Reinforcement Learning for test case prioritization and selection

Introduction:

Testing in Continuous Integration (CI) involves test case prioritization, selection, and execution at each cycle. The RETECS method uses reinforcement learning to select and prioritize test cases according to their duration, previous last execution and failure history. In a constantly changing environment, where new test cases are created and obsolete test cases are deleted, the RETECS method learns to prioritize error-prone test cases higher under guidance of a reward function and by observing previous CI cycle.

Problem Formulation:

Given a test suite(consisting of several test cases), we need to select and schedule the most promising test cases (subset of the test suite), such that the total duration of the test suite is less than a particular threshold. **Note**: The algorithm is completely domain independent and functions without any knowledge about the code or the code coverage of the test cases.

Use Case:

Growth in code size and feature improvement rate has seen increased reliance on continuous integration (CI) and testing to maintain quality. Even with enormous resources dedicated to testing, we are unable to regression test each code change individually, resulting in increased lag time between code check-ins and test result feedback to developers. Also, some projects are huge and have thousands of test cases to be run which take a lot of time and resources. Prioritization and Selection can help to give early feedback to developers and if the test suite is too huge and we have a time constraint in which we need to run our test cases, it selects the most important test cases and runs them i.e to minimize the round-trip time between code commits and developer feedback on failed test cases

Terminology:

Memory Representations:

Tableau form: It consists of two tables to track seen states and selected actions. In one table it is counted how often each distinct action was chosen per state. The other table stores the average received reward for these actions. The policy is then to choose that action with highest expected reward for the current state, which can be directly read from the table. When receiving rewards, cells for each rewarded combination of states and actions are updated by increasing the counter and calculating the running average of received rewards

Note: The tableau also restricts the agent. States and actions have to be discrete sets of limited size as each state/action pair is stored separately. Furthermore, with many possible states and actions, the policy approximation takes longer to converge towards an optimal policy as more experiences are necessary for the training.

ANN: ANNs can approximate functions with continuous states and actions and are easier to scale to larger state spaces. The downside of using ANNs is more complex configuration and higher training efforts than for the tableau. In the context of RETECS, **an ANN receives a state as input to the network and outputs a single continuous action, which directly resembles the test case's priority.**

Reward Functions :

Three reward functions are used :

1. **Failure Count Reward**: In the first reward function, all test cases, both scheduled and unscheduled, receive the number of failed test cases in the schedule as a reward. It is a basic, but intuitive reward function directly rewarding the RL agent on the goal of maximizing the number of failed test cases

2. **Test Case Failure Reward:** The second reward function returns the test case's verdict as each test case's individual reward. Scheduling failing test cases is intended and therefore reinforced. If a test case passed, no specific reward is given as including it neither improved nor reduced the schedule's quality according to available information. For the test cases that are not scheduled, no reward is given.

3. **Time-ranked Reward:** The third reward function explicitly includes the order of test cases and rewards each test case based on its rank in the test schedule and whether it failed. As a good schedule executes failing test cases early, every passed test case reduces the schedule's quality if it precedes a failing test case. Each test cases is rewarded by the total number of failed test cases, for failed test cases it is the same as reward function. For passed test cases, the reward is further decreased by the number of failed test cases ranked after the passed test case to penalize scheduling passing test cases early

Evaluation Metrics :

The different evaluation metrics are the following:

1. **Normalized APFD** :

Definition 4.1. Normalized APFD

$$NAPFD(\mathcal{TS}_i) = p - \frac{\sum_{t \in \mathcal{TS}_i^{fail}} rank(t)}{|\mathcal{TS}_i^{fail}| \times |\mathcal{TS}_i|} + \frac{p}{2 \times |\mathcal{TS}_i}$$

$$\text{with } p = \frac{|\mathcal{TS}_i^{fail}|}{|\mathcal{TS}_i^{total,fail}|}$$

It measures the quality via the ranks of failure-detecting test cases in the test execution order

2. **Recall :**

Recall = Total No of Faults Detected / Total no of faults present

Other Parameters:

No of actions, History Length, State Size, learning Rate, Epsilon , Hidden Nodes etc

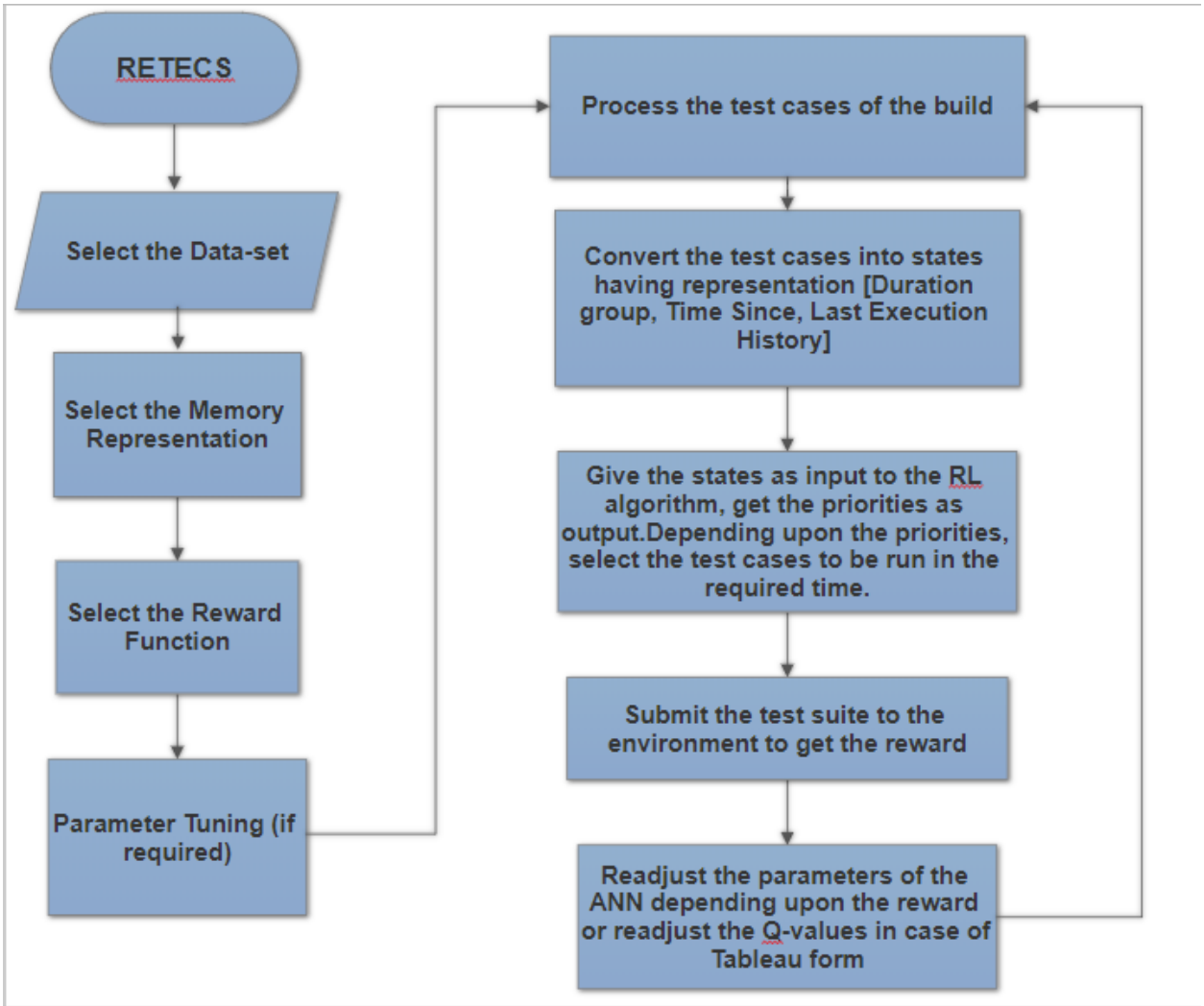
State Representation:

