COMP 6660 Fall 2020 Assignment 1C

Jordan Sosnowski
jjs0029@auburn.edu

Table of Contents

Green 3:

For this assignment series, we were asked to implement constraint-satisfaction for our evolutionary algorithm. Specifically, a penalty function for constraints. Instead of immediately invalidating solutions and giving them a fitness of 0 a penalty function would be applied to them. My penalty function was as follows. First, you iterate through a dictionary of black cells that are not satisfied. You then get the difference in what the value of bulbs around the cell should be vs what was and take the absolute value of that. This is added to a penalty sum. After checking unsatisfied black cells, I go on to check the amount of intersecting light rays there were. Each number of intersecting rays are added up to the penalty sum as well. Finally, this penalty sum is multiplied by the penalty coefficient and subtracted from the original fitness.

```
def penalty_function(self, fitness, black_cells, intersections, penalty_coef):
    """ Penalizes current individual based on the amount of intersecting rays
    and black cells that are not constrained.

    f' = f - p*(black_cells + intersections)

    where f is the original fitness
    where f' is the new fitness
    where p is the penalty coefficient
    """

    penalty = 0
    for expected, actual_list in black_cells.items():
        for actual in actual_list:
            penalty += abs(expected - actual)

    for section in intersections.values():
        for intersection in section:
            penalty += intersection

    fitness -= penalty*penalty_coef
    return fitness
```

Using a penalty function was also easy to integrate as I simply inserted it into the fitness function but with a conditional statement based on a user-defined parameter.

```
def calculate_fitness(self, board, penalty_coef):
    """ Calculates fitness of current board.

    board - RxC numpy array representing the game board, filled with bulbs,
    black cells, and light
    """
    fitness = lightup.calculate_completion(board)
    black_cells = lightup.check_black_cells(board, self.ignore_black_cells)
    intersections = lightup.check_intersections(board)
    fitness_prime = fitness

    if self.constraint_algorithm in CON_PENALTY:
        if penalty_coef is None:
            penalty_coef = self.penalty_coefficient
        fitness_prime = self.penalty_function(
            fitness, black_cells, intersections, penalty_coef)

    penalty = fitness - fitness_prime
    return fitness_prime, penalty
```

We are additionally asked to implement a new survival strategy. Thus far we have been using a "plus" survival strategy or a (μ+λ)-EA. Now we were asked to implement a comma or generational EA [(μ,λ)-EA]. For a generational EA for every generation, the existing generation is discarded and only the children live on. Due to the modularity of my code, this was rather easy to implement. After the children are created, I call the below function to determine which individuals will be passed to survival selection.

```
def survival_strategy_selection(self, old_generation, children):
    """ Decide on next generation for survival function.

    old_generation - list of individuals that make up the older generation
    children - list of individuals that make up children for the new generation
    """

    if self.survival_strategy_algorithm in PLUS:
        # (μ + λ)-EA
        return old_generation + children

    if self.survival_strategy_algorithm in COMMA:
        # (μ, λ)-EA
        return children
```

In addition to the survival strategy, we were asked to implement a new parent selection algorithm and two new survival selection algorithms. First, I implemented a uniform random parent selection algorithm which was trivial.

```
def uniform_random_parent(self, individuals):
    """ Selects parents randomly with a uniform distribution

    individuals - list of Individual objects
    """

    mating_pool = random.choices(individuals, k=self.children*2)
    return mating_pool
```

Following this, I implemented the uniform random for survival selection as well. Since for survival, we do not want to sample with the replacement I used `random.sample` as `random.choices` use replacement.

```
def uniform_random_survival(self, individuals):
    """ Selects parents randomly without replacement with a uniform distribution

    individuals - list of Individual objects
    """

    mating_pool = random.sample(individuals, k=self.parents)
    return mating_pool
```

Last was FPS for survival selection. For this I had to use NumPy's `random.choice` as Python's standard library's sample does not allow weight to be passed to it.

```
def fitness_proportional_selection_survival(self, individuals):
    """ Selects parents based on fitness probability without replacement.

    individuals - list of Individual objects
    """

    fitness_sum = sum(individuals)
    if fitness_sum == 0:
        return random.choices(individuals, k=self.children*2)
```

```
        probs = [individual.fitness /
                 fitness_sum for individual in individuals]

        return list(np.random.choice(individuals, p=probs, size=self.children*2, replace=False)
)
```

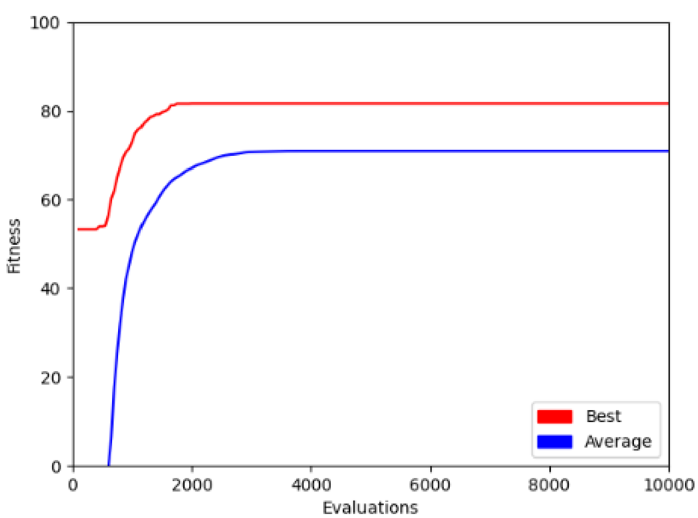
Performance

For the problems for this assignment, I used the same EA parameters for each one. They are listed below:

