CS547 – ADVANCED TOPICS IN SOFTWARE ENGINEERING

ASSIGNMENT - 1

Group: CS547Assignment1Group22

Members:

1. HARSHAN RETHINAVELU SELVAKUMAR – 202480548 (Contribution – 50%)

2. MANOJ KUMAR DHARMARAJ – 202468855 (Contribution – 50%)

1 Introduction

To minimize the costs associated with regression testing, software testers can prioritize their test cases based on specific criteria. A key objective of this prioritization is to enhance the fault detection rate of the test suite [5]. This analysis explores two optimization algorithms – Hill Climbing (HC) and Genetic Algorithm (GA) – to establish test order prioritization based on the Average Percentage of Faults Detected (APFD) metric. The evaluation of results involves a comparison of APFD scores and their consistency across various iterations, offering insights into the efficiency of each algorithm in optimizing fault detection.

2 Fitness function

This fitness function aims to determine the performance of each test order through the application of Average Percentage Faults Detected (APFD) measure. The formula that represents APFD is outlined as follows:

$$APFD = 1 - \left(\frac{Total\ positions\ detected}{Number\ of\ Tests\ \times Number\ of\ Faults}\right) + \left(\frac{1}{2\ \times Number\ of\ Tests}\right)$$

In the code, the fit function evaluates a given test order by examining the sequence of fault detections. The intention is to prioritize test orders that reveals faults at an earlier point, which ultimately leads to an improved APFD score. In the context of multi-objective optimization, where the goal is to minimize the testing time while simultaneously balancing multiple factors, the APFD can be utilized within the fitness function to facilitate early fault detection during the test execution.

3 Hill Climbing Algorithm

The Hill Climbing method is an optimization strategy that builds a search path within the search space, continuing until it reaches a local optima [1]. Here, hill climbing seeks to enhance a single solution by exploring its neighborhood, which are slight modifications of the existing solution. It selects the best neighbor if it results in improvement of Average Percentage of Faults Detected (APFD) scores.

3.1 Implementation

3.1.1 Initialization

The Hill Climbing algorithm initiates with a test order that is generated randomly, which is subsequently assessed through the APFD based fitness function. Following the evaluation, the algorithm proceeds to iterate, aiming to seek an improved test order by exploring neighboring solutions.

3.1.2 Choice of Neighborhood

In this approach, a neighboring solution is created by exchanging two randomly chosen elements within the current test order. Such a basic swap operation maintains the validity of neighboring solutions as permutations of the original test order, thereby allowing for slight variations in the test orders.

The neighbor function produces a new solution by duplicating the existing test order, randomly choosing two indices, and interchanging their positions. This definition of neighborhood through swapping is computationally efficient and allows the Hill Climb algorithm to explore the search space in stepwise manner.

3.1.3 Iterative Enhancement

In each iteration of the HC algorithm, the fitness of neighboring solution, measured by the APFD is analyzed. In instances where a neighboring solution demonstrates a higher APFD than the current one, it will take the place of the current solution. This procedure continues for a specified number of iterations, leading the algorithm towards a locally optimal solution.

4 Genetic Algorithm

Genetic Algorithms (GAs) are widely utilized optimization methods that function through three fundamental genetic operators: selection, crossover and mutation [3]. Genetic Algorithm (GA) functions by utilizing a set of chromosomes, referred to as population. The population is typically initialized in random. Genetic Algorithms employ two primary operators to create new solutions from the existing ones: crossover and mutation. The crossover operator is a crucial operator of GA. This process typically involves the merging of two parent chromosomes to create new chromosomes called offspring. The mutation operator introduces random changes to the characteristics of chromosomes. Typically, mutation is implemented at gene level [2].

4.1 Choice of Representation

In this implementation, each individual within the GA corresponds to a test order, represented as a list of indices that relate to specific tests. Individuals are defined as list of integers, where each integer signifies a test index. The DEAP library supports this representation by defining individuals with a specific fitness type (FitnessMax), which directs our GA towards maximizing the APFD.

