Assignment 2

Test case prioritisation (TCP)

TCP is a by-product of regression testing. Test cases are generated and indicate which tests find specific faults. Instead of preforming all the tests, which can be heavily resource consuming, the ideal situation would be to do as few tests, but cover as much of the search space as possible. Therefore, prioritising certain tests that can find the highest number of faults is what the objective of TCP is.

Target

Since 1 indicates a test has found a fault, and 0 indicates no faults have been found, the ideal test would find a fault in every location it tested. Therefore, the aim of my searches is to compare the tests with the ideal target of [1, 1, 1, ……, 1]. This ideal target would find every single fault over every location it was testing with just one test.

Fitness function:

The fitness function penalises specific fault returns. If no faults are found, that means that either the system has no faults, or that the test is not fit for function. Therefore, in this case where faults are expected to be in the system, the heaviest penalty goes to not finding any faults. The next case is where the values exceed those in the ideal case. This is based on reducing the number of tests performed that can find the same fault. However, finding the same fault using several tests is better than not finding any faults. Finally, the component of the function that carries least weight is when a value matches exactly that of the target values. A weight was given to this for an indication of how many results are the same as the target, just by glancing at the fitness value of the test sample.

$$OF = F_1/10 + F_2/100 + F_3$$

Where OF is the overall fitness, F1 is the fitness value associated with values greater than target values. F2 is the fitness value associated with values equal to target values. F3 is the value associated with no faults found.

Random Search (RS)

The RS operates by first randomly selecting test case(s) from the txt file. These tests can either be concatenated together, or a single test can be selected as the sample. For this assignment, only one test case was selected for each run. That sample is then evaluated using the fitness function. The chosen number of iterations is specified by the user within the code. The output consists of a Dataframe with each of the samples in the order that they were randomly chosen. Once the search method was completed, the next stage was to iterate over those searches to create a generational population that could be used. This final

product was the highest performing achievers from all the searches. Once the highest achievers were found, then evaluating which tests would be performed in what order was found by sorting the results based on their OF.

Hill Climber (HC)

The HC operates similarly to the RS, whereby it randomly selects a location within the txt file, then creates a population with the surrounding tests around that test. Each test case was scored based on the number of faults they found. Then they were ordered and the highest performer made it to the next stage. That test case was the sample that made it onto fitness testing. New samples were made each time depending on the number of iterations required by the user. The final output consisted of all the samples collected and placed into a final DataFrame. Both in the original order they were selected and another where they were sorted by lowest OF first.

Genetic Algorithm (GA)

Never managed to get this working ☹. Therefore, I will cover how I intended on developing the key parts to make the GA. The basis is the same as the RS. However, instead of adding only one sample from the txt field to the greater population, one general population would be made first. Each individual within the population would then be evaluated using the OF. After this happens, the population would be evaluated, whereby the top scorers would be left alone, then the remaining would be put through a loop statement with nested if statements. Randomly generate sub populations by selecting individuals from the population and place into a new DataFrame. If the combined OF for the sub population was greater than a randomly chosen integer, then preform a cross over with another randomly selected sub population. Afterwards, join the sub populations together with the highest performers to create a new population. Afterwards, randomly select an individual from the bottom half of the new population and switch it (mutate) with one from the original population. Then pass this to the determine the OF for the new population. This new population is then refed back into the code, to iterate over X number of times, 10 for consistencies sake. After X times, the final population would then be the population that would be used, as well as a sorted version of that population, for comparison against the RS and HC.

Average Percentage of Faults Detected (APFD)

The data sets were then normalised using:

$$\frac{(F_1 + F_2) \Big/ No\ of\ possible\ faults}{\sum (F_1 + F_2) \Big/ No\ of\ possible\ faults}$$

Where for the small txt file, No of possible faults was always 9 and for the larger it was 39. F_1 was the number of values in the sample that were larger than the values in the target. F_2 was the number of values equal to the target values. APFD was calculated using Microsoft Excel, using $\sum(x_i + x_{i+1})/2$. Where x_i was the value associated with the sum of the normalised number of faults found per test, and x_{i+1} was just the next increment from x_i.

Toolkit used

The version of Python used was Python 3.5 with PyCharm CE.