A test case is represented as a state consisting of the following features:

1. **Time Since Group**: This feature incorporates the knowledge of the last execution time of a test case. This feature ensures that the test cases that have been recently added or the test cases that haven't been run from quite some time get an added priority in this particular cycle. A continuous or a discrete function can be used to compute it.
2. **Duration Group**: This feature incorporates the knowledge of the approximate duration of the test case. It ensures that the test cases having a small run time are given a preference (as we want to schedule more test cases in the time available). A continuous or a discrete function can be used to compute it.
3. **Previous Execution Verdicts**: The state representation also consists of the verdicts of the last **History length** (parameter that is tuned) CI cycles. Experimentally, it is found out that having a history length of 4 performs better.

   Thus a state consists of [Time Since Group, Duration Group ,Verdict 1,Verdict 2, Verdict 3, Verdict 4]

   (Assuming the history length as 4)



RETECS basic flowchart

Timeline:

1. **Get familiar with Reinforcement Learning:**

   Watched the Lecture series on Reinforcement learning from David Silver.
   http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

   Read about Reinforcement Learning from Wikipedia.
   https://en.wikipedia.org/wiki/Reinforcement_learning

   Looked at the implementation of Reinforcement Learning from a blog.
   http://www.wildml.com/2016/10/learning-reinforcement-learning/

2. **Get familiar with ANN**

   Watched the Youtube Lecture Series .
   https://www.youtube.com/watch?v=b99UVkWzYTQ

   Read blogs that were sent by Poornima

   https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/
   http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7
   https://www.codeproject.com/Articles/16419/AI-Neural-Network-for-beginners-Part-of
   http://adventuresinmachinelearning.com/neural-networks-tutorial/
   https://www.springboard.com/blog/beginners-guide-neural-network-in-python-scikit-learn-0-18/
   https://en.wikipedia.org/wiki/Artificial_neural_network

3. **Understand the Research Paper on RL for test case prioritization**

   The paper introduces RETECS, a new method for automatically learning test case selection and prioritization in CI with the goal to minimize the round-trip time between code commits and developer feedback on failed test cases. The RETECS method uses reinforcement learning to select and prioritize test cases according to their duration, previous last execution and failure history.

4. **Setup the Environment containing all the relevant libraries for RETECS**

   Installed the relevant libraries needed for running RETECS.

5. **Test Run on ABB Robtics and Google Data Sets**

   For testing the code, it was run on ABB Robotics and Google Data

6. **Found Out Discrepancies in the Research Paper**

I found out some discrepancies in the graphs as well as the conclusion about the best reward function and memory representation made in the research paper. For verifying my observations, I did some modifications in the code (the comparison conditions) and found out that my observations were correct. I also plotted the graph for recording the same.

7. Generate CSV from JSON dump file which is compatible with RETECS code

Wrote a script which takes the JSON dump file and with the use of hash-maps and other techniques, anonymizes the data, formulates the last execution time, the verdict history and other parameters needed. It generates a CSV file having the specific fields in the required format.

8. Modify the RETECS code so that SIEMENS data runs on it

Some error regarding the timestamp was occurring while I tried to run the code on Siemens data. Did some minor modifications and the error got resolved.

9. Experimentation by changing the important functions of the code

Changed the preprocess function , increased the number of distinct layers in the preprocess discrete function, tuned the hyper parameters (no of hidden layers, activation etc). The best set of functions and hyper parameters was considered for further evaluation.

10. Improving the efficiency of the Algorithm

Significantly Improved the NAPFD average and the recall of ANN based RETECS algorithm having Test Case Failure as Reward Function by changing the action size and the preprocess function in the Code .

The NAPFD was improved from 0.47 to 0.8
The Recall was improved from 57% to 88%

11. Comparison between the Different Prioritization Techniques on Siemens Data

The results were formulated. It was found out that Failure Count reward function and Time-ranked Reward function with ANN memory representation performed significantly best .

12. Extended Test data-set for better comparison of RETECS with other Prioritization techniques.

The Siemens dataset was extended to take into account the build history of last 180 CI cycles.

13. Include the priority of the test cases in the RETECS algorithm.

After giving 8 different priorities to the test cases depending upon the "Test Type", modified the RETECS algorithm to include the priorities of the test cases as well. Different functions can be fit in depending upon the use case.

Modified the Algorithm which takes a linear sum of both the priority of the test case and the failing probability and prioritizes according to this metric. The sum of priorities detected increased by 9%. It can be further improved by adjusting the ratios of the coefficients of the sum or by making the relation non-linear.

Tested it by giving priorities based on Test types. In the Code also, I have mentioned the location where we need to modify in order to include the priority as well.

14. Study and try to think of a method to map the effects of the committed code changes in the different files to the verdict of the test cases.

Read a research paper and a blog regarding the same. It requires code-coverage techniques to develop mappings and hence is a white box approach. We don't have any data available for testing. Dropped this idea.

15. Code Setup on code.siemens + documentation .

Code Documentation was done and it was setup as a git project.

16. Read Research Paper on implementation of new prioritization technique which incorporates clustering approach, code coverage as well as the history.

Code Coverage not available in the required format for testing this approach.

17. Read Research paper on a different prioritization technique which uses Test Case Descriptions and SVR for assigning priorities to the test cases.

Test Case Descriptions not available for the undergoing projects of Siemens. The usage of Test Case Descriptions is the novel idea and without it the approach will perform sub-optimally.

18. **Extract the test case build data of Hadoop Project from Jenkins**

For testing the RETECS on other datasets, I wrote a script where you just need to specify the project name, the initial build and the final build and it extracts data from the project present on the Jenkins Server, converts it into the required format by anonmyzing the data and formulating the required fields. It also writes the test build data from the initial build to the final build specified into a CSV file in the required format.