4.2 DEAP Toolbox function registration

In the implementation, the DEAP library is utilized to incorporate the necessary functions into a toolbox (tool). Each function is associated with a specific genetic operation within the GA:

- **Individual Creation:** Registers a function to generate unique GA Individual.
- **Population Initialization:** Initializes the population as a list of individuals.
- **Evaluation:** Defines fitness function to evaluate the effectiveness of each test order.
- **Selection:** Registers tournament selection with a size of three.
- **Crossover:** Registers ordered crossover.
- **Mutation:** Registers with shuffle mutation.

4.3 Selection

The mechanism employed for selection is known as tournament selection. During each generation, individuals are divided into small groups, termed tournaments, each containing three individuals. From these groups, only the individual with the highest fitness score in each tournament is chosen to proceed, allowing for potential crossover and mutation. This selection method prioritizes the advancements of high-fitness individuals.

4.4 Crossover

The primary aim of crossover is to facilitate the exchange of genes, thereby ensuring that the offspring receive a combination of genes from their parents[3]. The genetic algorithm utilizes ordered crossover, a method that is particularly effective for representations based on permutations, including the sequence of test orders. Ordered Crossover involves the following steps [4]:

- A segment is randomly chosen from the first parent.
- This chosen segment is then placed into the offspring.
- The remaining positions in the offspring are subsequently filled by the elements from the second parent, ensuring that any elements already included in the chosen segment are already skipped.

4.5 Mutation

The process of mutation facilitates genetic diversity by making slight adjustments to an individual's genes. In the implementation, shuffle mutation is used with a probability of 0.05 for test index. This method operates by randomly choosing two indices within the individual and swapping their positions.

4.6 Parameters for Genetic Algorithm

The GA implemented in this analysis features a population size of 20, a crossover probability (cxpb) of 0.5, and a mutation probability (mutpb) of 0.2. It is designed to run for 50 generations, ensuring balance between computational efficiency and the quality of the solutions obtained. These parameters enable the GA to effectively explore and enhance test orders, with the goal of maximizing the APFD.

5 Evaluation

5.1 Comparison

To determine the effectiveness of the Hill Climbing (HC) and Genetic Algorithm (GA) algorithms, it is necessary to compare their results with those obtained from a random baseline. This baseline is created by producing several random test orders and calculating the APFD for each. Such a comparison reveals the improvements made by the HC and GA algorithms in relation to random ordering.

5.2 Metrics

Both algorithms, HC and GA, were run ten times on the same test matrix. The metrics obtained from these runs were subsequently recorded:

- **APFD Mean:** Analyzing the mean APFD across several runs offers valuable information regarding the standard effectiveness of each algorithm in the context of test prioritization. A higher mean APFD value denotes improved performance indicating that faults are identified earlier within the testing sequence on average.
- **APFD Standard Deviation:** The standard deviation associated with APFD is indicative of the consistency of the results obtained. A smaller standard deviation in APFD scores suggests a higher degree of consistency in performance across different runs.