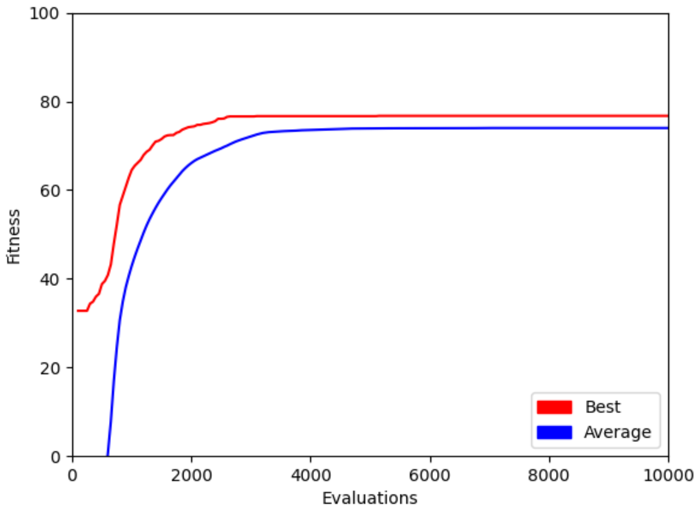
```
{
    "children": 50,
    "parents": 100,
    "parent_alg": "sus",
    "child_alg": "one point crossover",
    "survival_alg": "truncation",
    "termination_alg": "num of evals",
    "num_of_runs": 30,
    "initialization_alg": "vanilla",
    "survival_strategy_alg": "plus",
    "fitness_evals": 10000,
    "mutation_rate": 0.40,
    "penalty_coefficient": 10,
    "constraint": "penalty",
    "ignore_black_cells": false,
    "log_1b": true
}
```

As BC# are easier problems the graphs are what I expected. Since C# are more difficult their poor graphs are also not surprising. I think I could achieve higher performance on those but since that is not the focus on these assignments I did not put to much effort into those portions.

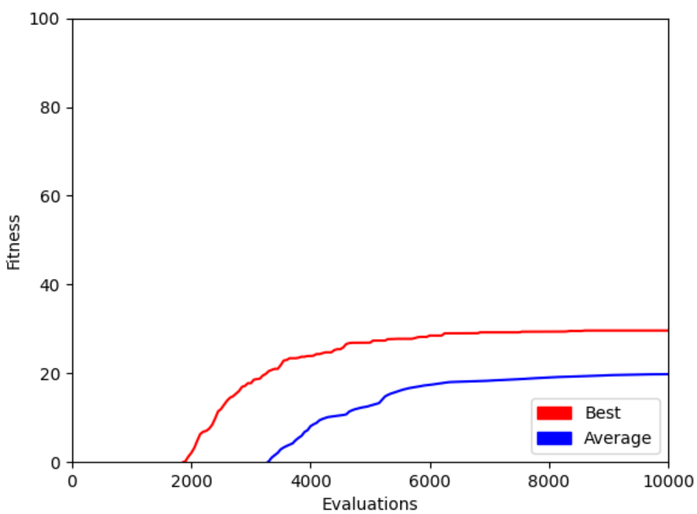
BC1



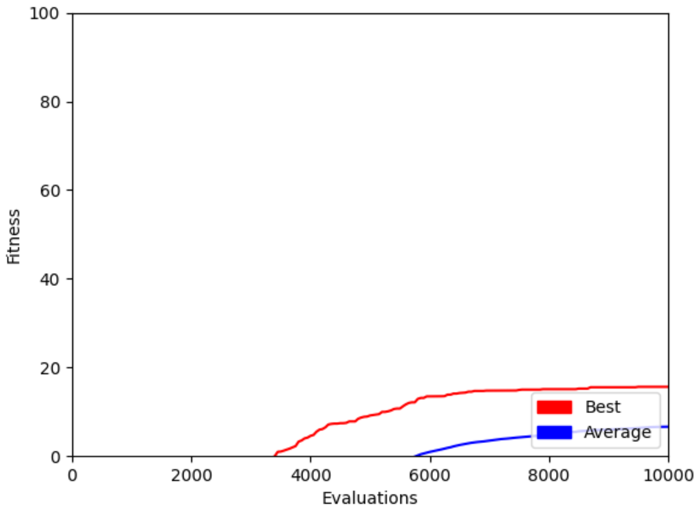
BC2


100
80
60
40
20
0
Fitness
Best
Average
0
2000
4000
6000
8000
10000
Evaluations

C1


100
80
60
40
20
0
Fitness
Best
Average
0
2000
4000
6000
8000
10000
Evaluations

C2


100
80
60
40
20
0
Fitness
0
2000
4000
6000
8000
10000
Evaluations
Best
Average

Green 3.1: 1B Evolutionary Algorithm vs 1C Evolutionary Algorithm

1BC

Here we have a comparison against 1B's EA and 1C's EA. 1B is using a "vanilla" EA and 1C is using a penalty constraint EA. As you can see 1C has higher finesses where 1C mostly has finesses of 0. Unfortunately, this means that most of the solutions in 1C are technically invalid but if this EA could run longer in theory, we would see these penalized solutions go away and be replaced with "valid" ones.