19. **Analyze the results of the RETECS algorithm on Hadoop Project**

    Performs fairly well on Hadoop Project as well. Takes 34 seconds (on an average) for Prioritizing and Selecting the test cases of one build.

    For available time = 0.5 * Total Required time :

    Recall = 88.47 %
    NAPFD = 0.906

20. **Extract the information regarding Commit id, the files changed and the authors involved for every build for Hadoop Project from Jenkins**

    The ip is blocked by the Jenkins server after making many api calls.

Improvements implemented, abnormalities detected and other useful observations made in the RETECS code :

1. Wrote a script which takes the JSON dump file and with the use of hash-maps and other techniques, anonymizes the data, formulates the last execution time, the verdict history and other parameters needed. It generates a CSV file having the specific fields in the required format.

2. Did some minor modifications to the code, so that the Siemens Dataset runs on it. Some error regarding the parsing of the last execution time was there, corrected it.

3. The Network Representation with Failure Count as the reward function was performing poorly on the Siemens Data-set as well as on other default datasets. I changed the implementation and instead of using a classifier, used a MLP Regressor. By doing so, the efficiency of this combination improved a lot .

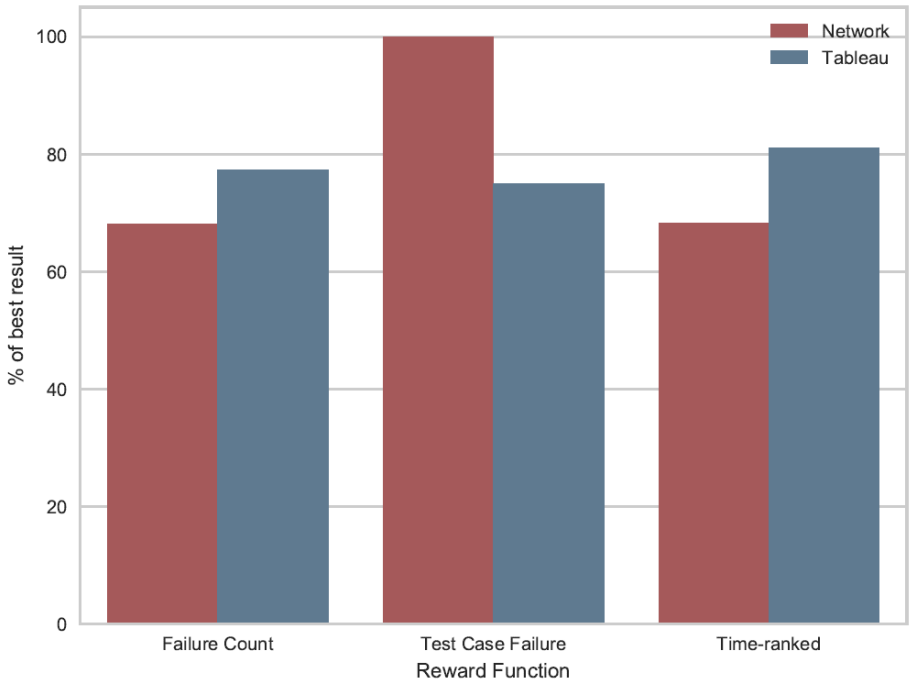
4. Several parameters are involved in the algorithm and the tuning of those parameters was done. The parameters include history length, no of hidden layers, tuning the parameters of the MLP Regressor, learning rate, epsilon etc.

5. Changed the Preprocess function which was earlier set as a discrete function to a continuous function. By doing so, the NAPFD average and the Recall improved for the ANN memory representation with Test Case Failure as the Reward Function. Checked the code on other self thought preprocess functions. The preprocess function represents

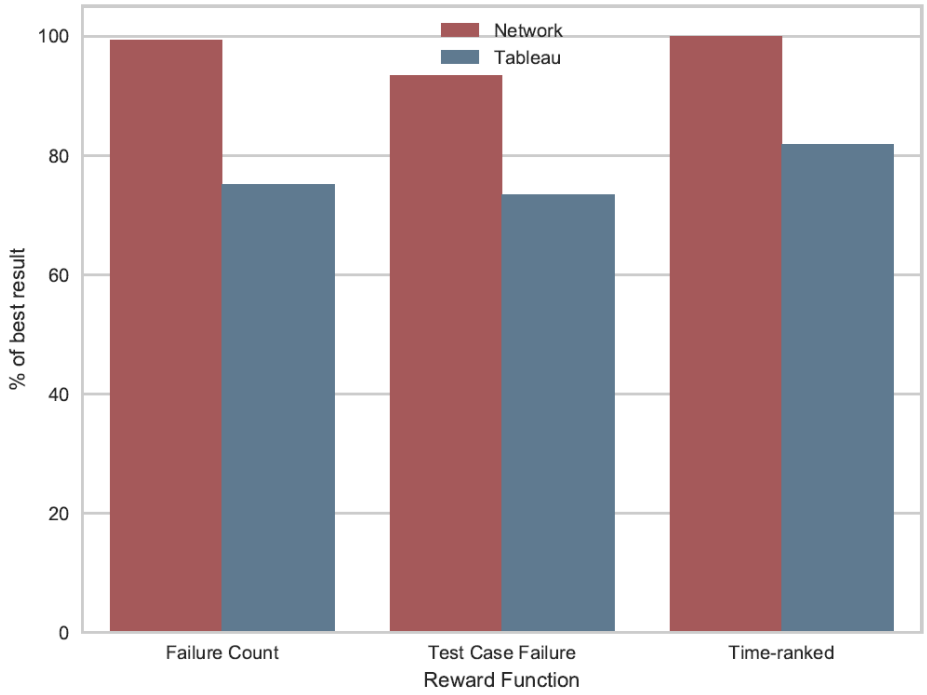
the test case as a state having the last history length verdicts, the duration group and the time since. The preprocess function should be kept as discrete for Tableau form. In the preprocess function, for the continuous case as well, the time since and the duration group should be scaled for better convergence. There were only 3 distinct levels in the discrete function. I increased the no of distinct levels, but the performance remained almost similar. For the Siemens Dataset, having the preprocess function as continuous or discrete for Test Case Failure and the Time Ranked reward functions with ANN memory representation shows similar results.

6. Found a major discrepancy in the Research paper on this topic. The analysis that they have done on the different reward functions and the graphs they have plotted on the same are not correct. After observing the code for comparison, found out a error and the conclusion thus found is not correct. In the Research Paper, it is concluded that the Test Case failure reward with ANN memory Representation performs the best, but that is false. Actually, using the Failure Count Reward function or the Time Ranked reward function with ANN memory representation performs better. A graph to illustrate this fact was also plotted.

