Exploration of Adaptive Random Test Replication Technology for Numerical Programs

Zeran Bao , Undergraduate Student Member, Xi 'an Jiaotong University

Abstract—**As a highly effective method for generating test cases, adaptive random testing is widely utilized across various fields, including fuzzing and AI testing. Among the numerous functional testing approaches, random testing serves as the fundamental method. It involves the random selection of test cases from the input domain until a specific condition is met, such as identifying errors within a software system. However, due to its limited utilization of additional information, the effectiveness of random testing is constrained. Consequently, adaptive random testing has been proposed to ensure the randomness of test cases and their even distribution throughout the entire input domain. This project focuses on numerical programs with an aim to replicate existing classical adaptive random testing algorithms and compare their efficacy. After studying the source code of the framework and conducting a thorough review of relevant academic literature, our team incorporated their own insights into the process of reconstructing the work of predecessors. Subsequently, we independently developed a comprehensive framework that facilitated customized data transmission, test case generation and execution, as well as evaluation procedures. Additionally, we utilized echarts to generate visually intuitive charts on the front-end.**

***Index Terms*—Adaptive random test, replication technology, numerical programs , frontend and backend development**

I. Introduction

Software testing is a crucial component of the software development life cycle, ensuring that the system adheres to specifications and minimizes errors. As software complexity and version iterations increase, maintaining quality and reliability in a time-efficient and cost-effective manner becomes paramount, particularly in large, fast-paced companies that adopt Continuous Integration (CI) strategies. CI facilitates early detection of system defects, provides developers with rapid feedback on code quality, shortens the software development cycle, and enhances product quality. Regression testing is essential for managing software changes and has become impractical to re-run entirely for large-scale industrial systems due to the prevalence of continuous integration. To reduce regression testing costs and improve efficiency, various test case optimization techniques have been proposed.

Regression testing, as one of the tools for managing software changes, becomes the most important part of practical software testing. And with the prevalence of continuous integration, it becomes impractical to re-run the entire test suite for large-scale industrial systems. In order to reduce the cost of regression testing and improve the efficiency of regression testing, a variety of test case optimization techniques have been proposed, such as test case identification and repair, test suite reduction, test suite expansion, test case selection and test case prioritization.

Numerical program orientation in Adaptive Random Testing (ART) ensures that test cases are primarily focused on numerical programs, allowing for efficient and adaptive generation of evenly scattered test cases throughout the input domain. Nevertheless, ART applied to various fields has different points. Numerical program orientation means that the test cases we are primarily interested in should be related to numerical programs.

Random testing is highly efficient in generating test cases; however, it has a fatal shortcoming: limited utilization of additional information beyond the given data, which can act as a constraint when testing diverse types of software. In contrast, Adaptive Random Testing (ART) ensures the randomness and even distribution of test cases throughout the entire input domain, making it "adaptive". ART has gained industry recognition for combining the strengths of random testing while mitigating its weaknesses. This indicates that adaptive random testing for numerical programs is a well-established yet dynamic field, offering valuable insights for further exploration.

By delving into the source code of the framework and leveraging insights from literature reviews, this paper enriches the existing knowledge base by infusing its own perspectives into the recreation process. It constructs a comprehensive framework encompassing custom data transmission, test case generation, operation, and evaluation. Additionally, it employs echarts to create visually appealing charts in the frontend, enhancing the overall presentation of the research findings.

From the aforementioned information, it is evident that adaptive random testing for numerical programs represents a well-established yet dynamic field, from which we can continue to derive substantial insights. Our team leveraged the source code of the framework and integrated findings from relevant literature to enhance our understanding. Through this process, we incorporated original perspectives while reconstructing predecessor content, ultimately developing a customized framework encompassing data transmission, test case generation and execution, as well as evaluation capabilities. Additionally, we utilized echarts to generate visually intuitive charts on the front-end.

.

II. Project Structure

A. *Overall Architecture*

The overall aechitecture of project is shown as figure 1.The frontend only interacts with the Controller and sends GET and POST requests wrapped in single and multiple request objects. Controller core functions consists of task distribution, receiving and sending requests.The primary functions of the two ART modules are as follows: The ART_Empirical module executes aspecific ART algorithm and validates the results produced by said algorithm; while the ART_Algorithm module conducts mutation testing and calculates compilation kill rate.

```
|   ProgramStarter.java
|
├──artMethod
|   |   ARTEnum.java 算法的枚举类，用来进行反射实例化
|   |
|   ├──hybrid 运动多种方式的ART
|   |   ├──DMART
|   |   └──localization
|   |
|   ├──PartitioningBasedStrategy 基于分区思想的ART
|   |   ├──bisectionPartition
|   |   ├──iterativePartition
|   |   ├──proportionalRT
|   |   ├──randomPartition
|   |   └──staticPartition
|   ├──quasiRandomStrategy
|   ├──searchBasedStrategy 基于搜索的ART
|   └──STFCS 基于候选人的ART
├──faultZone 三种典型的失败域
├──model 通用的模型
├──simulation 仿真评估的方法
├──socketEntity 与controller进行通信的方法
└──util 一些通用的方法
```

