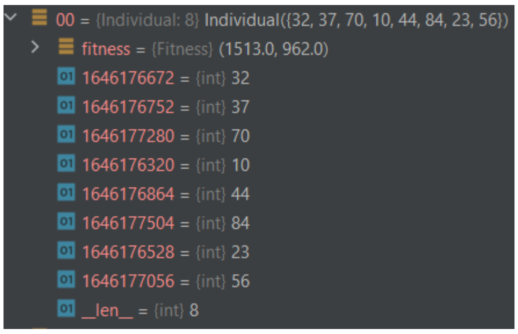
AIML426 Assignment 1 – Debruconn

Part 1 – Knapsack Problem

Overall Outline



```
00 = {Individual: 8} Individual({32, 37, 70, 10, 44, 84, 23, 56})
  fitness = {Fitness} (1513.0, 962.0)
  1646176672 = {int} 32
  1646176752 = {int} 37
  1646177280 = {int} 70
  1646176320 = {int} 10
  1646176864 = {int} 44
  1646177504 = {int} 84
  1646176528 = {int} 23
  1646177056 = {int} 56
  __len__ = {int} 8
```

For my Knapsack program, I use the DEAP library to generate class files and methods to run my genetic algorithm. I manually define the cross-over, mutation, fitness and individual representation that the algorithm uses.

Representation

Individuals in my Genetic Algorithm (GA) are defined as objects containing indices of knapsack items selected (shown in the image). Each of these indices can be used to construct a binary string to represent a GA individual. An example of an individual in my program would be {0,2,3} which would equate to the binary string of "10110000".

Fitness Function

The fitness function I have chosen was shown on the lecture slides. This essentially maximises the value of the items but minimises the weight of the individual. If the total selected items weight is over the bag's capacity, my fitness function will return a bad fitness result to make the current selection less likely to reproduce.

$$fit = \sum_{i=1}^{M} v_i x_i - \alpha * \max(0, \sum_{i=1}^{M} w_i x_i - Q)$$

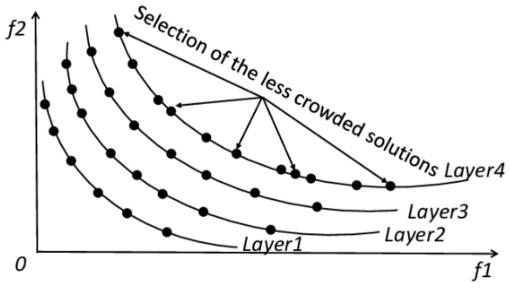
Crossover

The crossover function I use will take 2 individuals from the population, select a random index from both individuals and then exchange a section each. As my objects are represented by recording indices of a GA binary string, crossover will exchange indices that aren't necessarily consecutive.

Mutation

The mutation function will remove one random item from an individual, and then populate it with another random item. This is achieved using sets in my program to avoid duplicated items.

Selection



When selecting parents for the next generation, I use the NSGA-II selection algorithm. This algorithm works by selecting a large group of individuals first through tournament selection. Individuals are then selected based on a crowded-comparison operator using a Manhattan distance function to ensure that individuals are all mostly unique. By using the crowd-comparison operator, clustered individuals can be trimmed and therefore not dominate the next generation.

Parameters

For my parameters I have:

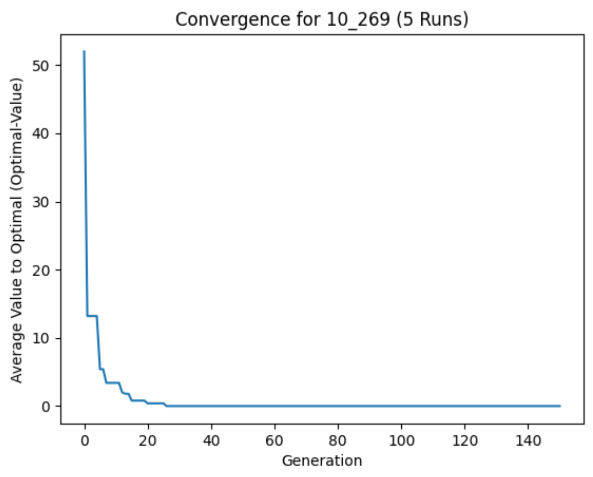
Number of generations	150
Elitism population	30
Children in each population	100
Crossover probability	65%

Mutation probability	35%

Results

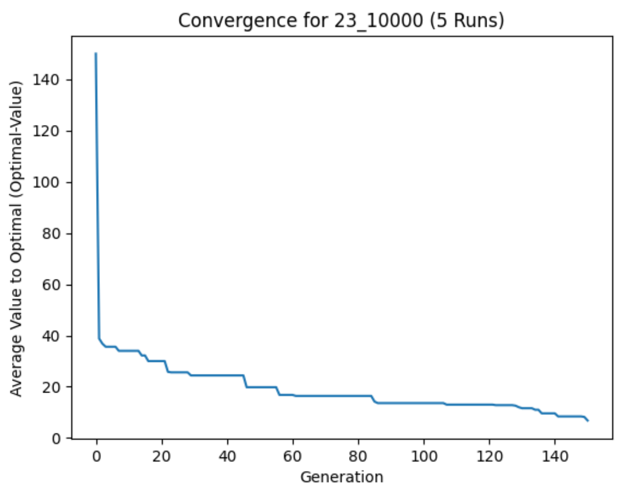
10_269 File

Run / Seed	Best Value	Weight	Standard Deviation (of best generation)
1	295	269	52.16
2	295	269	89.89
3	295	269	88.21
4	295	269	88.15
5	295	269	90.14
Mean (from past 5)	295	269	81.71
Standard Deviation	0	0	16.5446623

We can see from my convergence curve that it takes my program roughly 40 generations to reach convergence. The program on average finds an individual with roughly a 282 value after a couple generations. Because the dataset is so small, it is expected to be able to reach convergence quickly as there is only a few combinations possible to reach the optimal value. I think my program performs to standard, there however may be more optimal parameters to reach convergence faster.	

