)	185.28 (4.45)
	500	25/25	22.64 (0.64)	29.44 (0.51)	15221 (253.31)	333.56 (9.29)

Table 1: Uniformly Scaled Counting Ones Function

Linearly Scaled Counting Ones Function						
Crossover	PopSize	Successes	Gen. (First Hit)	Gen. (Converge)	Fct Evals	CPU time (ms)
2X	50	0/25	NA (NA)	11.88 (1.64)	645 (82.06)	17.52 (12.97)
	100	0/25	NA (NA)	13.8 (2.35)	1481 (234.52)	34.68 (8.67)
	250	0/25	NA (NA)	32.72 (4.43)	8431 (1107.55)	179.88 (29.93)
	500	1/25	40 (0)	47.64 (3.96)	24321 (1978.43)	491.32 (41.87)
UX	50	11/25	34.27 (3.07)	39.24 (2.55)	2013 (127.70)	43.76 (4.88)
	100	24/25	31.58 (1.67)	37.92 (1.61)	3893 (160.52)	86.32 (6.26)
	250	25/25	29.28 (1.46)	36.96 (0.84)	9491 (210.16)	218.72 (19.95)
	500	25/25	28.04 (1.37)	36.72 (0.68)	18861 (339.12)	415.92 (10.57)

Table 2: Linearly Scaled Counting Ones Function

Deceptive Trap Function (tightly linked)						
Crossover	PopSize	Successes	Gen. (First Hit)	Gen. (Converge)	Fct Evals	CPU time (ms)
2X	50	0/25	NA (NA)	11.6 (2.02)	631 (101.04)	15.84 (3.66)
	100	0/25	NA (NA)	16.64 (3.05)	1765 (305.34)	45.52 (11.61)
	250	0/25	NA (NA)	33 (5.42)	8501 (1354.01)	202 (36.39)
	500	2/25	40.5 (0.71)	47.68 (5.43)	24341 (2714.62)	584.64 (71.39)
UX	50	0/25	NA (NA)	46.28 (6.01)	2365 (300.53)	57.60 (8.56)
	100	0/25	NA (NA)	51.6 (4.44)	5261 (444.41)	129.72 (11.48)
	250	0/25	NA (NA)	59.24 (9.27)	15061 (2318.72)	372.08 (56.95)
	500	0/25	NA (NA)	64.28 (11.66)	32641 (5830.09)	795.24 (142.31)

Table 3: Deceptive Trap Function (tightly linked)

Non-deceptive Trap Function (tightly linked)						
Crossover	PopSize	Successes	Gen. (First Hit)	Gen. (Converge)	Fct Evals	CPU time (ms)
2X	50	0/25	NA (NA)	11.28 (2.57)	615 (128.71)	19.28 (9.44)
	100	0/25	NA (NA)	18.04 (4.73)	1905 (473.00)	58.20 (24.61)
	250	0/25	NA (NA)	34.88 (3.84)	8971 (961.01)	229.16 (35.82)
	500	5/25	35.20 (6.06)	41.48 (3.74)	21241 (1871.50)	492.68 (44.38)
UX	50	0/25	NA (NA)	53.16 (5.89)	2709 (294.28)	67.76 (8.17)
	100	0/25	NA (NA)	60.72 (6.43)	6173 (643.25)	155.44 (15.17)
	250	7/25	57.14 (12.9)	66.52 (8.53)	16881 (2132.49)	426.96 (49.6)
	500	24/25	50.33 (5.89)	64.24 (7.95)	32621 (3974.61)	816.36 (98.96)

Table 4: Non-deceptive Trap Function (tightly linked)

Deceptive Trap Function (randomly linked)						
Crossover	PopSize	Successes	Gen. (First Hit)	Gen. (Converge)	Fct Evals	CPU time (ms)
2X	50	0/25	NA (NA)	14.92 (3.43)	797 (171.34)	27.40 (10.02)
	100	0/25	NA (NA)	22.64 (5.71)	2365 (570.73)	80.60 (20.23)
	250	0/25	NA (NA)	43.88 (8.67)	11221 (2167.85)	358.68 (98.73)
	500	0/25	NA (NA)	57.44 (5.12)	29221 (2562.06)	894.24 (75.82)
UX	50	0/25	NA (NA)	48.92 (6.73)	2497 (336.32)	79.24 (9.71)
	100	0/25	NA (NA)	52.88 (5.49)	5389 (549.48)	170.92 (17.46)
	250	0/25	NA (NA)	60.08 (9.92)	15271 (2480.05)	489.76 (77.63)
	500	0/25	NA (NA)	65.88 (10.77)	33441 (5385.78)	1069.60 (176.44)

Table 5: Deceptive Trap Function (randomly linked)

Non-deceptive Trap Function (randomly linked)						
Crossover	PopSize	Successes	Gen. (First Hit)	Gen. (Converge)	Fct Evals	CPU time (ms)
2X	50	0/25	NA (NA)	15.08 (3.45)	805 (172.55)	28.32 (9.38)
	100	0/25	NA (NA)	22.64 (5.88)	2365 (587.99)	82.12 (22.58)
	250	0/25	NA (NA)	45.80 (7.89)	11701 (1972.47)	397.92 (88.61)
	500	0/25	NA (NA)	65.24 (7.24)	33121 (3617.90)	1017.44 (105.01)
UX	50	0/25	NA (NA)	52.48 (5.19)	2675 (259.45)	85.00 (9.02)
	100	0/25	NA (NA)	64.24 (10.60)	6525 (1059.59)	208.36 (34.39)
	250	10/25	55.30 (5.91)	66.64 (8.91)	16911 (2227.76)	552.48 (78.37)
	500	23/25	52.48 (6.79)	64.80 (9.24)	32901 (4618.8)	1079.72 (153.38)

Table 6: Non-deceptive Trap Function (randomly linked)

Summary of ANOVA (F test)

H_1: There is significant difference on the mean of generation converge, at least one of the mean is different

Significance level $\alpha = 0.05$, $F_{crit} = 2.6148$

Crossover	Function	F	P-value
UX	Uniformly Scaled Counting Ones Function	1.59841905455302	0.188223693572215
	Linearly Scaled Counting Ones Function	23.9990645645353	0.000000000000006
	Deceptive Trap Function (tightly linked)	7.82698106237096	0.000036837434692
	Non-deceptive Trap Function (tightly linked)	17.4509324825173	0.000000000051163
	Deceptive Trap Function (randomly linked)	8.40759432264796	0.000016294793790
	Non-deceptive Trap Function (randomly linked)	13.0571679836139	0.000000023706457
2X	Uniformly Scaled Counting Ones Function	14.0032363811219	0.000000006296541
	Linearly Scaled Counting Ones Function	23.9732252416208	0.000000000000006
	Deceptive Trap Function (tightly linked)	13.1835395321239	0.000000019856924
	Non-deceptive Trap Function (tightly linked)	12.6561013787089	0.000000041608930
	Deceptive Trap Function (randomly linked)	9.05622667132297	0.000006547507045
	Non-deceptive Trap Function (randomly linked)	11.0573676391559	0.000000392846580

Table 7: Outcome of our ANOVA (F test)