Discrepancy in the Research Paper:



RETECS



Modified RETECS

7. Compared the reward functions after doing the above mentioned changes and a difference could be clearly seen, both in the efficiency as well as the comparison. The time ranked and the failure count reward are better reward functions for our dataset as compared to the Test Case failure reward function.

8. Tested the code for different available times (5%, 10%, 20%, 30%, 40%, 50%) and formulated results. A correct value of the time could be selected depending upon our requirements after analyzing the recall and the NAPFD on the different available times.

9. The ANN performs better than the tableau representation. This is because, in Tableau form, the no of actions are discrete (100 only) and it needs a lot of data for the training phase.

10. While testing the stochastic nature of the reward functions, it was found that using the ANN with Test Case Failure Reward function used to show varying performances while the other two reward functions are also stochastic but their performance is more or less same (only 5-7 % variation).

11. Different test cases may have different priorities. To check whether we can incorporate this knowledge in the algorithm, I gave different priorities to the test cases, depending

upon the type of the test cases. I used a weighted average of the RETECS algorithm priority and the test case original priority for selecting and prioritizing the test cases. Modified the Algorithm which takes a linear sum of both the priority of the test case and the failing probability and prioritizes according to this metric. The efficiency increased by 9% (The sum of priorities detected as the evaluation metric). It can be further improved by adjusting the ratios of the coefficients of the sum or by making the relation non-linear.

12. In the time-ranked reward, it takes a weighted sum of the entries in the state representation. In the original algorithm, there is an equal weight age to all the entries of the state. We can adjust the weights of the different entries to improve the efficiency e.g the last verdict is more important than the 3rd last verdict and hence should be given more weight.

13. For Tableau Representation, a discrete preprocess function should be use, otherwise, it would take a long time to converge and thus the efficiency decreases. This is because it builds a table of the states and the actions. For every state and action pair, there is a Q-value associated and a frequency number associated. Action is basically assigning a priority to the test cases. The action size is the number of distinct priorities that can be assigned. With 1-E probability, it performs the action with the best Q-value and with E probability, it selects a random action. The E is decreased with time.

14. In the Heur_Weight algorithm, same weights are assigned to all the 6 features of the state representation namely the duration group, the time_since and the last 4 verdicts. Altering the weights to give more weightage to the latest verdict improve s should increase the efficiency of heuristic sort. Tried this modification in the RETECS algorithm by changing the weights but the efficiency came out to be more or less the same. It could, however, improve the efficiency in other data-sets.

15. For testing the RETECS on other datasets, I wrote a script where you just need to specify the project name, the initial build and the final build and it extracts data from the project present on the Jenkins Server, converts it into the required format by anonmyzing the data and formulating the required fields. It also writes the test build data from the initial build to the final build specified into a CSV file in the required format.

Results:

Available Time as 1% of the total required time

Average NAPFD as Evaluation Metric :

Memory Representation	Reward Function	Average NAPFD
ANN	**Failure Count Reward**	0.77
ANN	**Test Case Failure Reward**	0.69
ANN	**Time-ranked Reward**	0.84
Tableau	**Failure Count Reward**	0.62
Tableau	**Test Case Failure Reward**	0.55
Tableau	**Time-ranked Reward**	0.61

RECALL as Evaluation Metric:

Memory Representation	Reward Function	Recall (%age)
ANN	**Failure Count Reward**	72.07 %
ANN	**Test Case Failure Reward**	67.7 %
ANN	**Time-ranked Reward**	81.1 %
Tableau	**Failure Count Reward**	65.3 %
Tableau	**Test Case Failure Reward**	63.53%
Tableau	**Time-ranked Reward**	63.82%

Note: After observing the dataset, it was found out that the duration of the failing test cases is small and hence in a very small time also, the algorithm is able to detect around 81% of failed test cases.

Available Time as 5% of the total required time

Average NAPFD as Evaluation Metric :

Memory Representation	Reward Function	Average NAPFD
ANN	**Failure Count Reward**	0.83
ANN	**Test Case Failure Reward**	0.82
ANN	**Time-ranked Reward**	0.86
Tableau	**Failure Count Reward**	0.63
Tableau	**Test Case Failure Reward**	0.57
Tableau	**Time-ranked Reward**	0.59

RECALL as Evaluation Metric:

Memory Representation	Reward Function	Recall (%age)
ANN	**Failure Count Reward**	80.41 %
ANN	**Test Case Failure Reward**	75.53%
ANN	**Time-ranked Reward**	85 %
Tableau	**Failure Count Reward**	68.1 %
Tableau	**Test Case Failure Reward**	66.12%
Tableau	**Time-ranked Reward**	64.01%

Available Time as 10% of the total required time

Average NAPFD as Evaluation Metric :

Memory Representation	Reward Function	Average NAPFD
ANN	**Failure Count Reward**	0.86
ANN	**Test Case Failure Reward**	0.65
ANN	**Time-ranked Reward**	0.884
Tableau	**Failure Count Reward**	0.60
Tableau	**Test Case Failure Reward**	0.58
Tableau	**Time-ranked Reward**	0.63

RECALL as Evaluation Metric:

Memory Representation	Reward Function	Recall (%age)
ANN	**Failure Count Reward**	86.49 %
ANN	**Test Case Failure Reward**	68.17 %
ANN	**Time-ranked Reward**	89.2 %
Tableau	**Failure Count Reward**	69.31 %
Tableau	**Test Case Failure Reward**	68.9 %
Tableau	**Time-ranked Reward**	67.71%

Available Time as 20% of the total required time

Average NAPFD as Evaluation Metric :

Memory Representation	Reward Function	Average NAPFD
ANN	**Failure Count Reward**	0.89
ANN	**Test Case Failure Reward**	0.75
ANN	**Time-ranked Reward**	0.90
Tableau	**Failure Count Reward**	0.56
Tableau	**Test Case Failure Reward**	0.57
Tableau	**Time-ranked Reward**	0.65

RECALL as Evaluation Metric:

Memory Representation	Reward Function	Recall (%age)
ANN	**Failure Count Reward**	92.84 %
ANN	**Test Case Failure Reward**	88.2%
ANN	**Time-ranked Reward**	93.47 %
Tableau	**Failure Count Reward**	73.35 %
Tableau	**Test Case Failure Reward**	70.20%
Tableau	**Time-ranked Reward**	71.67%

Available Time as 40% of the total required time

Average NAPFD as Evaluation Metric :

Memory Representation	Reward Function	Average NAPFD
ANN	**Failure Count Reward**	0.91
ANN	**Test Case Failure Reward**	0.75
ANN	**Time-ranked Reward**	0.917
Tableau	**Failure Count Reward**	0.66
Tableau	**Test Case Failure Reward**	0.58
Tableau	**Time-ranked Reward**	0.69