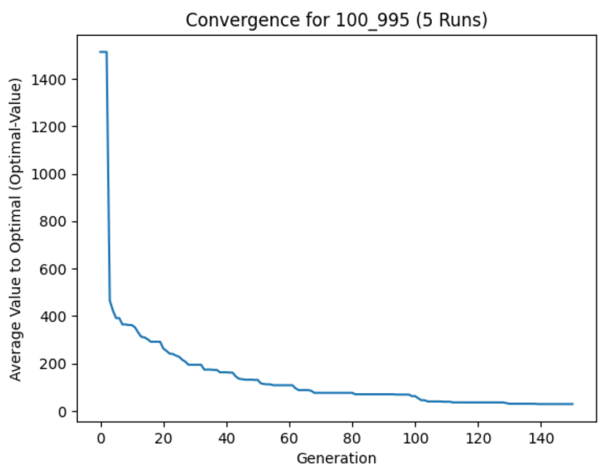
23_10000 File

Run / Seed	Best Value	Weight	Standard Deviation (of best generation)
1	9763	9774	3200.85
2	9762	9763	3088.25
3	9756	9766	3195.16
4	9753	9754	3144.62
5	9767	9768	3109.54
Mean (from past 5)	9760.2	9765	3147.684
Standard Deviation	5.6302753	7.34846923	50.1929958

Due to the high values in the data, I enlarged the section of the curve in the graph. The program very quickly gets individuals with values of 9700, which is to be expected as there are only 23 items in this dataset. There is a gradual increase in higher value individuals from generation 20 to 150. I suspect that with a higher generation number, that the optimal value will be reached in roughly 100 more generations following the current trend.	

100_995 File

Run / Seed	Best Value	Weight	Standard Deviation (of best generation)
1	1487	990	482.66
2	1484	978	471.67
3	1513	962	492.04
4	1428	990	457.90
5	1512	953	473.85
Mean (from past 5)	1484.8	974.6	475.624
Standard Deviation	34.5210081	16.6673333	12.766281

This curve is the most realistic convergence curve compared to real-world examples. There is a larger increase in fitness at the start that gradually slows down. This is likely due to the larger dataset and thus higher variability in individuals. The 2nd and 3rd generations remain at 0, this is due to every individual in the population either selecting no items or going over the bag capacity. This would likely be negative fitness values but in my program I hard-capped the values from going negative. From the convergence curve I think my program behaves adequately, so I wouldn't need to adjust the GA parameters further.	

Part 2 – Genetic Algorithm for Feature Selection

Overall Outline

For my feature selection genetic algorithm, I adapted the code from my part 1. I however had to implement a filter and a wrapper-based fitness function. Values used for the parameters have also been updated to account for the new problem / dataset.

Representation

To represent individuals in the population I store an array of indices for features to select in the data. The representation has not changed from the previous question apart from storing indices for features instead of knapsack items.

Fitness Function

For my fitness functions I use both a wrapper-based fitness function and a filter-based fitness function.

- **Wrapper based:** I create a KNN model with a K value of 5. This will plot all the points of the dataset using N dimensions (where N is the number of features. As the data has been normalised, all features are equally likely to affect the classification of an individual. The fitness function will then test against the dataset by classifying each piece of data based on the model and the 5 closest neighbours to the point.
- **Filter based:** For the filter-based fitness function I use an impurity function which will split the data into 2 groups based on the features selected by an individual. Based on the purity in the 2 sets (similar to how decision trees are split) will determine the fitness value produced.

Crossover

The crossover function in my program is the same as the previous question. 2 individuals will be parsed into my crossover function, a random index will be selected in both individuals then a segment in each individual will be swapped.

Mutation

The mutation function in my program is also the same as the previous question. An individual will be seleted with a 35% chance, a random index from the list of indices that the object holds will be replaced with a new index.

Selection

I also use the same selection algorithm as the previous question NSGA-II. This algorithm will first apply a roulette selection process and will then using a Manhattan distance function to select individuals with the largest variability. This is to avoid clusters of individuals from dominating the next generation.

Parameters

I would've liked to adjust these values further but due to time constraints (up to 2 hour run times for both datasets + both fitness functions) I could not. These are my current parameters:

Number of generations	150
Elitism population	30

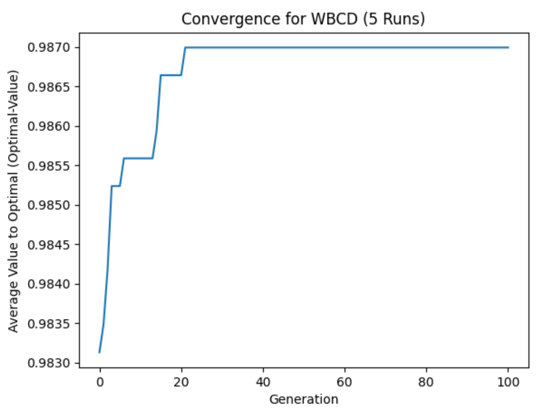
Children in each population	100
Crossover probability	65%
Mutation probability	35%

Results

WBCD Wrapper-Based (KNN)

Run/Seed	Accuracy	Standard Deviation (individuals)	Time	Best Solution
1	93.15%	2.22044605e-16	254.49s	0, 2, 6, 12, 15, 20, 21, 24, 26, 31
2	95.08%	2.22044605e-16	251.10s	0, 1, 3, 6, 7, 9, 11, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24,
3	93.85%	2.22044605e-16	244.11s	2, 3, 4, 7, 8, 10, 12, 14, 15, 16, 21, 22, 24, 25, 26, 27
4	94.20%	0	248.89s	0, 1, 2, 6, 7, 12, 14, 15, 17, 18, 20, 21, 23, 24, 25, 26, 27, 28
5	93.50%	2.22044605e-16	242.73s	1, 2, 4, 6, 8, 10, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22
Mean	93.956%	1.7764E-16	248.264s	
Standard Deviation	0.00740223	9.9301E-17	4.87541588	