Vanilla EA	Penalty Constraint EA
0	83
100	100
0	73
0	87
93	93
0	90
0	87
0	87
97	97
0	53
0	87
0	60
0	87
0	46
0	90
0	87
0	46
0	83
0	90
93	93
0	47
0	80
97	97
0	87
0	90
0	70
100	100
0	83
0	80
97	97

F-Test Two-Sample for Variances

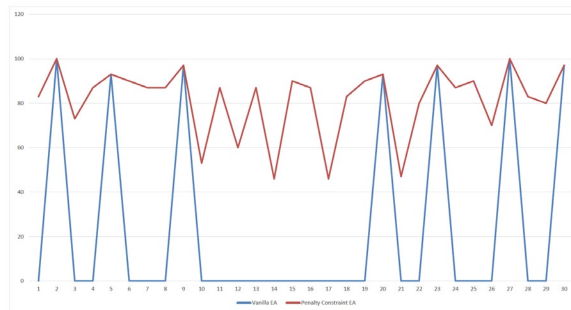
	Variable 1	*Variable 2*
Mean	22.56666667	81.66666667
Variance	1732.667816	254.1609195
Observations	30	30
df	29	29
F	6.817207851	
P(F<=f) one-tail	7.98539E-07	
F Critical one-tail	1.860811435	

F > F-Critical

assume equal

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	22.56666667	81.66666667
Variance	1732.667816	254.1609195
Observations	30	30
Pooled Variance	993.4143678	
Hypothesized Mean Difference	0	
df	58	
t Stat	-7.262194763	
P(T<=t) one-tail	5.31389E-10	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	1.06278E-09	
t Critical two-tail	2.001717484	



2BC

Here we can see the same thing except with the problem 2BC.

Vanilla EA	Penalty Constrain EA
0	87
0	80
0	81
0	69
0	69
0	90
0	90
0	70
0	77
0	81
0	43
0	77
0	64
0	63
0	87
0	47
0	76
0	90
0	39
0	87
0	81
0	90
0	77
0	87
0	87
0	90
97	97
0	76
0	61
89	89

F-Test Two-Sample for Variances

	Variable 1	*Variable 2*
Mean	6.2	76.73333333
Variance	557.8206897	215.3057471
Observations	30	30
df	29	29
F	2.590830468	
P(F<=f) one-tail	0.006280087	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	6.2	76.73333333
Variance	557.8206897	215.3057471
Observations	30	30
Pooled Variance	386.5632184	
Hypothesized Mean Difference	0	
df	58	
t Stat	-13.89407932	
P(T<=t) one-tail	2.08361E-20	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	4.16721E-20	
t Critical two-tail	2.001717484	

Green 3.2: Validity of Forced Analysis

1BC

For this analysis, we have a Validity Forced Vanilla EA against a Validity Forced Penalty Constraint EA. With the same logic as before we expect to see the penalty constraint to perform better.

Validity Forced EA	Validity Forced Penalty Constraint EA
0	87
0	80
97	97
0	87
93	93
97	97
0	87
97	97
97	97
93	93
97	97
0	83
0	90
0	90
0	83
100	100
0	87
97	97
0	87
97	97
97	97
97	97
0	87
97	97
93	93
0	87
93	93
83	83
93	93
97	97

F-Test Two-Sample for Variances

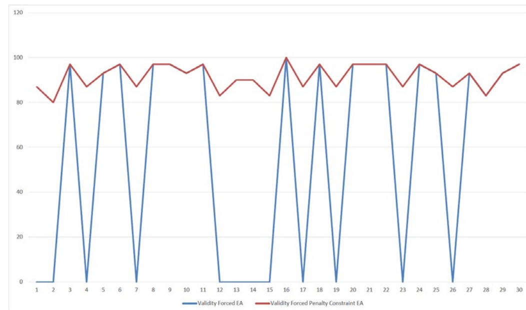
	Variable 1	*Variable 2*
Mean	57.16666667	91.66666667
Variance	2261.798851	31.40229885
Observations	30	30
df	29	29
F	72.02653734	
P(F<=f) one-tail	3.16084E-20	
F Critical one-tail	1.860811435	

F < F Critical

assume equal

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	57.16666667	91.66666667
Variance	2261.798851	31.40229885
Observations	30	30
Pooled Variance	1146.600575	
Hypothesized Mean Difference	0	
df	58	
t Stat	-3.94601423	
P(T<=t) one-tail	0.000108557	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	0.000217115	
t Critical two-tail	2.001717484	



For the next analysis, we compare validity forced vs a plain vanilla EA. Since the validity is enforced, we see that the Validity Forced EA is performing slightly better than the plain Vanilla EA which is to be expected.