The main toolkit used for the basis of the RS and HC was pandas, then for extra manipulation numpy was used. The main body holding information was the DataFrame. Most of the operations were carried out when the data was held within the DataFrame. Series were used so that vectors could be added together or added to DataFrames. DataFrames were limited to the operations they could perform, therefore elements of the Dataframe were passed onto a Series, and potentially turned into lists for numpy. However, keeping the data in the DataFrame was the main priority, as it kept the data together easier for viewing.

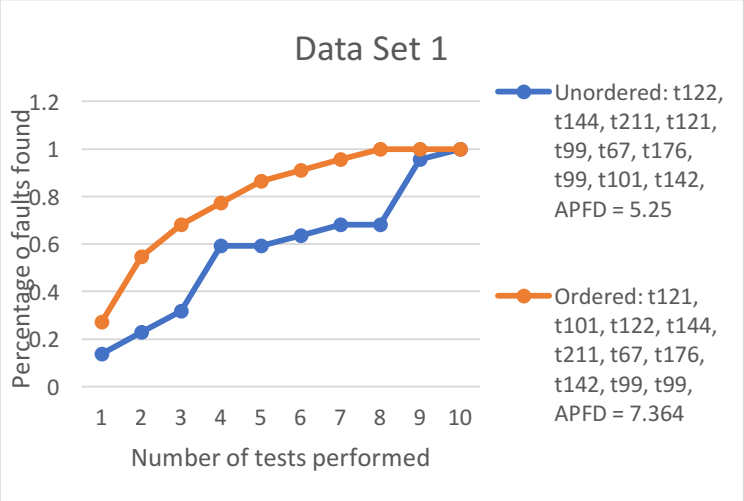
Issues that rose when making the GA were, it was initially developed using arrays and list. Due to this, it became harder to manage the data since it was held within nested arrays, within nested arrays within nested arrays etc. As well as data types becoming an issue. Therefore, managing large amounts of similar data would be best done using DataFrame.

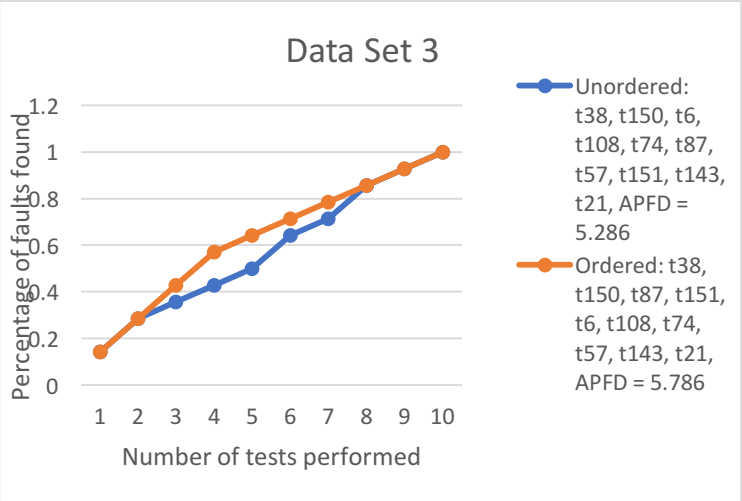
When developing the initial GA into one that could tackle the TCP problem, running the code took substantially longer than when running the code for the HC and the R, since the RS and HC were made using DataFrames whilst the developed GA was made using arrays. This was also due to pandas having been developed for managing copious amounts of data.