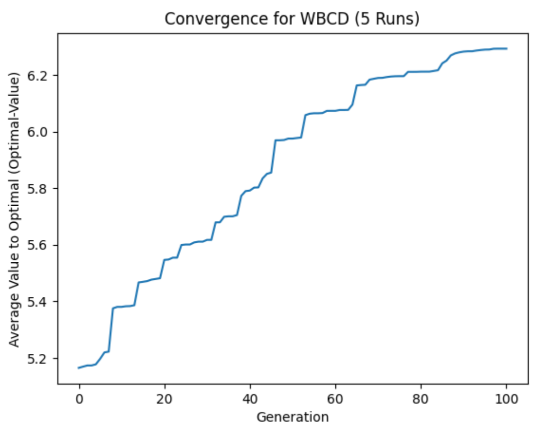
We can see from the convergence curve that my program reaches convergence early on (around gen 25). This is likely due to how I test the KNN model, individuals used in testing the model are also used in the creation, which would likely increase accuracies of the fitness function. Therefore, I have a lower fitness (93.9%) opposed to the KNN model with 98.7% accuracy.



WBCD Filter-Based

Run/Seed	Accuracy	Standard Deviation (individuals)	Time	Best Solution
1	94.02%	0.0042	761.13s	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
2	94.02%	0.0032	741.14s	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
3	93.67%	0.0043	663.86s	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
4	94.20%	0.0036	722.57s	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
5	93.67%	0.0033	896.23s	0, 1, 2, 3, 4, 6, 7, 8, 5, 10, 9, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
Mean	93.916%	0.00372	756.986S	
Standard Deviation	0.00236284	0.00050695	85.8983017	

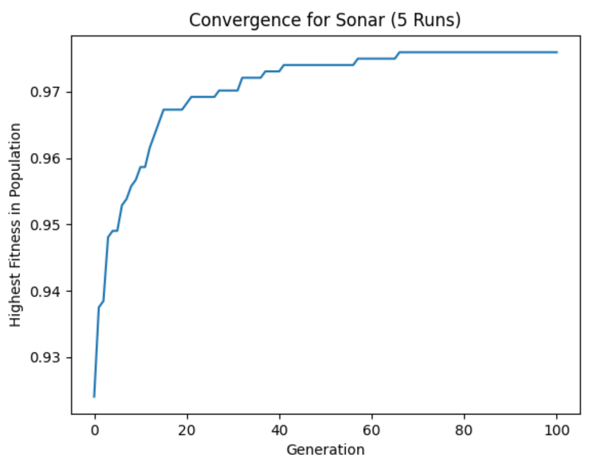
From the convergence curve we can see that the graph has a steady incline and has not met convergence yet. More generations are required to reach a higher accuracy using this filter-based fitness function. The processing time for the filter-based fitness function is almost 3 times slower than my KNN fitness function. This is likely due to the libraries I used, where the filter-based function may have suffered optimisation issues. The accuracies for both the filter-based and wrapper-based functions are surprisingly similar, but based on the



convergence curve for the filter-based function that it would likely produce higher accuracies if trained further.	

Sonar Wrapper-Based (KNN)

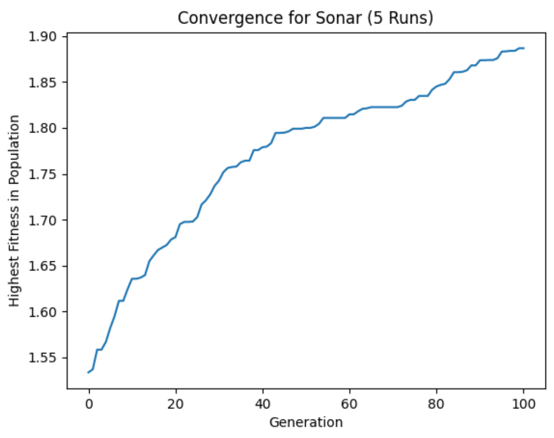
Run/Seed	Accuracy	Standard Deviation (individuals)	Time	Best Solution
1	71.15%	2.22044605e-16	106.24s	1, 4, 5, 7, 8, 9, 10, 11, 15, 16, 21, 24, 26, 32, 34, 35, 36, 39, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 53, 58, 59
2	67.79%	3.33066907e-16	97.51s	2, 4, 5, 6, 7, 8, 9, 15, 16, 17, 24, 27, 31, 34, 38, 39, 42, 45, 46, 49, 50, 53, 59
3	65.87%	3.33066907e-16	118.41s	0, 1, 2, 3, 7, 9, 10, 15, 16, 19, 23, 24, 27, 33, 43, 46, 48, 50, 56, 58
4	67.31%	3.33066907e-16	98.84s	0, 1, 2, 8, 10, 16, 19, 21, 22, 26, 27, 34, 36, 40, 41, 47, 51, 54, 57, 58
5	69.71%	3.33066907e-16	96.32s	33, 2, 3, 36, 35, 8, 10, 14, 47, 142, 21, 54, 24, 25, 56, 59
Mean	68.366%	3.1086E-16	103.464s	
Standard Deviation	0.02075688	4.9651E-17	9.20539679	

The convergence curve for the wrapper-based function on the Sonar data looks to be good, with convergence roughly around gen 75. The accuracies produced by the filter function are considerably low compared to the WBCD dataset. I think this is due to there being more possible features to select and therefore higher variability in creating individuals.	

Sonar Filter-Based

Run/Seed	Accuracy	Standard Deviation (individuals)	Time	Best Solution
1	72.11%	0.011	767.43s	0, 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 26, 28, 29, 32, 35, 36, 37, 38, 42, 43, 44, 45, 46, 47, 48, 51, 53, 54, 55, 57, 58, 59
2	73.08%	0.0060	786.48s	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15, 16, 19, 20, 21, 22, 24, 25, 27, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 39, 42, 43, 44, 45, 46, 47, 48, 49, 51, 54, 57, 58, 59
3	68.75%	0.0086	781.93s	0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 27, 28, 31, 32, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52, 57, 58, 59
4	73.08%	0.011	790.08s	0, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20, 21, 22, 23, 24, 26, 28, 31, 32, 33, 34, 36, 37, 38, 35, 40, 42, 43, 44, 45, 46, 47, 48, 49, 51, 54, 55, 57, 58, 59
5	70.19%	0.011	878.06s	0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 28, 29, 31, 32, 35, 36, 37, 38, 39, 42, 43, 44, 45, 46, 47, 48, 54, 55, 57, 58, 59

Mean	71.442%	0.00952	800.796s	
Standard Deviation	0.01912242	0.00222531	44.0418043	

The convergence curve for the sonar data looks like it hasn't made convergence yet, as there is no flatline towards the top of the curve. Compared to the previous dataset, this dataset has a much lower accuracy using the same variables. Similar to the results from the filter-based function for the WBCD dataset, more generations are required to reach convergence and produce higher accuracies. The processing time taken is exceptionally long compared to the wrapper-based function using the same dataset. Again, this would be due to optimisation issues of the library I used.	