Validity Forced EA	Vanilla EA
0	0
0	100
97	0
0	0
93	93
97	0
0	0
97	0
97	97
93	0
97	0
0	0
0	0
0	0
0	0
100	0
0	0
97	0
0	0
97	93
97	0
97	0
0	97
97	0
93	0
0	0
93	100
83	0
93	0
97	97

F-Test Two-Sample for Variances

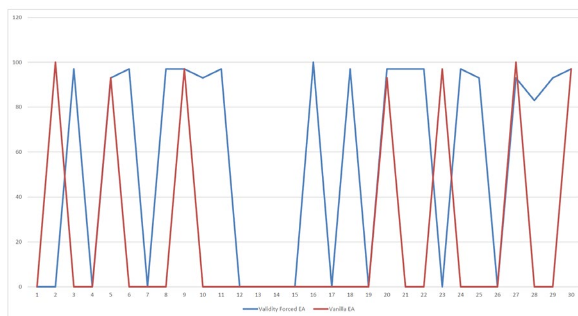
	Variable 1	*Variable 2*
Mean	57.16666667	22.56666667
Variance	2261.798851	1732.667816
Observations	30	30
df	29	29
F	1.305385158	
P(F<=f) one-tail	0.238737105	
F Critical one-tail	1.860811435	

F < F Critical

assume unquel

t-Test: Two-Sample Assuming Unequal Variances

	Variable 1	*Variable 2*
Mean	57.16666667	22.56666667
Variance	2261.798851	1732.667816
Observations	30	30
Hypothesized Mean Difference	0	
df	57	
t Stat	2.998522593	
P(T<=t) one-tail	0.002007823	
t Critical one-tail	1.672028888	
P(T<=t) two-tail	0.004015646	
t Critical two-tail	2.002465459	



2BC

Here we can see the same thing except with the problem 2BC.

Vanilla Validity EA	Penalty Validity Ea
89	89
97	97
77	77
86	86
80	80
77	77
0	90
89	89
0	87
77	77
77	77
0	87
89	89
89	89
89	89
0	90
0	90
0	80
0	76
0	90
86	86
0	87
86	86
86	86
0	84
0	90
0	90
80	80
0	84
94	94

F-Test Two-Sample for Variances

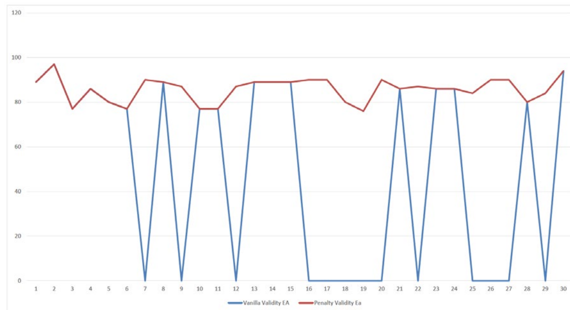
	Variable 1	Variable 2
Mean	48.26666667	85.76666667
Variance	1864.133333	29.9091954
Observations	30	30
df	29	29
F	62.32642865	
P(F<=f) one-tail	2.43015E-19	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	Variable 2
Mean	48.26666667	85.76666667
Variance	1864.133333	29.9091954
Observations	30	30
Pooled Variance	947.0212644	
Hypothesized Mean Difference	0	
df	58	
t Stat	-4.719511318	
P(T<=t) one-tail	7.67883E-06	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	1.53577E-05	
t Critical two-tail	2.001717484	



Vanilla Validity EA	Vanilla EA
89	0
97	0
77	0
86	0
80	0
77	0
0	0
89	0
0	0
77	0
77	0
0	0
89	0
89	0
89	0
0	0
0	0
0	0
0	0
0	0
86	0
0	0
86	0
86	0
0	0
0	0
0	97
80	0
0	0
94	89

F-Test Two-Sample for Variances

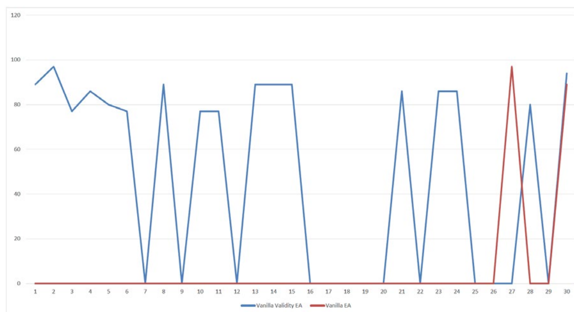
	Variable 1	Variable 2
Mean	48.26666667	6.2
Variance	1864.133333	557.8206897
Observations	30	30
df	29	29
F	3.341814615	
P(F<=f) one-tail	0.000870641	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	Variable 2
Mean	48.26666667	6.2
Variance	1864.133333	557.8206897
Observations	30	30
Pooled Variance	1210.977011	
Hypothesized Mean Difference	0	
df	58	
t Stat	4.681831511	
P(T<=t) one-tail	8.77532E-06	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	1.75506E-05	
t Critical two-tail	2.001717484	



Jordan Sosnowski

C1

Here we can see the same thing except with the problem C1 which is meant to be harder than the BC problems.