RECALL as Evaluation Metric:

Memory Representation	Reward Function	Recall (%age)
ANN	**Failure Count Reward**	97.28 %
ANN	**Test Case Failure Reward**	86.43%
ANN	**Time-ranked Reward**	97.48 %
Tableau	**Failure Count Reward**	82.51 %
Tableau	**Test Case Failure Reward**	74.8%
Tableau	**Time-ranked Reward**	77.93%

Available Time as 50% of the total required time

Average NAPFD as Evaluation Metric :

Memory Representation	Reward Function	Average NAPFD
ANN	**Failure Count Reward**	0.91
ANN	**Test Case Failure Reward**	0.85 (High Variance)
ANN	**Time-ranked Reward**	**0.92**
Tableau	**Failure Count Reward**	0.68
Tableau	**Test Case Failure Reward**	0.65
Tableau	**Time-ranked Reward**	0.70

RECALL as Evaluation Metric:

Memory Representation	Reward Function	Recall (%age)
ANN	**Failure Count Reward**	97.5 %
ANN	**Test Case Failure Reward**	High Variance
ANN	**Time-ranked Reward**	**97.87 %**
Tableau	**Failure Count Reward**	82.65 %
Tableau	**Test Case Failure Reward**	78.9%
Tableau	**Time-ranked Reward**	85.69%

Inferences:

The ANN memory representation with Time Ranked reward and the ANN memory representation with the Failure Count Reward perform better than all other combinations across all the time durations for the Siemens Data. Also, the time ranked reward performs slightly better than the failure count reward.

Note: ANN with test case failure as the reward function has high variance.

Other Prioritization Algorithms

Heuristic Sort : Prioritizes, first by last execution results, then time not executed

Results:

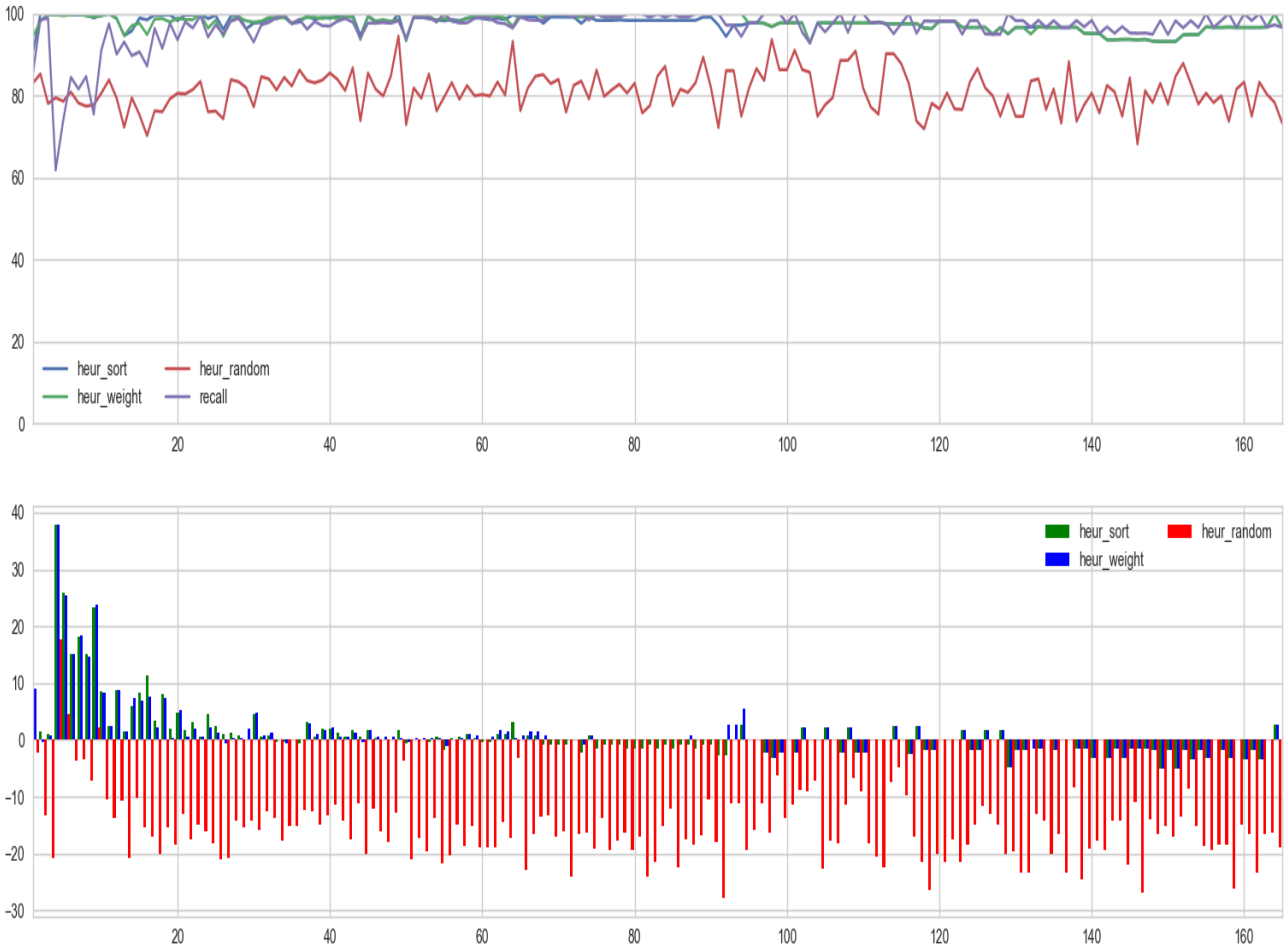
Available Time (in %age)	Average NAPFD	Recall (%age)
1%	**0.816**	**77.79**
5%	**0.84**	**82.85**
10%	**0.87**	**88.55**
20%	**0.902**	**93.07**
30%	**0.913**	**96.12**
40%	**0.916**	**97.5**
50%	**0.917**	**97.6**

Heuristic Weight: Prioritizes based on certain weights .

Available Time (in %age)	Average NAPFD	Recall (%age)
1%	**0.83**	**77.96**
5%	**0.88**	**86.56**
10%	**0.90**	**89.89**
20%	**0.92**	**94.62**
30%	**0.93**	**96.7**
40%	**0.939**	**97.7**
50%	**0.939**	**98.14**

Inferences:

For the Siemens dataset, the Heuristic weight algorithm and the ANN memory representation with the Time Ranked reward as the reward function perform the best with a very slight difference among both the two methods.