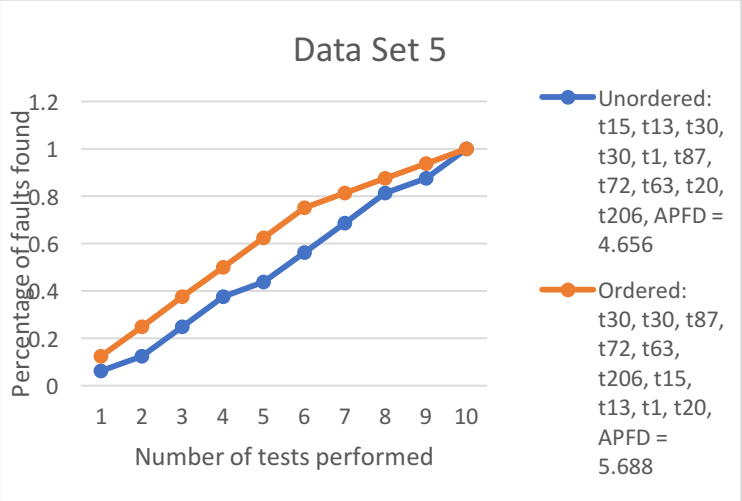
RESULTS

Random sets

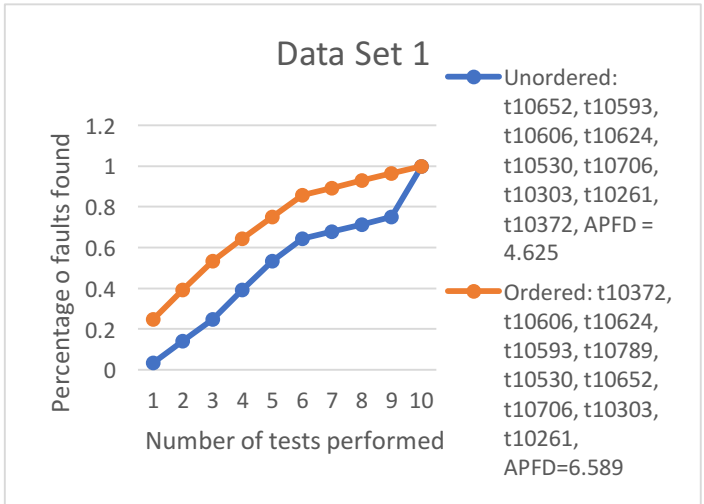
Small Set

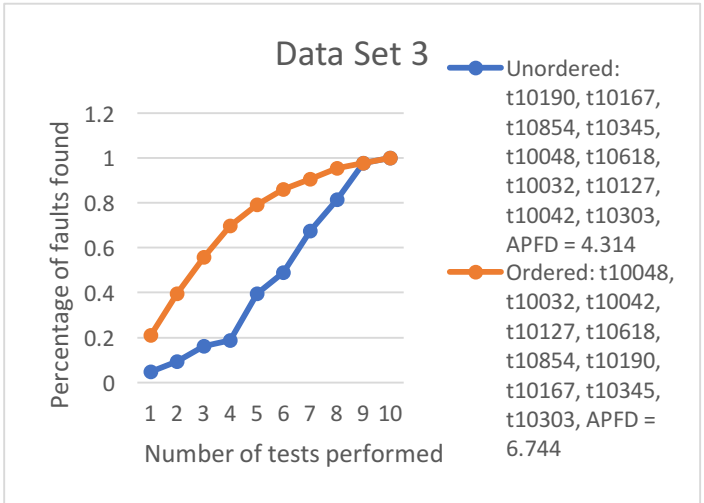


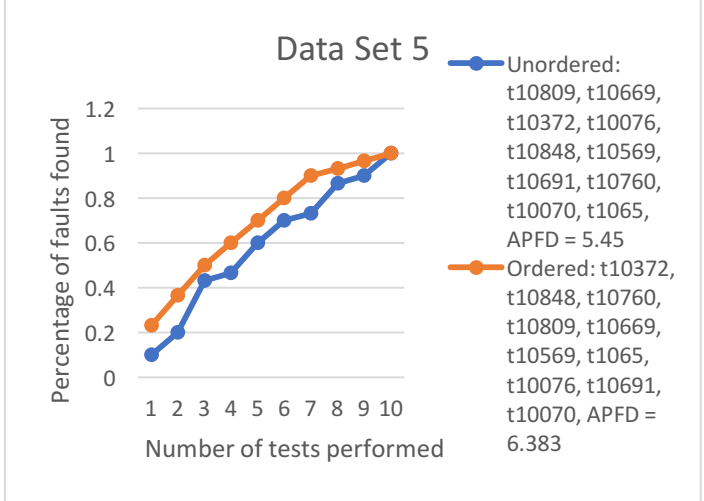




Large set

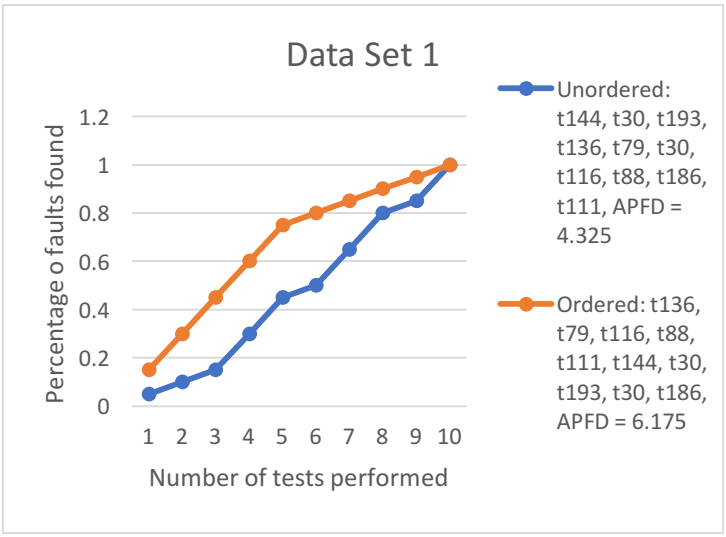


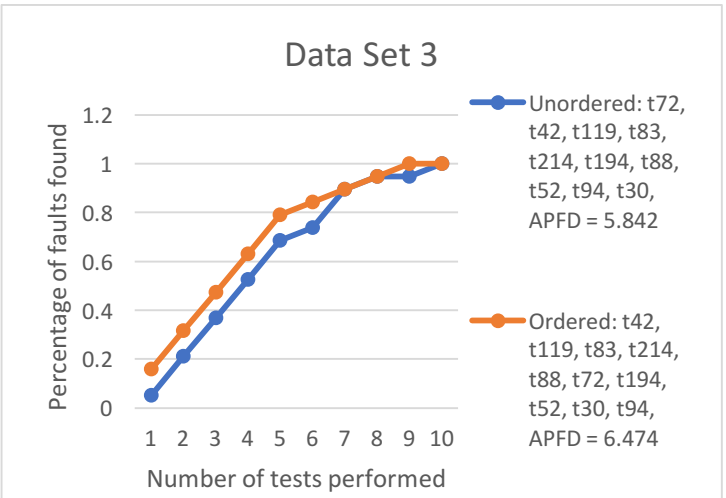


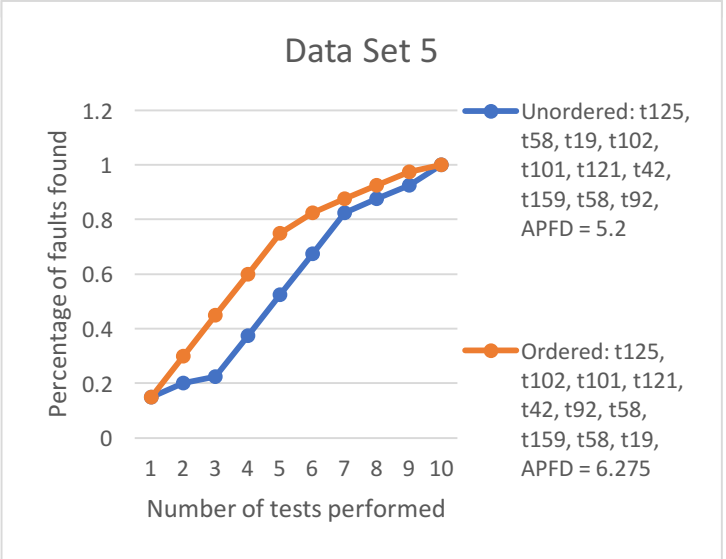


Hill Climber results

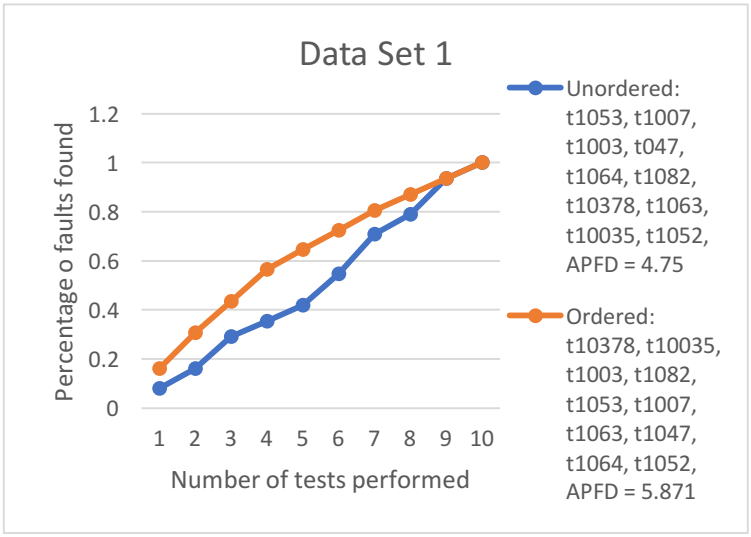
Small sets

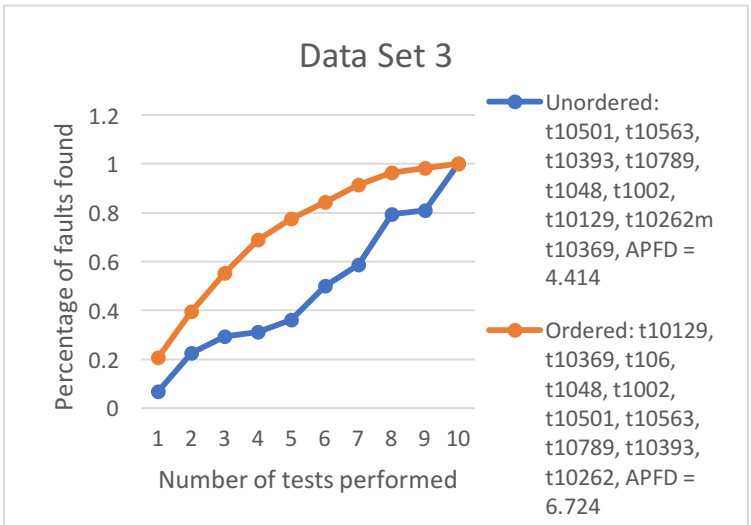


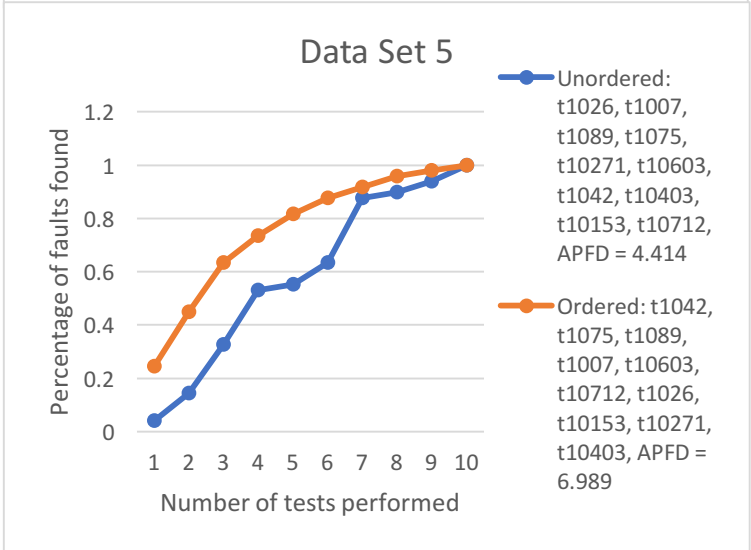




Large Set







Discussion

The baseline APFD values set by the RS, for tackling the TCP problem using small data sets are for the unordered case 4.656 and 5.688 for sorted case. Whereas for larger sets, the baseline unsorted is 4.314 and the sorted is 6.383. In contrast the HC baselines for the small set of unordered data was 4.325 and the sorted was 6.175. Whereas the larger baseline set for unordered was 4.414 and the sorted was 5.871. Therefore, despite the HC intentionally finding values that are the highest performers, it does not appear to be much better than the RS method for tackling the TCP problem.

The RS method proved just as capable as the HC for being used as a suitable method for the TCP problem, despite the only time it having any form of sorting was at the end, after all the data had been collected. On average for small datasets, RS works better on unsorted data, whereas the HC works better on sorted data. For large data sets, the RS works better for unsorted data whilst both are equally good at tackling sorted data.

Therefore, in conclusion, the HC and RS methods are equally as effective for tackling the TCP problem, since neither showed that one was definitively better than the other overall.