Vanilla Penalty EA	Penalty Validity Ea
27	35
50	29
22	12
23	17
29	25
28	10
29	21
29	12
27	21
22	20
17	20
52	7
30	25
55	35
17	29
37	20
21	39
21	42
13	43
50	19
57	29
33	33
21	8
35	28
18	11
19	7
16	36
31	29
26	28
34	33

F-Test Two-Sample for Variances

	Variable 1	Variable 2
Mean	29.63333333	24.1
Variance	147.6885057	110.9896552
Observations	30	30
df	29	29
F	1.330651091	
P(F<=f) one-tail	0.223237388	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variances

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	Variable 2
Mean	29.63333333	24.1
Variance	147.6885057	110.9896552
Observations	30	30
Pooled Variance	129.3390805	
Hypothesized Mean Difference	0	
df	58	
t Stat	1.884376061	
P(T<=t) one-tail	0.032264375	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	0.064528751	
t Critical two-tail	2.001717484	

C2

Vanilla Penalty EA	Penalty Validity Ea
12	16
7	9
6	12
3	20
8	2
31	22
31	19
16	12
18	21
11	18
8	35
12	8
14	36
31	7
23	13
12	18
1	20
29	27
6	31
9	17
25	22
19	12
19	23
13	14
30	19
16	19
24	33
15	24
29	14
-9	21

F-Test Two-Sample for Variances

	Variable 1	Variable 2
Mean	15.63333333	18.8
Variance	102.9298851	66.16551724
Observations	30	30
df	29	29
F	1.555642415	
P(F<=f) one-tail	0.120016298	
F Critical one-tail	1.860811435	

F < F Critical

assume unequal variance

t-Test: Two-Sample Assuming Unequal Variances

	Variable 1	Variable 2
Mean	15.63333333	18.8
Variance	102.9298851	66.16551724
Observations	30	30
Hypothesized Mean Difference	0	
df	55	
t Stat	-1.333819549	
P(T<=t) one-tail	0.093880152	
t Critical one-tail	1.673033965	
P(T<=t) two-tail	0.187760304	
t Critical two-tail	2.004044783	

Green 3.3: Exploring Penalty Coefficients

For the last green objective, we were asked to explore penalty coefficients. When I first implemented the penalty coefficient, I picked 10 as that seemed like a good number, and the results, I got were not awful, so I stuck with it. For this objective, I picked a coefficient that was lower and was significantly higher; 5 and 50. My main question after doing this was what is the best way of finding a good penalty coefficient? As if you make it too high it will just kill everything, but if its too low the invalid solutions will look "good". Additionally, what is the best way of checking that information? In our logs we only check fitness but if we have a low penalty it will look like everything has a "good" fitness.

10 vs 5

We can see that a penalty of 5 outperforms a penalty coefficient of 10. However, as discussed later a lower penalty coefficient can allow seriously "invalid" solutions to flourish. So I think it would most likely be best to be a little higher but with lower performance as the solutions are most likely similar.

C1

10 Coef	5 Coef
27	57
50	57
22	45
23	42
29	53
28	50
29	56
29	54
27	73
22	66
17	47
52	58
30	49
55	54
17	52
37	58
21	64
21	44
13	62
50	49
57	59
33	49
21	57
35	52
18	47
19	56
16	56
31	52
26	56
34	57

F-Test Two-Sample for Variances

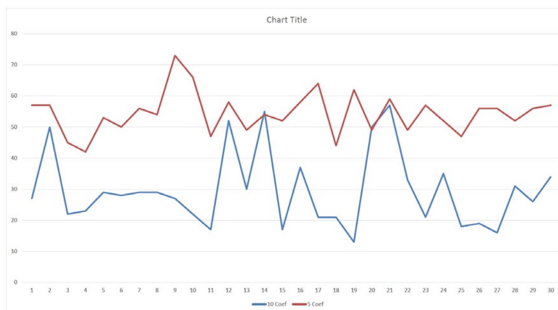
	Variable 1	*Variable 2*
Mean	29.63333333	54.36666667
Variance	147.6885057	44.86091954
Observations	30	30
df	29	29
F	3.292141741	
P(F<=f) one-tail	0.000986375	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	29.63333333	54.36666667
Variance	147.6885057	44.86091954
Observations	30	30
Pooled Variance	96.27471264	
Hypothesized Mean Difference	0	
df	58	
t Stat	-9.762749917	
P(T<=t) one-tail	3.74972E-14	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	7.49945E-14	
t Critical two-tail	2.001717484	