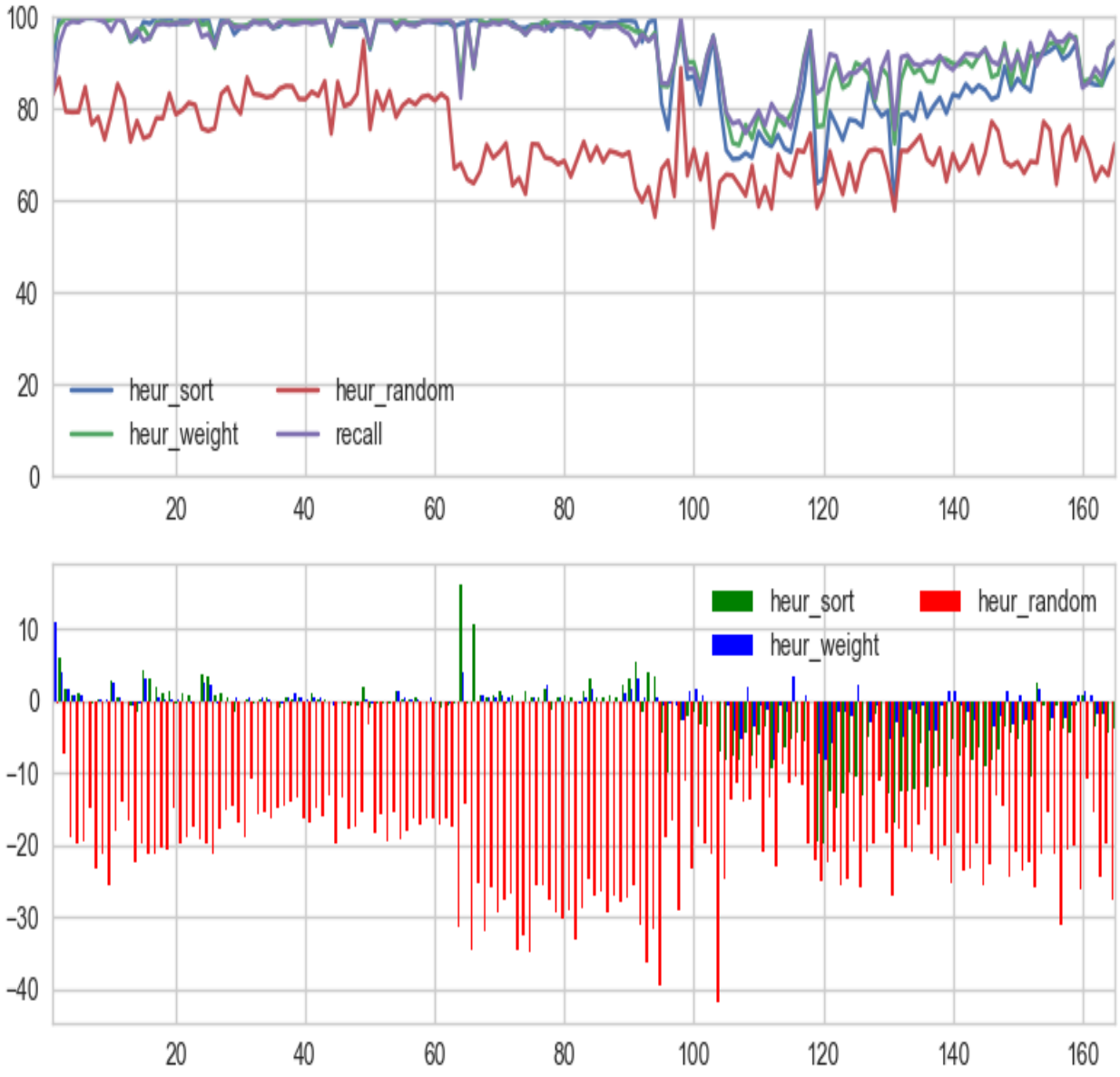
Evaluation Metric as Recall

Inference:

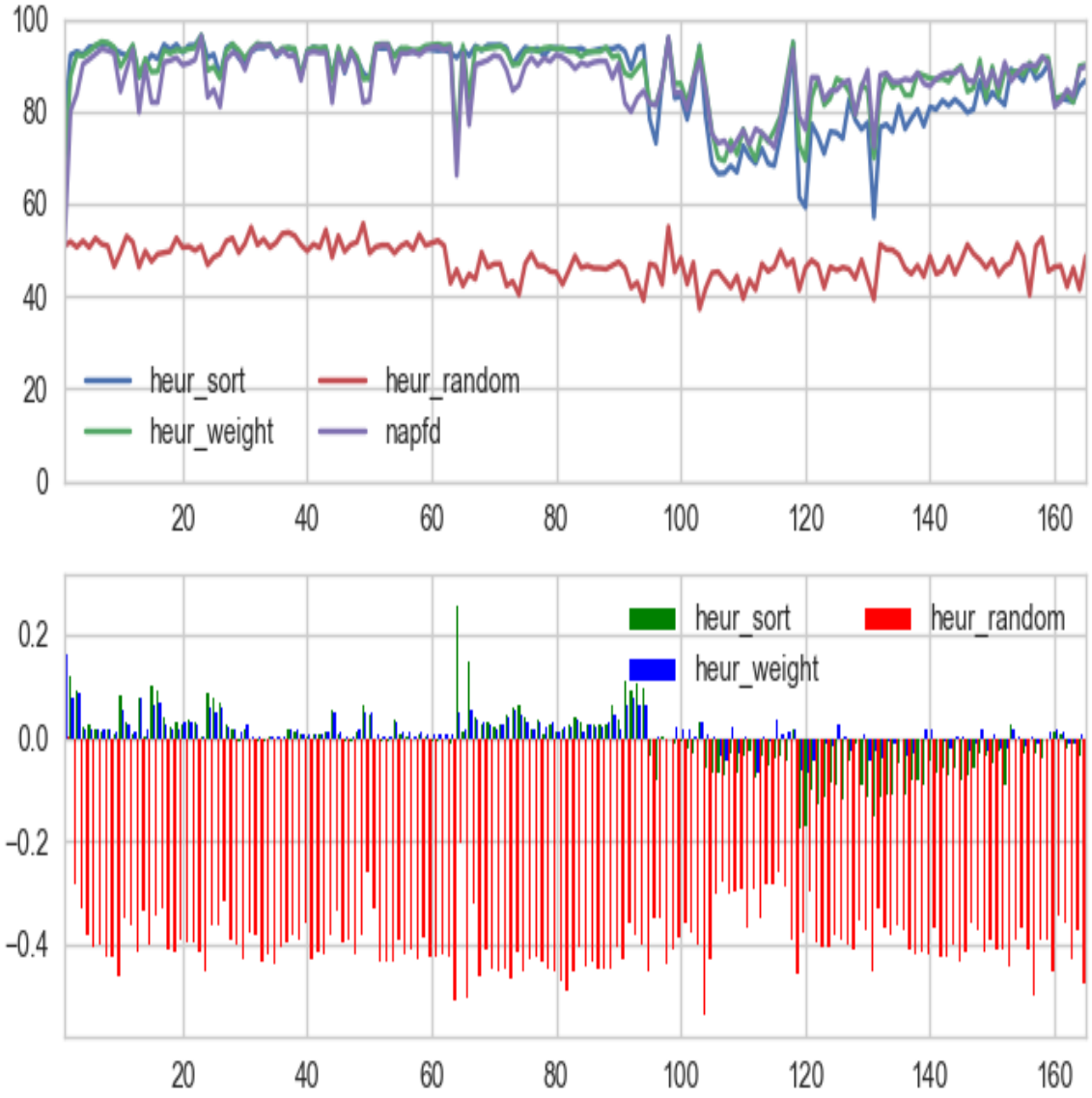
This trend can be clearly seen. In the initial 30-40 cycles, the RETECS algorithm performs poorly as compared to the other prioritization techniques, then it starts to even out and after around 100 cycles it starts performing better.