Fig. 1. Overall architecture of project

B. Frontend Architecture

The frontend adopts vue3 architecture, with utility class components folder components, modules network and router responsible for data transmission, and two independent pages views. Figure 2 reveals the brief structure of the front-end source code. The main function is page display, including ART form input information collection, ART results visualization, single or multiple ART algorithm switching.

```
FRONTIER\TCP-FRONTIER\SRC
|  App.vue
|  main.js
├─components
|      draw_charts.js
|      Layout.vue
|      test.js
├─network
|      network.js
|      request.js
├─router
|      index.js
└─views
        MultiART.vue
        SingleART.vue
```

Fig. 2. Structure of the front-end source code

C. Backend Architecture

The back end is mainly composed of three modules: Controller, ART_Algorithm and ART_Empirical.

1) **Controller**
Con troller receives the front-end data, parses the request, sends the request to ART_Empirical to generate test cases and runs the simulation test. After the test cases are generated, ART_Algorithm is wakened to test the mutation kill rate.
The structure of Controller is shown in figure 3. Spring Boot interacts with the frontend, encapsulating request objects and return objects, and stateless services. The Socket communicates with the backend and customizes the protocol interaction. For multi-TCP evaluation, the evaluation method is customized.

```
src/main/java/com/example
| entity
| service
| socket
| util
| web
└──ControllerStarter
```

Fig. 3. Structure of Controller

2) **ART_Algorithm**
ART_Empirical takes the Controller command, extracts and parses the test request body, and returns a partial evaluation result. It implements a custom evaluation framework, custom data extraction methods with common, unified data encapsulation, 15+1 ART algorithms, custom data storageThe structure of ART_Algorithm is shown in figure 4.

```
|   ProgramStarter.java
|
├──artMethod
|   |   ARTEnum.java
|   |
|   ├──hybrid
|   |   ├──DMART
|   |   └──localization
|   |
|   ├──PartitioningBasedStrategy
|   |   ├──bisectionPartition
|   |   ├──iterativePartition
|   |   ├──proportionalRT
|   |   ├──randomPartition
|   |   └──staticPartition
|   ├──quasiRandomStrategy
|   ├──searchBasedStrategy
|   └──STFCS
├──faultZone
├──model
├──simulation
├──socketEntity 与controller
└──util
```

Fig. 4. Structure of ART_Algorithm

3) **ART_Empirical**
ART_Algorithm receives Controller commands, runs ART to generate test cases, stores them, and returns partial evaluation results. It implements a custom test run and evaluation framework and custom data extraction.

```
|   ProgramStarter.java
|
├──artMethod
|   |   ARTEnum.java
|   |
|   ├──hybrid
|   |   ├──DMART
|   |   └──localization
|   |
|   ├──PartitioningBasedStrategy
|   |   ├──bisectionPartition
|   |   ├──iterativePartition
|   |   ├──proportionalRT
|   |   ├──randomPartition
|   |   └──staticPartition
|   ├──quasiRandomStrategy
|   ├──searchBasedStrategy
|   └──STFCS
├──faultZone
├──model
├──simulation
├──socketEntity 与controller
└──util
```

Fig. 4. Structure of ART_ Empirical

III. Algorithm

We successfully archieved 18 ART algorithms.Their categories, names and source paper information are as follows:

分类	算法名称	来源论文
Select-Test-From-Candidates Strategy(STFCS)	FSCS	Adaptive Random Testing, Fixed-Size-Candidate-Set (FSCS) ART
	RRT	Restricted random testing:Adaptive random testing by exclusion
	FSCS-ART-DNC	Application of a Failure Driven Test Profile in Random Testing
	RRT-DNC	Application of a Failure Driven Test Profile in Random Testing
	Inverted FSCS-ART	Fixed-Size-Candidate-Set (FSCS) ART
	Farthest-CGConstriant-ART	Adaptive Random Testing with CG Constraint
	Closest-CGConstriant-ART	Adaptive Random Testing with CG Constraint
	DM_ARRT(DM-ART-2L)	Adaptive random testing based on distribution metrics
Partitioning-Based Strategy	Static partitioning	Adaptive Random Testing by Static Partitioning
	Random partitioning	Adaptive random testing through dynamic partitioning
	Bisection partitioning	Adaptive random testing through dynamic partitioning
	Iterative Partitioning	Adaptive random testing through iterative partitioning
	Proportional Random Testing	Fixed-Size-Candidate-Set (FSCS) ART
Quasi-Random Strategy	Randomized Quasi-Random Testing	Randomized Quasi-Random Testing
Search-Base Strategy	DiversityART	Diversity oriented test data generation using metaheuristic earch techniques
Hybrid-Based Strategies	DM-ART(STFCS + PBS)	Enhancing mirror adaptive random testing through dynamic partitioning.
	DC-ART(The ART of divide and conquer)	An innovative approach to improving the efficiency of adaptive random testing
	LocalizationART(improvement of random partition)	Adaptive random testing by localization

TABLE I

Categories, names and source paper information of 18 ART algorithms

Next we choose some important algorithms and explain their principle and code implementation.