C2

10 Coef	5 Coef
12	48
7	53
6	54
3	50
8	46
31	58
31	65
16	55
18	66
11	57
8	48
12	53
14	46
31	52
23	53
12	53
1	48
29	48
6	48
9	47
25	53
19	53
19	51
13	53
30	51
16	49
24	46
15	49
29	58
-9	57

F-Test Two-Sample for Variances

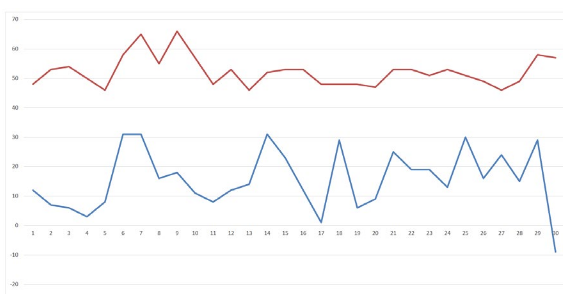
	Variable 1	*Variable 2*
Mean	15.63333333	52.26666667
Variance	102.9298851	25.5816092
Observations	30	30
df	29	29
F	4.023589145	
P(F<=f) one-tail	0.000170529	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	15.63333333	52.26666667
Variance	102.9298851	25.5816092
Observations	30	30
Pooled Variance	64.25574713	
Hypothesized Mean Difference	0	
df	58	
t Stat	-17.6997071	
P(T<=t) one-tail	2.3494E-25	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	4.69881E-25	
t Critical two-tail	2.001717484	



50 v 10

Here we have a coefficient of 50 running against 10. Logically it makes sense that the coefficient of 50 is seriously underperforming. As one mistake would have a fitness of 50 and two mistakes would have a fitness of 0 (assuming the base fitness was 100).

C1

50 Coef	10 Coef
-227	27
-186	50
-213	22
-224	23
-233	29
-76	28
-219	29
-220	29
-212	27
-173	22
-240	17
-115	52
-167	30
-170	55
-264	17
-221	37
-158	21
-232	21
-212	13
-162	50
-180	57
-125	33
-290	21
-185	35
-270	18
-179	19
-180	16
-258	31
-183	26
-219	34

F-Test Two-Sample for Variances

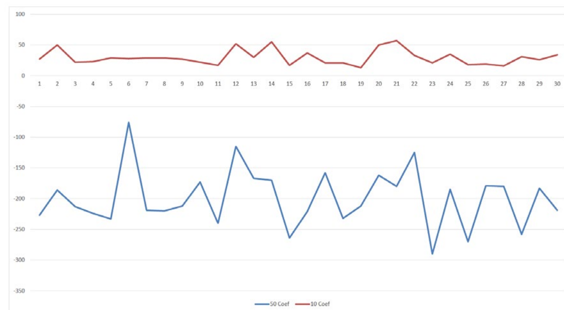
	Variable 1	*Variable 2*
Mean	-199.7666667	29.63333333
Variance	2193.081609	147.6885057
Observations	30	30
df	29	29
F	14.84937232	
P(F<=f) one-tail	6.91186E-11	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	-199.7666667	29.63333333
Variance	2193.081609	147.6885057
Observations	30	30
Pooled Variance	1170.385057	
Hypothesized Mean Difference	0	
df	58	
t Stat	-25.97016201	
P(T<=t) one-tail	6.24425E-34	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	1.24885E-33	
t Critical two-tail	2.001717484	



C2

50 Coef	10 Coef
-281	12
-68	7
-162	6
-220	3
-217	8
-214	31
-165	31
-220	16
-213	18
-224	11
-212	8
-317	12
-169	14
-173	31
-177	23
-212	12
-275	1
-223	29
-111	6
-227	9
-182	25
-274	19
-227	19
-173	13
-211	30
-271	16
-235	24
-223	15
-166	29
-217	-9

F-Test Two-Sample for Variances

	Variable 1	*Variable 2*
Mean	-208.6333333	15.63333333
Variance	2519.550575	102.9298851
Observations	30	30
df	29	29
F	24.47831913	
P(F<=f) one-tail	9.71105E-14	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	-208.6333333	15.63333333
Variance	2519.550575	102.9298851
Observations	30	30
Pooled Variance	1311.24023	
Hypothesized Mean Difference	0	
df	58	
t Stat	-23.98662939	
P(T<=t) one-tail	4.25626E-32	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	8.51253E-32	
t Critical two-tail	2.001717484	

Jordan Sosnowski

50 v 5

As seen earlier 5 performs the “best” so it is no surprise that it outperforms 50.

