CS547 – ADVANCED TOPICS IN SOFTWARE ENGINEERING

ASSIGNMENT – 1

PART -2

Group: CS547Assignment1Part2Group47

Members:

1. HARSHAN RETHINAVELU SELVAKUMAR – 202480548 (Contribution – 50%)
2. MANOJ KUMAR DHARMARAJ – 202468855 (Contribution – 50%)

1 Genetic Algorithm (GA) Implementation:

1.1 Choice of Representation

The Genetic Algorithm (GA) uses a binary representation for its individuals, with each bit indicating whether a test case is selected or not (1 or 0). This binary representation is particularly effective for the task of choosing test cases while maximizing the fault detection and minimizing the execution time.

1.2 Fitness Function

The fitness function acts as a multi-objective optimization function. The fitness function is structured to satisfy two specific objectives:

1. Maximizing faults detected
2. Minimizing execution time

1.3 Crossover

The algorithm uses uniform crossover. It is a type of crossover operation for the generation of offspring from parent chromosomes. In this, the genetic data of the child is independently selected from each of the two parents (Immanuel and Chakraborty., 2019).

1.4 Mutation

Bit-flip mutation is used in the implementation of this algorithm.

1.5 Algorithm Implementation and framework

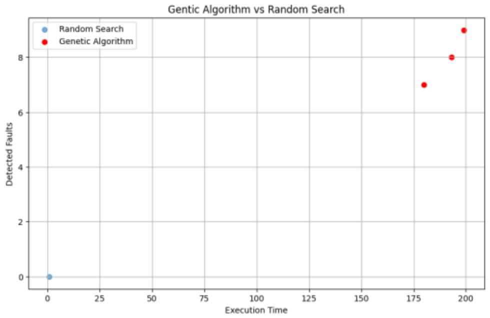
DEAP: DEAP is used. DEAP is a novel evolutionary computation framework for rapid protyping and testing of ideas [2].

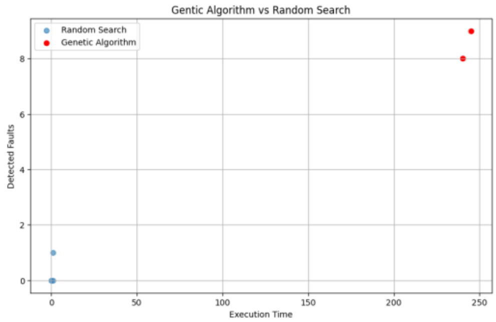
NSGA-II: NSGA-II is used for performing the selection. NSGA-II is designed for multi-objective optimization that uses non-dominated sorting to minimize computational complexity (Deb et al., 2002).

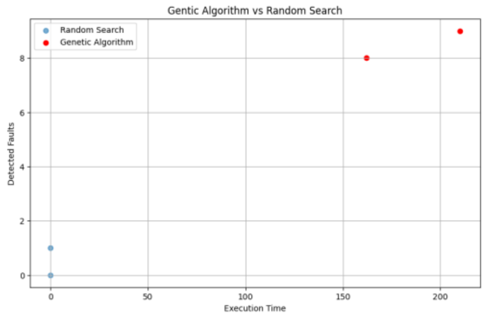
2 Evaluation

The Genetic Algorithm (GA) is executed using three distinct sets of parameters (population size, generations, crossover probability, mutation probability) to generate a variety of solutions.

2.1 Smallfaultmatrixplustime dataset





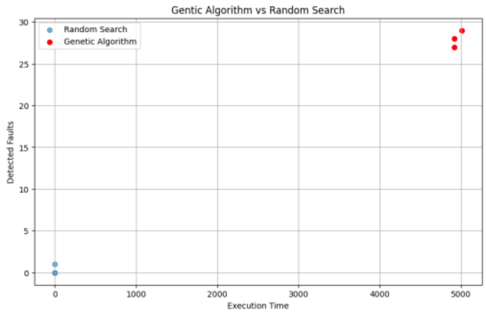


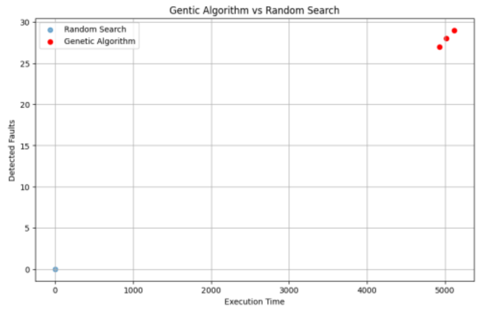
The performance of GA is effective as the Pareto Front produce solutions detecting 7 to 9 faults with execution time ranging from 180 to 250. The GA consistently generates solutions of high quality, effectively managing the trade-off between fault detection and execution time. On the other hand, random search is ineffective in yielding significant solutions, as majority of solutions tend to concentrate around the origin, detecting fewer faults and negligible execution times.

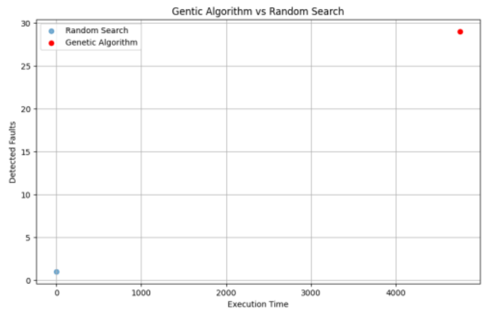
Inference:

- GA significantly surpass random search by identifying a wide range of high quality solutions.
- The Pareto front generated by GA illustrates its efficiency in managing the trade-off between the two objectives of the problem effectively.

2.2 Bigfaultmatrixplustime dataset







The performance of GA on this dataset demonstrates its scalability and effectiveness with Pareto front including solutions identifying between 27 and 29 faults, with execution times varying from 4750 to 5100. The GA effectively explores the large search space and discovers optimal solutions, showing its robustness in managing increased complexity of the dataset. In contrast, random search exhibits considerably poor performance, struggling to identify significant faults or attain acceptable execution time. This highlights its inability to handle large and complex datasets.

Inference:

- The GA effectively adjust to large problem scales while sustaining their performance levels due to its robustness.
- Random search fails to handle large, complex problem datasets.

Average of faults detected, Average execution time, Max of faults detected and Minimum Execution time are also computed and displayed in the output.

References

[1] Immanuel, S.D. and Chakraborty, U.Kr. (2019). *Genetic Algorithm: An Approach on Optimization*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICCES45898.2019.9002372.

[2] Readthedocs.io. (2019). *DEAP documentation — DEAP 1.3.0 documentation*. [online] Available at: https://deap.readthedocs.io/en/master/.

[3] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2), pp.182–197.