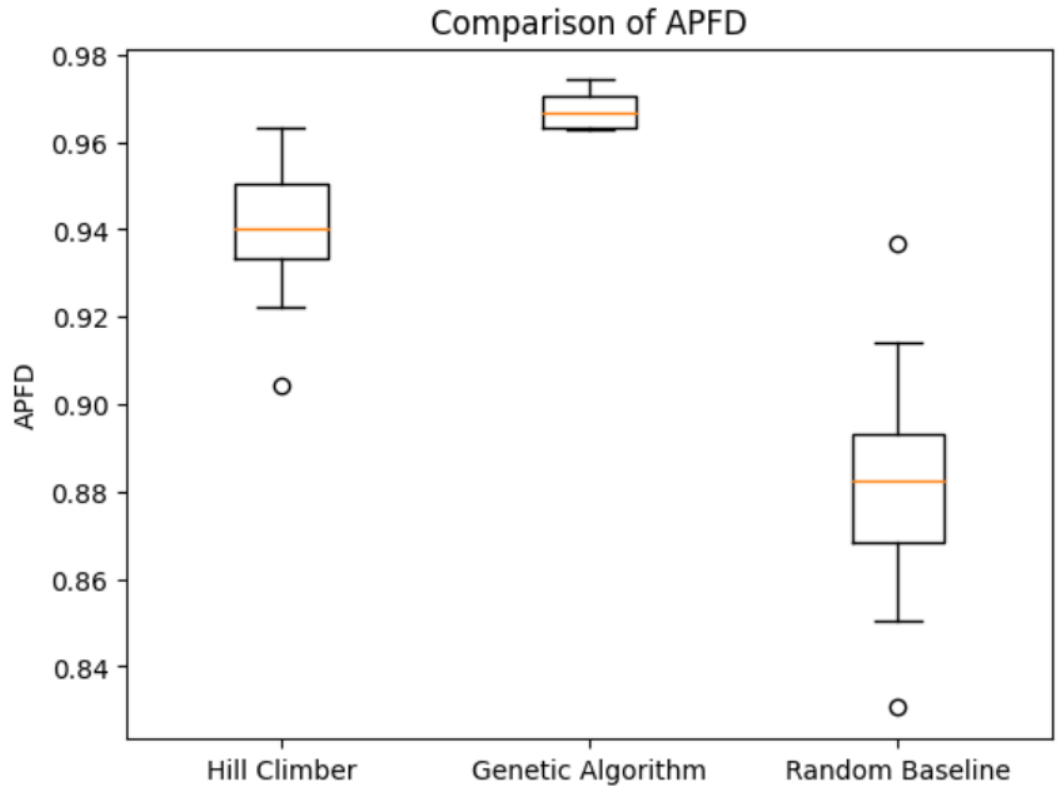
5.3 Results and Analysis

The analysis of the results was conducted by evaluating the APFD scores obtained from all runs, by comparing HC, GA and random baseline solutions. To facilitate visual comparison of the APFD scores, box plots were created.

The mean (APFD) and standard deviation (APFD) of HC, GA and random baseline are represented in the table below:

Algorithm	**Mean (APFD)**	**Standard Deviation (APFD)**
Hill Climbing (HC)	0.9400847104568808	0.0169169011110809
Genetic Algorithm (GA)	0.9673725871406749	0.004250912824510742
Random Baseline	0.8820441605332594	0.028641595231804626

5.3.1 Box Plot



The box plot illustrates the APFD scores across three algorithms: Hill climbing, Genetic Algorithm and Random Baseline.

For the Hill Climbing algorithm, the APFD is relatively high at approximately 0.94, with a moderate variability and a few outliers falling below 0.90, which indicates occasional reduced performance.

In contrast, the Genetic Algorithm exhibits the highest APFD, nearing 0.97, characterized by minimal variability and the absence of significant outliers. This reflects a stable and reliable performance.

However, the Random Baseline reveals the lowest APFD at around 0.88 with considerable variability and multiple outliers, suggesting a lack of consistency and generally poor performance.

This analysis highlights that the Genetic Algorithm is the most effective and superior for optimizing and maximizing the APFD scores.

References

[1] Al-Betar, M.A. (2016). $$\beta$$ β -Hill climbing: an exploratory local search. Neural Computing and Applications, 28(S1), pp.153–168. doi:https://doi.org/10.1007/s00521-016-2328-2.

[2] Konak, A., Coit, D.W. and Smith, A.E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety, 91(9), pp.992–1007. doi:https://doi.org/10.1016/j.ress.2005.11.018.

[3] Zainuddin, F.A., Abd Samad, M.F. and Tunggal, D., 2020. A review of crossover methods and problem representation of genetic algorithm in recent engineering applications. International Journal of Advanced Science and Technology, 29(6s), pp.759-769.

[4] Ono, I., Yamamura, M. and Kobayashi, S. (2002). A genetic algorithm for job-shop scheduling problems using job-based order crossover. doi:https://doi.org/10.1109/icec.1996.542658.

[5] Elbaum, S., Malishevsky, A.G. and Rothermel, G. (2002). Test case prioritization: a family of empirical studies. IEEE Transactions on Software Engineering, [online] 28(2), pp.159–182. doi:https://doi.org/10.1109/32.988497.