C1

50 Coef	5 Coef
-227	57
-186	57
-213	45
-224	42
-233	53
-76	50
-219	56
-220	54
-212	73
-173	66
-240	47
-115	58
-167	49
-170	54
-264	52
-221	58
-158	64
-232	44
-212	62
-162	49
-180	59
-125	49
-290	57
-185	52
-270	47
-179	56
-180	56
-258	52
-183	56
-219	57

F-Test Two-Sample for Variances

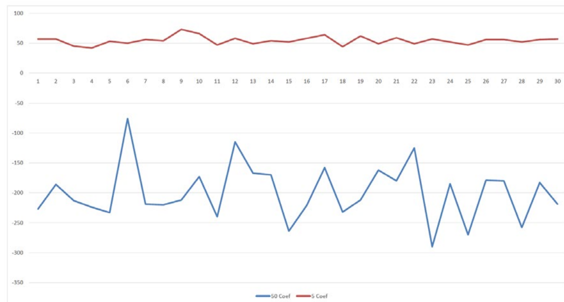
	Variable 1	*Variable 2*
Mean	-199.7666667	54.36666667
Variance	2193.081609	44.86091954
Observations	30	30
df	29	29
F	48.88623844	
P(F<=f) one-tail	7.3145E-18	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	-199.7666667	54.36666667
Variance	2193.081609	44.86091954
Observations	30	30
Pooled Variance	1118.971264	
Hypothesized Mean Difference	0	
df	58	
t Stat	-29.42373466	
P(T<=t) one-tail	7.39895E-37	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	1.47979E-36	
t Critical two-tail	2.001717484	



C2

50 Coef	5 Coef
-281	48
-68	53
-162	54
-220	50
-217	46
-214	58
-165	65
-220	55
-213	66
-224	57
-212	48
-317	53
-169	46
-173	52
-177	53
-212	53
-275	48
-223	48
-111	48
-227	47
-182	53
-274	53
-227	51
-173	53
-211	51
-271	49
-235	46
-223	49
-166	58
-217	57

F-Test Two-Sample for Variances

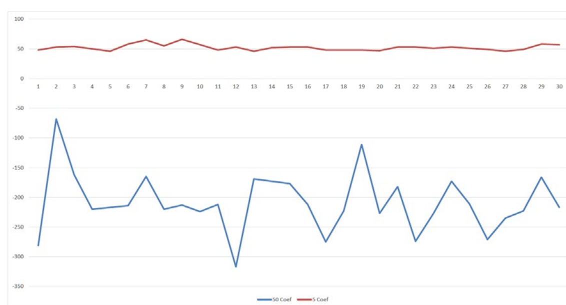
	Variable 1	*Variable 2*
Mean	-208.6333333	52.26666667
Variance	2519.550575	25.5816092
Observations	30	30
df	29	29
F	98.49069914	
P(F<=f) one-tail	3.73781E-22	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	-208.6333333	52.26666667
Variance	2519.550575	25.5816092
Observations	30	30
Pooled Variance	1272.566092	
Hypothesized Mean Difference	0	
df	58	
t Stat	-28.3256272	
P(T<=t) one-tail	5.83337E-36	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	1.16667E-35	
t Critical two-tail	2.001717484	



Yellow 1: Self-adaptive Mutation Rate

A self-adaptive mutation rate, as far as I understand, is having the mutation rate being defined as a gene that can be passed between parents instead of something that is defined to be static for the entire experiment. As I used a class-based approach for my Individuals all I had to do was add a new instance variable to it. Then during recombination, mutation, and initialization I had to ensure to generate/pick/mutate/use that new gene. I did expect the self-adaptive mutation to do better but they perform very similar.

C1

self-adaptivity mutation	Base
36	35
22	29
15	12
37	17
25	25
0	10
13	21
7	12
28	21
28	20
36	20
46	7
23	25
33	35
13	29
29	20
7	39
19	42
20	43
50	19
4	29
22	33
8	8
6	28
19	11
33	7
19	36
18	29
18	28
21	33

F-Test Two-Sample for Variances

	Variable 1	*Variable 2*
Mean	21.83333333	24.1
Variance	149.454023	110.9896552
Observations	30	30
df	29	29
F	1.346558134	
P(F<=f) one-tail	0.213929127	
F Critical one-tail	1.860811435	

F < F Critical

assume unequal variance

t-Test: Two-Sample Assuming Unequal Variances

	Variable 1	*Variable 2*
Mean	21.83333333	24.1
Variance	149.454023	110.9896552
Observations	30	30
Hypothesized Mean Difference	0	
df	57	
t Stat	-0.76929228	
P(T<=t) one-tail	0.222448018	
t Critical one-tail	1.672028888	
P(T<=t) two-tail	0.444896037	
t Critical two-tail	2.002465459	

C2

self-adaptivity mutation	base
21	16
23	9
16	12
26	20
35	2
25	22
18	19
27	12
14	21
29	18
24	35
2	8
14	36
15	7
0	13
29	18
11	20
37	27
18	31
15	17
15	22
24	12
21	23
5	14
17	19
13	19
24	33
13	24
43	14
23	21

F-Test Two-Sample for Variances

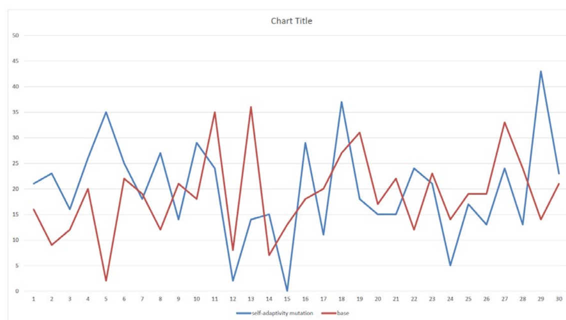
	Variable 1	*Variable 2*
Mean	19.9	18.8
Variance	92.78275862	66.16551724
Observations	30	30
df	29	29
F	1.402282677	
P(F<=f) one-tail	0.183944101	
F Critical one-tail	1.860811435	