Note: The RETECS algorithm is stochastic and thus there may be a slight change in the graphs on subsequent runs.

Siemens Dataset Generated by ignoring the abnormal test cases only for that cycle only

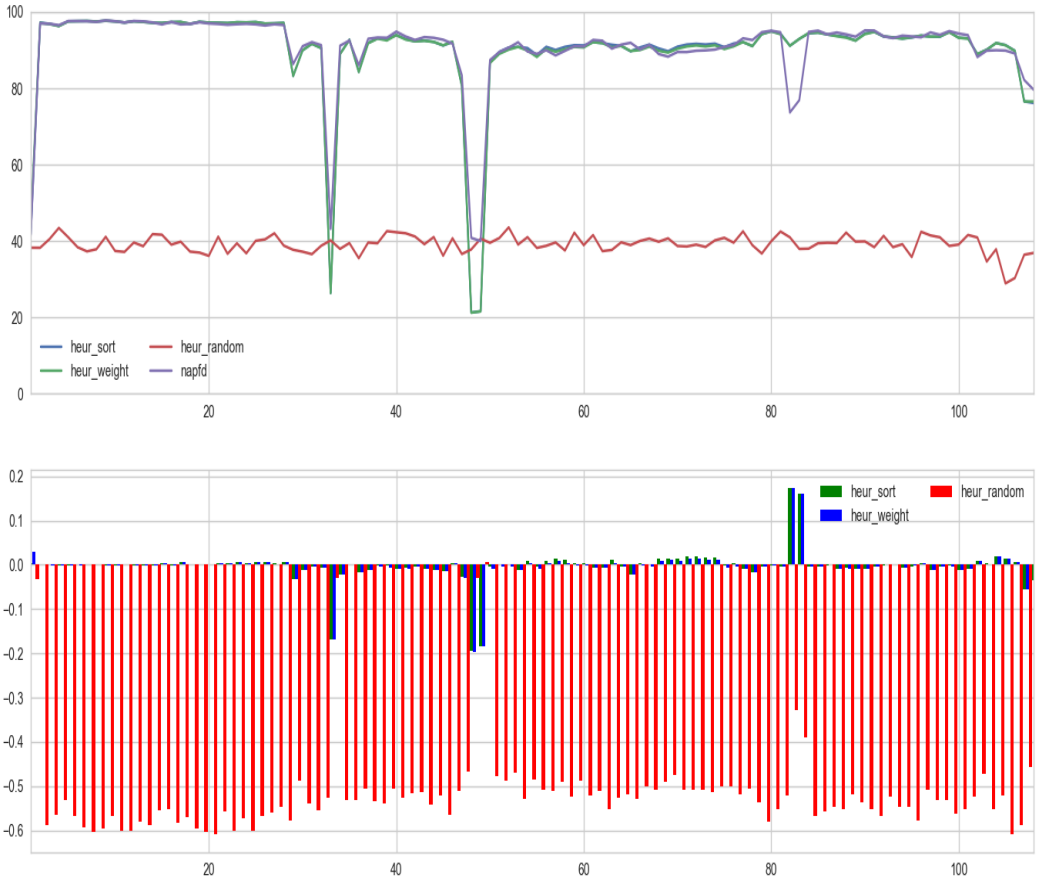


Evaluation Metric as Recall

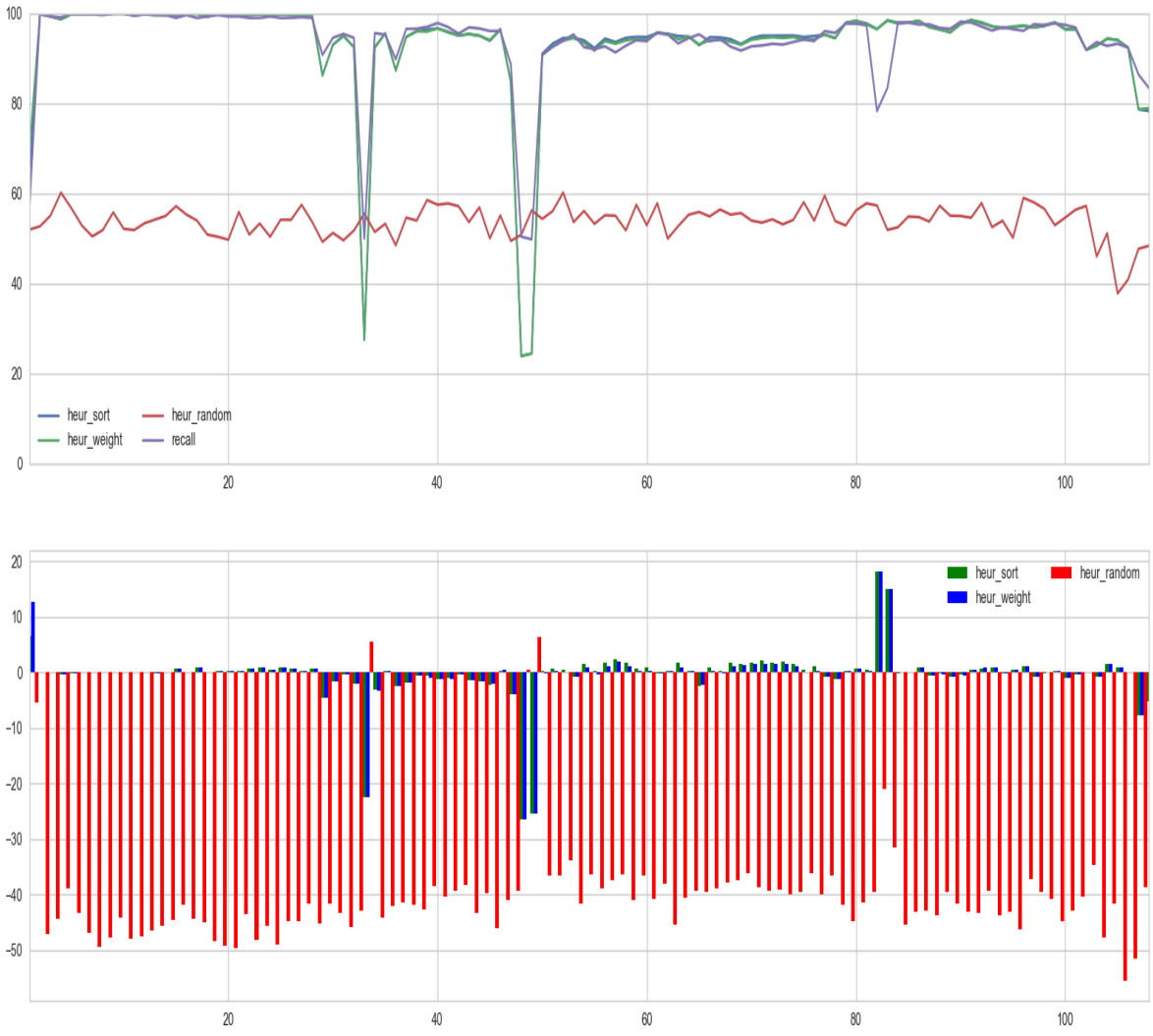


Evaluation Metric as NAPFD

Hadoop Project:



Evaluation Metric as NAPFD



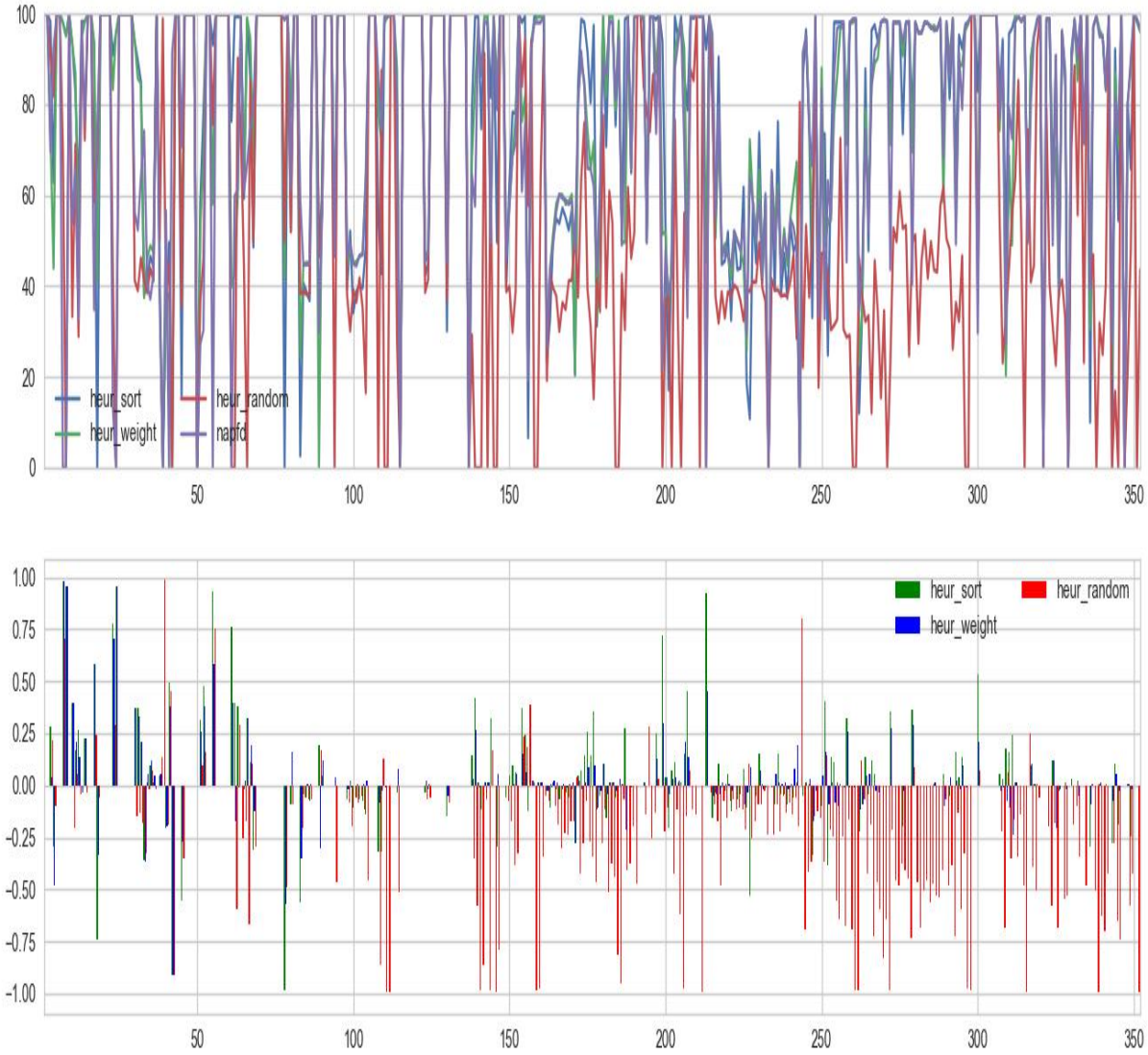
Evaluation Metric as Recall

For 50% as the available time:

Recall = 88.47%

NAPFD = 0.906

ABB Dataset:



Evaluation Metric as NAPFD

Future Improvements:

1. The priority of the test cases can be included (already test checked by giving random priorities) and depending upon the requirements and the use case, either the weights could be modified or we can use a continuous function.

2. We can include some domain knowledge to make the algorithm more informed. If we know the relation between the test cases and the functionalities they are checking, we can incorporate this knowledge and prioritize accordingly so that we end up checking all functionalities rather than scheduling all the failing test cases pertaining to the same failing function.

3. Other feature vectors may also be added for representing the test cases. These may include the priority, the test type etc.

4. If the knowledge of the code coverage by the test cases is also known, it could be also used as additional information and the algorithm can be modified to make use of this knowledge also.

5. In the ANN memory representation, the weights of the Neural Nets are updated after 5 iterations. The frequency of updating can be increased or decreased depending upon the dynamic nature of the testing code.

6. Each failed test cases indicates a different failure in the system under test. This is not always true. One test case can fail due to multiple failures in the system and one failure can lead to multiple failing test cases. This information is not easily available. Nevertheless, this approach tries to find all failing test cases and thereby indirectly also all detectable failures. To address the threat, if possible, include failure causes as input features in future work. (Research Paper)

7. A simple MLP Regressor was used. However ,with extended available data amounts, an extension towards larger networks and deep learning techniques can also help in improving the efficiency.