Homework Two Prime

Iacobescu Tudor, A6

3 dec 2019

1 Introduction

This document follows the improvement of the algorithm from T2 with a few enhancements specific to genetic algorithms. The following will explain these enhancements, how they were implemented, and how the results compare between runs with and without them.

2 Best result from all generations

The first change is the simplest one: previously, after the generational cycle ended, the result of the algorithm was chosen as the best individual of the remaining generation. This has been modified, so that the result is chosen as the best individual found in any generation. This serves to lessen the impact of the best individuals never reaching the last generation due to bad luck.

3 Diversity multiplier for mutation

Another addition was that of a "multiplier" for the mutation chance, which increases if the diversity of a generation is too low. The mutation function was modified as such:

```
for bit in individual:
    if random(0, 1) < MUT_CHANCE * mult / individual.size()
        bit = !bit
```

The multiplier is actually set in the selection function, by comparing the average result of an individual with the best one in the population.

```
evalAvg = evalSum / POP_SIZE
if (evalAvg - bestCurrentEval <= DIVERSITY_TRESHHOLD):
    multiplier += MULTIPLIER_INCREMENT
else:
    multiplier = 1.0
```

The diversity treshhold was 0.1, and the multiplier increment, 0.5. As such, for example, if the best individual in a population isn't over 10% better than the average for two generations, it becomes twice as likely for the mutation function to alter a bit. The multiplier is reset when the population becomes diverse enough again.

4 Gray encoded individuals

This is the change that, as will be shown, proved less effective than initially hoped. The idea is that it is ideal for the mutation function to favor mutations that produce a smaller difference in the result. Mutating a bitstring that's encoded in Gray code should lead to less of a change than mutating a normal bitstring, as adjacent values only differ by one bit in Gray code.

In practice, the only change is that, before being converted to numbers which can be evaluated by the function, bitstrings are first decoded from Gray code. For example, instead of:

```
eval = func(bitsToNum(individual))
```

We would have:

```
eval = func(bitsToNum(grayDecode(individual)))
```

The "grayDecode" function is fairly simple as well.

```
grayDecode(gray):
    bits = []
    bits[0] = gray[0]
    for i in range(1, grey.size()):
        if gray[i]:
            bits[i] = !bits[i-1]
        else:
            bits[i] = bits[i-1]
```

5 Results

Two tests were ran - one with the first two changes, and another with the third as well. Here is how they compared:

function	d	w/o gray encoding		with all changes		without changes		min(f)
		rMean	tMean	rMean	tMean	rMean	tMean	
Dixon & Price	2	0.02	49417.40	0.00	57809.87	0.74	49621.27	0
	5	1.27	87824.83	1.48	106361.97	13.10	89026.77	0
	30	3323.21	419335.50	16094.16	517335.67	4944.14	443351.67	0
Michalewicz	2	-1.80	45518.97	-1.80	53814.23	-1.78	41696.53	-1.8013
	5	-4.45	79538.43	-4.41	95255.93	-4.39	82597.03	-4.687658
	30	-23.44	391488.97	-13.66	497810.60	-23.64	402037.57	apx. -30
Rastrigin	2	0.37	37302.67	0.47	55627.87	1.61	48127.53	0
	5	6.85	66938.70	6.50	102987.40	8.10	82636.00	0
	30	78.26	335209.63	80.95	486807.27	81.56	389563.27	0
Sphere	2	0.00	47741.73	0.00	55770.53	0.01	49067.73	0
	5	0.00	83875.00	0.01	101609.63	0.04	85068.53	0
	30	1.71	396491.77	4.36	482097.93	1.82	395912.27	0

Table 1: Results table - the two modified versions, compared to the original.

The first two changes proved fruitful - the version utilizing them yielded generally improved results with only a negligible performance impact.

Adding Gray encoding wasn't as succesful. The quality of results is mixed - in many cases worse - and the time impact is significant. It is, however, possible that this would not be the case with different functions, and the performance hit could be mitigated by a more efficient implementation.

6 Conclusion

Taking the best result from all generations and implementing a super-mutation method both prove to useful in obtaining a good result. Grey encoding, on the other hand, is hard to get pronounce oneself upon - it's possibly situational, at the very least.