F < F Critical

assume unequal variances

t-Test: Two-Sample Assuming Unequal Variances

	Variable 1	*Variable 2*
Mean	19.9	18.8
Variance	92.78275862	66.16551724
Observations	30	30
Hypothesized Mean Difference	0	
df	56	
t Stat	0.477887204	
P(T<=t) one-tail	0.31729535	
t Critical one-tail	1.672522303	
P(T<=t) two-tail	0.634590699	
t Critical two-tail	2.003240719	



Red 2: Self-adaptive Penalty Coefficient

For the self-adaptive penalty coefficient, it was essentially the same as the self-adaptive mutation rate. A new gene was added to allow the penalty coefficient to be defined and to be evolved per individual. I do not think a self-adaptive penalty coefficient sounds like a good idea, however. Since we no longer throw away invalid solutions, invalid solutions can thrive with a low penalty coefficient as they will look, as far as the fitness algorithm is concerned, as valid as "valid" solutions. Therefore, the most efficient penalty coefficient would be 0, allowing extremely invalid solutions to have high fitness values. Since our logging only records overall and average fitness one cannot see the actual board of these individuals but I would assume that most of them are producing extremely invalid solutions; such as intersecting rays or unsatisfied black cells.

C1

Base	self-adaptivity penalty
35	98.51349233
29	98.76598274
12	99.93154783
17	96.43673276
25	98.89764611
10	97.27910252
21	99.68579623
12	97.62778949
21	98.88338121
20	99.14742977
20	97.4041555
7	98.50086958
25	99.23223584
35	98.42612806
29	98.9276992
20	98.02070913
39	95.90952239
42	98.6019246
43	99.57625481
19	98.16446438
29	99.17504075
33	98.28261916
8	97.69119086
28	99.95488727
11	99.77693392
7	99.3815715
36	96.94309693
29	98.33545797
28	97.80100989
33	98.9341456

F-Test Two-Sample for Variances

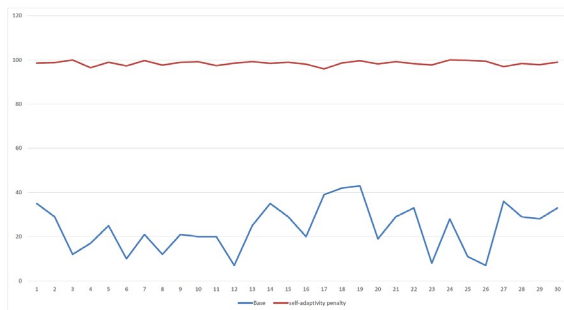
	Variable 1	*Variable 2*
Mean	24.1	98.47362728
Variance	110.9896552	1.023959678
Observations	30	30
df	29	29
F	108.3926033	
P(F<=f) one-tail	9.55367E-23	
F Critical one-tail	1.860811435	

F < F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	24.1	98.47362728
Variance	110.9896552	1.023959678
Observations	30	30
Pooled Variance	56.00680743	
Hypothesized Mean Difference	0	
df	58	
t Stat	-38.48966968	
P(T<=t) one-tail	2.69868E-43	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	5.39737E-43	
t Critical two-tail	2.001717484	



C2

base	self-adaptivity penalty
16	98.58708902
9	97.77089032
12	99.86971713
20	99.39419861
2	97.51339485
22	99.40958748
19	98.89313062
12	99.84415584
21	99.73001833
18	99.97978017
35	98.81741526
8	99.6196266
36	99.12341946
7	98.86841303
13	98.09245905
18	99.29847408
20	97.07634327
27	97.3166693
31	98.81906314
17	99.92182287
22	97.71806091
12	98.64215736
23	99.19298544
14	97.83429786
19	99.77261719
19	95.28961942
33	98.5435956
24	98.22118479
14	99.38474406
21	97.69944911

F-Test Two-Sample for Variances

	Variable 1	*Variable 2*
Mean	18.8	98.67481267
Variance	66.16551724	1.137243853
Observations	30	30
df	29	29
F	58.18058904	
P(F<=f) one-tail	6.39529E-19	
F Critical one-tail	1.860811435	

F > F Critical

assume equal variance

t-Test: Two-Sample Assuming Equal Variances

	Variable 1	*Variable 2*
Mean	18.8	98.67481267
Variance	66.16551724	1.137243853
Observations	30	30
Pooled Variance	33.65138055	
Hypothesized Mean Difference	0	
df	58	
t Stat	-53.32784564	
P(T<=t) one-tail	2.77562E-51	
t Critical one-tail	1.671552762	
P(T<=t) two-tail	5.55123E-51	
t Critical two-tail	2.001717484