Part 3 – NSGA II

Representation

Individuals are represented as an object that contains indices for each of the features that are selected. This is easier to process genetic functions such as crossovers / mutations. Each individual can be used to construct a binary string to represent a genetic object ({0,1,2} = "111000000" etc).

Fitness Function

For the fitness function I calculate fitness based on 2 factors: the classification error of an individual and the length of an individual. Both are minimised, where the smallest length individuals and the highest accuracy individuals are the most likely to propagate to the next generations.

Crossover

The crossover function for this part of the assignment is the same as the previous 2 parts. 2 Individuals are selected a random index is selected in each to determine segments to swap. A segment from each individual is swapped with the other.

Mutation

The mutation function is also the same as the last 2 parts. A random index from an individual is replaced with a new index from the list of possible indices.

Selection

For the selection algorithm I also use NSGA-II as I have from the previous questions. This algorithm works by selecting a large group of individuals first through tournament selection.

Individuals are then selected based on a crowded-comparison operator using a Manhattan distance function to ensure that individuals are all mostly unique. By using the crowd-comparison operator, clustered individuals can be trimmed and therefore not dominate the next generation.

Parameters

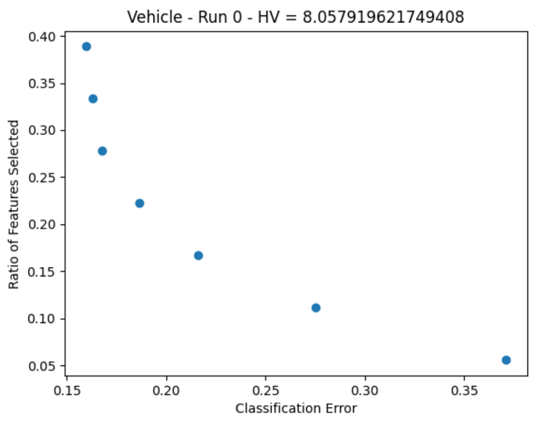
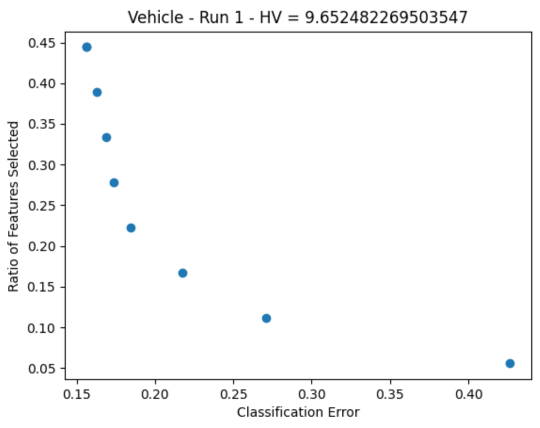
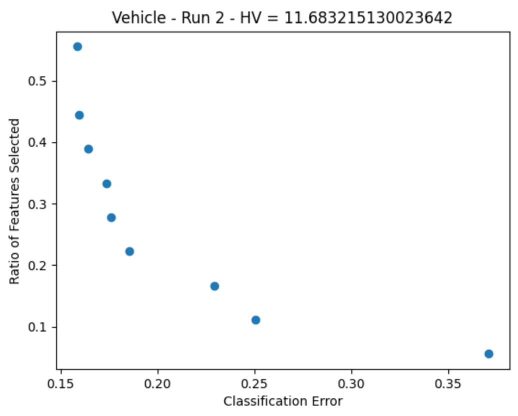
For my parameters I have:

Number of generations	150
Elitism population	30
Children in each population	100
Crossover probability	70%
Mutation probability	25%

Results

Note: The library I used to work out the hypervolume scales the hypervolume based on the number of points. To account for this I have divided the value the library provided with the number of points to get a relative hypervolume value.

Vehicle Dataset

1	Dataset = Vehicle, Run = 0

Individual	Classification Error	Ratio of Features
0	0.159574	0.388889
1	0.163121	0.333333
2	0.167849	0.277778
3	0.186761	0.222222
4	0.216312	0.166667
5	0.275414	0.111111
6	0.371158	0.055556

Hypervolume = 8.057919621749408 / 6
= 1.342

2 Dataset = Vehicle, Run = 1

Individual	Classification Error	Ratio of Features
0	0.156028	0.444444
1	0.156028	0.444444
2	0.163121	0.388889
3	0.169031	0.333333
4	0.173759	0.277778
5	0.184397	0.222222
6	0.217494	0.166667
7	0.270686	0.111111
8	0.426714	0.055556

Hypervolume = 9.652482269503547 / 8
= 1.2065

3 Dataset = Vehicle, Run = 2

Individual	Classification Error	Ratio of Features
0	0.158392	0.555556
1	0.159574	0.444444
2	0.164303	0.388889
3	0.173759	0.333333
4	0.176123	0.277778
5	0.185579	0.222222
6	0.229314	0.166667
7	0.250591	0.111111
8	0.371158	0.055556

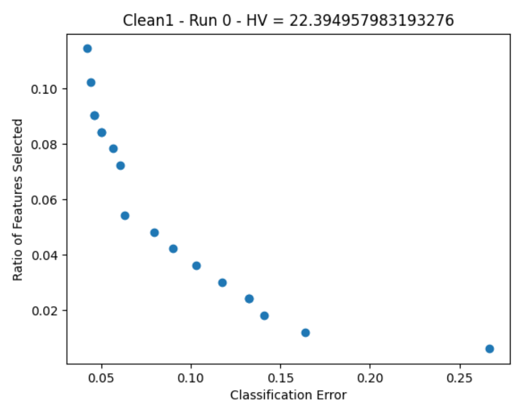
Hypervolume = 11.683215130023642 / 8
= 1.4604

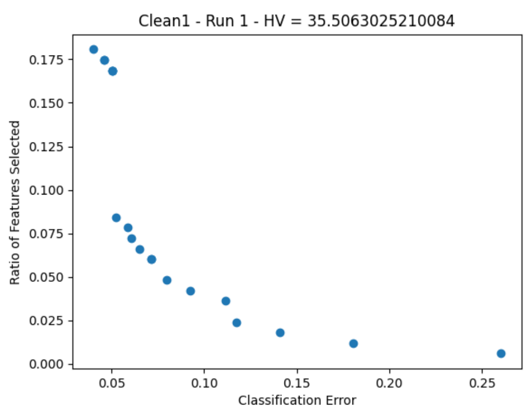
From the vehicles dataset we can see that there is a distinct curve of points in all 3 runs. The lowest number of features selected (5%) can produce a 45% classification error. By adding more features, the classification error reduces, only up to a point where using 38% of features produces the lowest classification error of 15%.

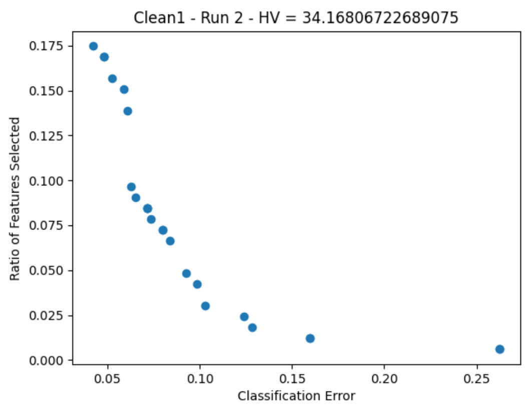
When using all features the classification error is 0.1749408983451537, this is surprising as there are a few points produced from my results which have a higher accuracy using less features. This likely means that there are some features in the vehicle dataset that harm the classification (make it less likely to classify).

Each run produces a similar number of points, and the distribution of points remains even. This indicates that the algorithm and parameters are operating effectively and producing ideal outcomes. The hypervolume also remains between 1.2 and 1.4, which indicates a lower set quality with a more even distribution between points in the pareto front.

Musk Dataset







1 Dataset = Clean1, Run = 0

Individual	Classification Error	Ratio of Features
0	0.042017	0.114458
1	0.044118	0.10241
2	0.046218	0.090361
3	0.046218	0.090361
4	0.05042	0.084337
5	0.05042	0.084337

6	0.056723	0.078313
7	0.060924	0.072289
8	0.063025	0.054217
9	0.079832	0.048193
10	0.090336	0.042169
11	0.102941	0.036145
12	0.117647	0.03012
13	0.132353	0.024096
14	0.132353	0.024096
15	0.140756	0.018072
16	0.163866	0.012048
17	0.266807	0.006024

Hypervolume = 22.394957983193276 / 17
= 1.3174

2 Dataset = Clean1, Run = 1

Individual	Classification Error	Ratio of Features
0	0.039916	0.180723
1	0.046218	0.174699
2	0.046218	0.174699
3	0.05042	0.168675
4	0.05042	0.168675
5	0.05042	0.168675
6	0.052521	0.084337
7	0.058824	0.078313
8	0.060924	0.072289
9	0.065126	0.066265
10	0.071429	0.060241
11	0.071429	0.060241
12	0.079832	0.048193
13	0.092437	0.042169
14	0.111345	0.036145
15	0.117647	0.024096
16	0.140756	0.018072
17	0.180672	0.012048
18	0.260504	0.006024

Hypervolume = 35.5063025210084 / 18
= 1.9726

3 Dataset = Clean1, Run = 2

Individual	Classification Error	Ratio of Features
0	0.042017	0.174699
1	0.048319	0.168675

2	0.048319	0.168675
3	0.052521	0.156627
4	0.058824	0.150602
5	0.060924	0.138554
6	0.063025	0.096386
7	0.065126	0.090361
8	0.071429	0.084337
9	0.071429	0.084337
10	0.071429	0.084337
11	0.073529	0.078313
12	0.079832	0.072289
13	0.079832	0.072289
14	0.084034	0.066265
15	0.092437	0.048193
16	0.098739	0.042169
17	0.102941	0.03012
18	0.12395	0.024096
19	0.128151	0.018072
20	0.159664	0.012048
21	0.159664	0.012048
22	0.262605	0.006024
23	0.262605	0.006024

Hypervolume = 34.16806722689075 / 23
= 1.4856

Selecting all features gives a 0.08823529411764708 which is worse than some of the points of data listed above with less features. Therefore, there are some features which are not useful to the classification of an individual.

The results produced by the musk dataset have a higher hypervolume value (between 1.31 and 1.97) and contain a larger number of points. This indicates that there is a better set quality than the vehicle dataset using the same algorithm. The musk dataset produces some results which are close to a 0-classification error, with the best result likely being the point using 11% of features and producing a 4.2% classification error.

In the graph for run-1 we can see a large jump between the first 3 points and the rest, while the first 3 points produces a slightly higher accuracy (1-4%) those points use double the features of the remaining. This makes the first 3 points less desirable for selection, unless accuracy is crucial for the program's application.

Part 4

Function Set

For my function set I used the following:

- Basic arithmetic operators (+, -, /, *)
- Negation operator (-)
- Absolute operator (| x |)

- Trigonometry operators (sin, cos)

I did not use additional operators such as "tan" due to the problem having a lower likelihood of using it, which could potentially reduce accuracies with redundant functions. All the operators selected are used in the functions from the handout and can in-turn produce solutions close-to or the same as the example functions.

Terminal Set

For my terminal set I used the following:

- Random integers between -10 and 10
- The argument "x", mandatory to produce non-linear functions

The random integers are capped to 10 and -10 because having too large of numbers can produce functions with exaggerated features, which in-turn will likely lead to lower accuracies or training times.

Parameters

Population	1000
Number of Generations	150
Mutation probability	65%
Crossover probability	15%
Termination Criteria	Number of generations reached
Max tree depth	Unlimited

Evaluation

For the fitness function I minimise the distance between the function my genetic program produces and the target functions. This is done by testing the function my program against a set of points (comprising of 100 points between x=-20 and x=20, every 0.20). The distance of each point in the function my produced GP function and the target function is summed. Each point's distance is also run under an absolute function to account for negative distances. Functions with the lowest total fitness are likely to continue to future generations and propagate its features to more individuals by crossover.

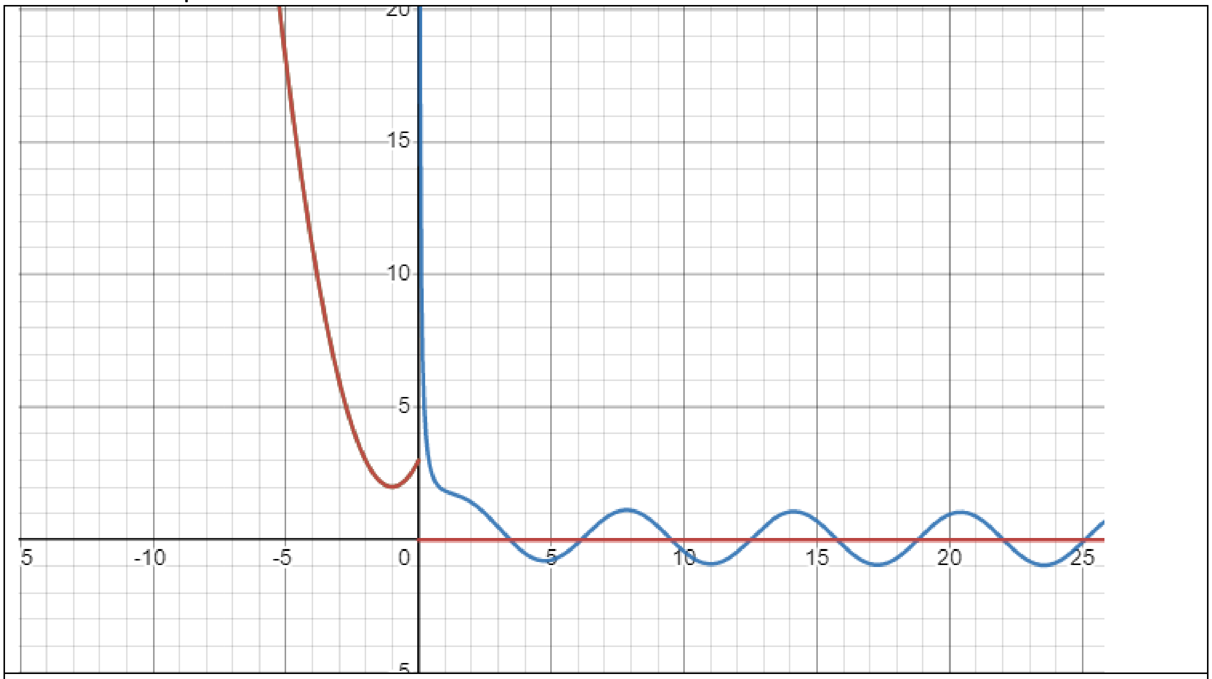
Results

Run / Seed	Fitness	Best Tree
0	0.3654166681124046	mul(div(sub(2, abs(add(add(sub(abs(abs(x)), x), abs(div(-3, abs(x)))), x))), 2), sub(x, abs(x)))
1	0.5866055693405542	sub(abs(sub(add(div(add(abs(x), sub(abs(sub(add(div(x, -2), neg(2)), abs(abs(x)))), 10)), abs(sub(add(div(add(-2, sub(abs(sub(neg(x), x)), x)), x), sin(abs(x))), x))), sub(abs(sub(neg(x), abs(sub(sub(sub(sub(abs(mul(sub(x, abs(x)), add(div(x, -2), neg(2)))), -9), x), abs(x)), 7)))), x)), x)), x)
2	1.6109340854435248	add(abs(add(add(add(x, mul(mul(x, add(sub(abs(x), x), -10)), sin(cos(-1)))), add(x, abs(x))), add(div(add(add(add(x, sub(abs(neg(x)), sub(-4, x))),

		add(x, add(x, sin(cos(add(sub(abs(x), x), -10)))))), 1), abs(x)), add(add(div(add(x, 1), add(5, 1)), -10), x)))), neg(div(add(add(add(add(sub(sub(add(add(7, x), add(add(abs(x), abs(sub(x, sin(abs(x))))), 7)), sin(abs(x))), x), sub(x, sin(abs(x)))), x), cos(div(add(add(mul(x, add(sub(abs(neg(abs(x))), x), -10)), add(add(abs(10), x), add(abs(neg(abs(x))), sin(5)))), add(x, 1)), abs(x)))), cos(div(add(add(add(x, sub(abs(neg(x)), sub(sin(x), abs(x)))), add(x, x)), add(add(add(x, mul(mul(x, add(sub(abs(x), x), -10)), sin(cos(-1)))), add(x, abs(x))), add(div(add(add(cos(-1), add(x, add(x, sin(cos(-5))))), 1), abs(x)), add(add(div(add(x, 1), add(5, 1)), -10), x)))), abs(mul(mul(x, add(sub(x, x), -10)), sin(cos(-1))))))), abs(-4))))
Mean	0.854319	
Standard Deviation	0.664516	

Run 0:

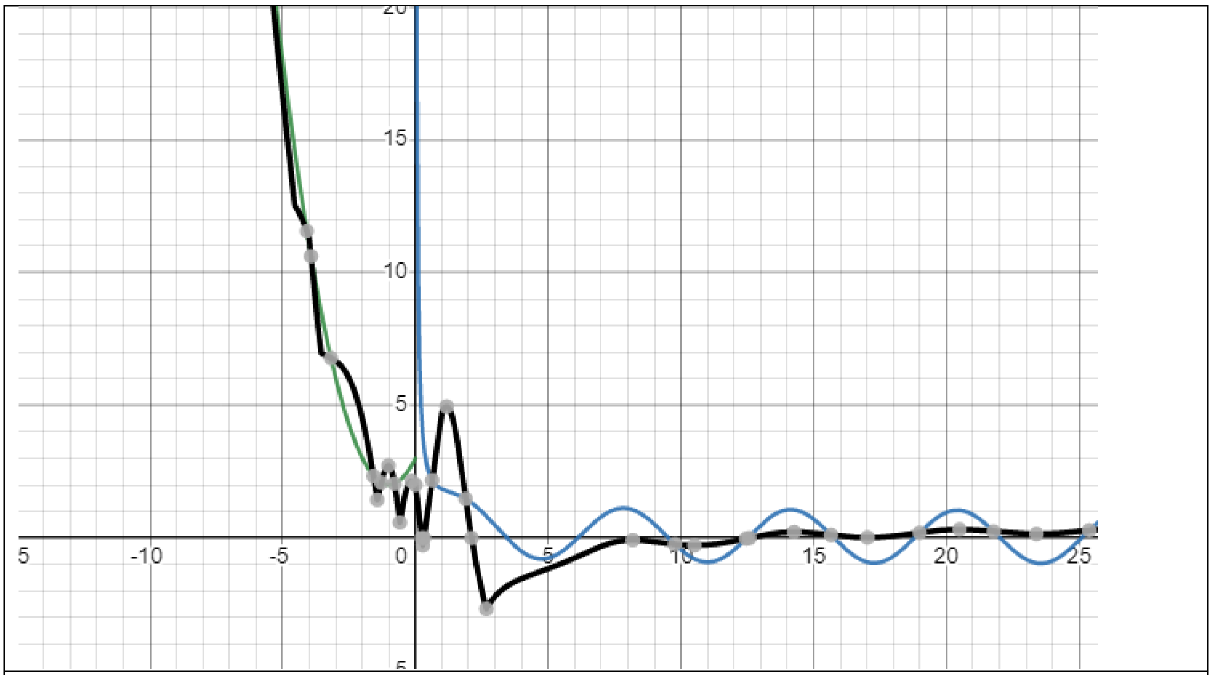
Red line = GP produced function



To reproduce on desmos.com:
(((2-\left|(((\left|\left|x\right|\right|-x)+\left|(-3/\left|x\right|)\right|)+x)\right|)/2)*(x-\left|x\right|))

Run 1:

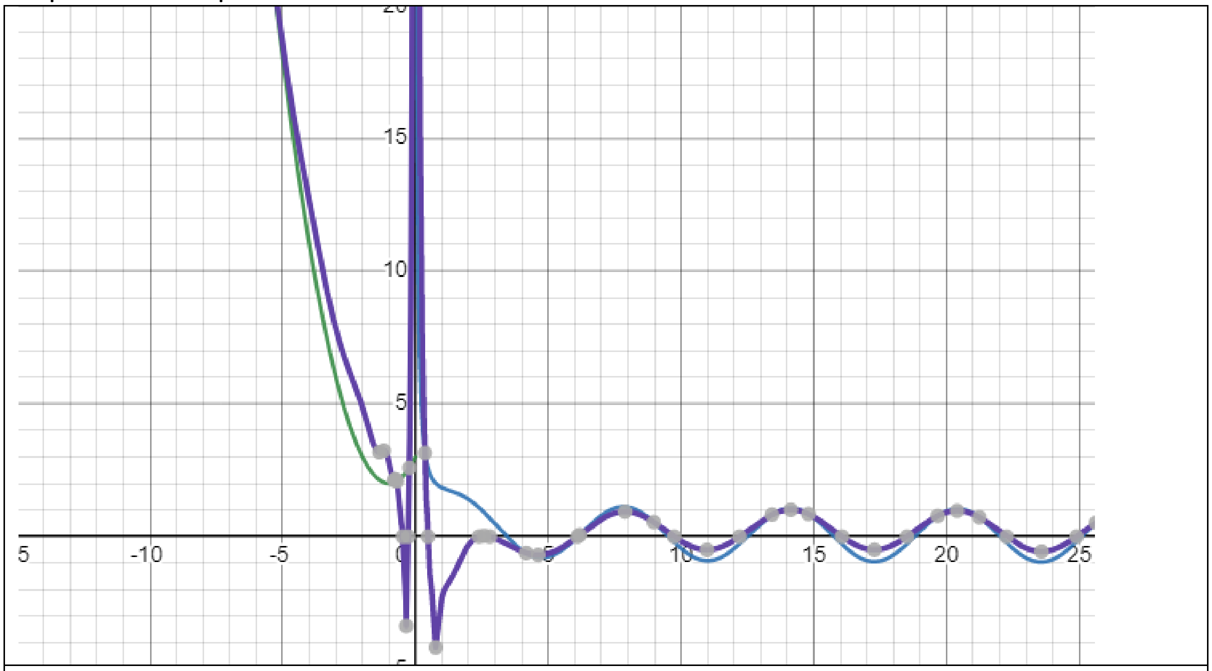
Black line = GP produced function



To reproduce on desmos.com:
(\left|((((\left|x\right|+(\left|(((x/-2)+-2)-\left|\left|x\right|\right|)\right|-10))/\left|((((-2+(\left|(-x-x)\right|-x))/x)+\sin(\left|x\right|))-x)\right|)+(\left|(-x-\left|((((\left|((x-\left|x\right|)*((x/-2)+-2))\right|--9)-x)-\left|x\right|)-7)\right|)\right|-x))-x)\right|-x)

Run 2:

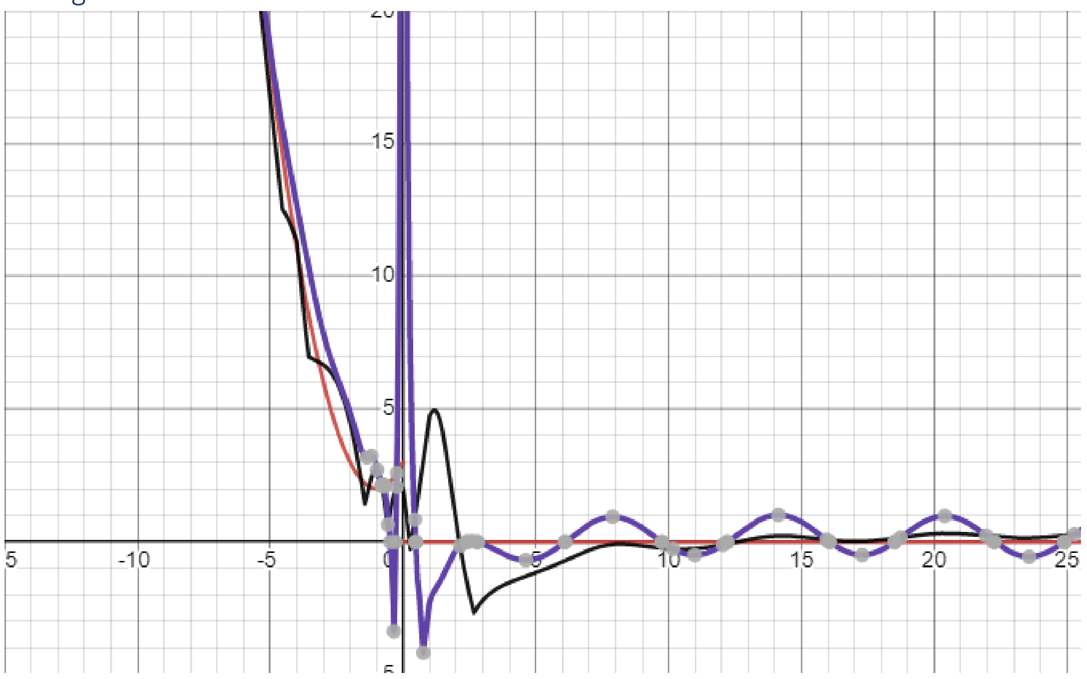
Purple line = GP produced function



To reproduce on desmos.com:
(\left|(((x+((x*((\left|x\right|-x)+-10))*\sin(\cos(-1))))+(x+\left|x\right|))+(((((x+(\left|-x\right|-(-4-x)))+(x+(x+\sin(\cos(((\left|x\right|-x)+-10))))))+1)/\left|x\right|)+((((x+1)/(5+1))+-10)+x)))\right|+-((((((((7+x)+((\left|x\right|+\left|(x-\sin(\left|x\right|))\right|)+7))-\sin(\left|x\right|))-

```
x)+(x-\sin(\left|x\right|)))+x)+\cos(((((x*((\left|-\left|x\right|\right|-x)+-
10))+((\left|10\right|+x)+(\left|-
\left|x\right|\right|+\sin(5))))+(x+1))/\left|x\right|)))+\cos(((((x+(\left|-x\right|-(\sin(x)-
\left|x\right|)))+(x+x))+(((x+((x*((\left|x\right|-x)+-10))*\sin(\cos(-
1))))+(x+\left|x\right|))+((((\cos(-1)+(x+(x+\sin(\cos(-
5)))))+1)/\left|x\right|)+((((x+1)/(5+1))+-10)+x))))/\left|((x*((x-x)+-10))*\sin(\cos(-
1)))\right|)))/\left|-4\right|))
```

All together + Discussion



The produced functions from my 3 runs all had relatively low fitness values (all lower than 1.62), which indicates that my GP is producing good solutions with the parameters I gave it.

Run 1 (red line) managed to match the left equation perfectly but not the right side. Run 3 (purple line) was able to replicate the sin wave at the correct x-intercept but not the correct wave amplitude. I predict that with more generations that can be corrected relatively easily. I did find that since I only tested values from -20 to 20, that some of my produced functions would produce random results outside of this range.

My run 1 function (red line) was the only function to account for the rapid jump at x=0, in the other 2 functions there is erratic behaviour as there is a large difference in functions. All functions matched the left side closely especially when y > 17, this is likely where most of the fitness comes from. The target function at x<=0 is easier to produce using my terminal and function set compared to the function at x>0.

Part 5

Parameters

Population	100
Φ1 (Phi 1)	1.49618

Φ2 (Phi 2)	1.49618
W (inertia)	0.7298

The Phi and inertia values I found on the lecture slides and have produced the best results from other values I have tested the PSO algorithm with.

Fitness Function

The fitness functions for this question are the Rosenbrock and Griewank functions. Each individual is parsed into the function and the individual's distance towards the goal states are measured. The fitness function minimises the distance an individual is towards the goal.

Particle encoding / Topology type

Each particle contains its position and velocities as vector arrays of d (20 or 50) dimensions. This will be its position on each dimensions plane and the positive/negative velocity towards each planes direction.

Stopping Criteria

For my stopping criteria I decided to stop the program once it has reached a maximum generation count. The maximum generation is 1000. This has been an adequate stopping criteria as results produced have all been under 20 fitness.

Results

These results are the average of 30 runs using the specified function and D value.

Function	Mean	Standard Deviation
Rosenbrock (D=20)	12.959554790847994	20.22933715442558
Griewank (D=20)	0.02132537415824042	0.013419103020393817
Griewank (D=50)	0.027280792284405688	0.023480454229282306

From the results we can see there's a clear distinction between the results of the Rosenbrock and Griewank functions. There is a lot more variability between runs using the Rosenbrock function as seen by the standard deviation of 20.2. The Griewank functions produced consistent results between each run with 0.01/0.02 standard deviations.

We can see that there is very little difference when changing the D value using the Griewank function. This should be expected as there is still equal chance to find the optimal value on every plane, whether that be with 50 dimensions or 20 dimensions. Every generation will continually update each planes position and velocity values to get optimal solutions.