A. FSCS-ART-DNC

The flow chart of the algorithm is as follows, where the red box part is the addition of the algorithm based on FSCS.

1. Input an integer $maxTrial$, and a real number $initialR$, where $maxTrial > 0$ and $initialR > 0$.
2. Set $n = 0$, $E = \{\}$, and $reveal$ = **false**.
3. Set $\alpha_1 = \alpha_2 = \cdots = \alpha_N = 0.5$, where N denotes the dimension of I.
4. **while** (**not** $reveal$)
5. **if** ($n = 0$)
6. Randomly generate a test case t from I, according to a uniform distribution.
7. **else**
8. Set $noTrial = 0$, $R = initialR$, and $outside$ = **false**.
9. **for** each element $e_i \in E$, where $i = 1, 2, \cdots, n$
10. Determine a circular exclusion zone z_i, whose size is set as $\frac{R \cdot |I|}{|E|}$.
11. **end_for**
12. **while** (**not** $outside$)
13. Increment $noTrial$ by 1.
14. **if** ($noTrial = maxTrial$)
15. Set $noTrial = 0$ and $R = \max\{0, R - 0.1\}$.
16. **for** each element $e_i \in E$, where $i = 1, 2, \cdots, n$
17. Determine a circular exclusion zone z_i, whose size is set as $\frac{R \cdot |I|}{|E|}$.
18. **end_for**
19. **end_if**
20. Randomly generate a candidate c from I, where each coordinate c_l of c is generated based on α_l; and according to (2), and $l = 1, 2, \cdots, N$.
21. **if** ($c \notin \bigcup_{i=1}^{|E|} z_i$)
22. Set $outside$ = **true**, and $t = c$.
23. **end_if**
24. **end_while**
25. **end_if**
26. Use t to test the target program.
27. **if** (t reveals a failure)
28. Set $reveal$ = **true**.
29. **else**
30. Store t into E, and increment n by 1.
31. Calculate $p^l_{tc-boundary}$, and the s-expected value of $P^l_{can-central}$ for each coordinate according to (4), and (6), respectively, where $l = 1, 2, \cdots, N$.
32. Calculate the values of $\alpha_1, \alpha_2, \cdots, \alpha_N$ according to (5).
33. **end_if**
34. **end_while**
35. Report the failure detected, and exit.

B. RRT-DNC

The algorithm adopts the same idea as FSCS-ART-DNC, but it is implemented on RRT instead of FSCS-ART.

Input an integer $maxTrial$, and a real number $initialR$, where $maxTrial > 0$ and $initialR > 0$.
Set $n = 0$, $E = \{\}$, and $reveal$ = **false**.
Set $\alpha_1 = \alpha_2 = \cdots = \alpha_N = 0.5$, where N denotes the dimension of I.
while (**not** $reveal$)
  if ($n = 0$)
    Randomly generate a test case t from I, according to a uniform distribution.
  else
    Set $noTrial = 0$, $R = initialR$, and $outside$ = **false**.
    for each element $e_i \in E$, where $i = 1, 2, \cdots, n$
      Determine a circular exclusion zone z_i, whose size is set as $\frac{R \cdot |I|}{|E|}$.
    end_for
    while (**not** $outside$)
      Increment $noTrial$ by 1.
      if ($noTrial = maxTrial$)
        Set $noTrial = 0$ and $R = \max\{0, R - 0.1\}$.
        for each element $e_i \in E$, where $i = 1, 2, \cdots, n$
          Determine a circular exclusion zone z_i, whose size is set as $\frac{R \cdot |I|}{|E|}$.
        end_for
      end_if
      Randomly generate a candidate c from I, where each coordinate c_l of c is generated based on α_l; and according to (2), and $l = 1, 2, \cdots, N$.
      if ($c \notin \bigcup_{i=1}^{|E|} z_i$)
        Set $outside$ = **true**, and $t = c$.
      end_if
    end_while
  end_if
  Use t to test the target program.
  if (t reveals a failure)
    Set $reveal$ = **true**.
  else
    Store t into E, and increment n by 1.
    Calculate $p^l_{tc-boundary}$, and the s-expected value of $P^l_{can-central}$ for each coordinate according to (4), and (6), respectively, where $l = 1, 2, \cdots, N$.
    Calculate the values of $\alpha_1, \alpha_2, \cdots, \alpha_N$ according to (5).
  end_if
end_while
Report the failure detected, and exit.

D. *Inverted FSCS-ART*

This algorithm is an improvement of FSCS-ART, which mainly solves the problem that FSCS-ART selects more test cases from the edge region than from the center region in the high-dimensional input domain. The method provided is to reverse the edge/center distribution of FSCS-ART test cases, so as to improve the fault detection efficiency.

Function (5) maps the FSCS-ART test cases from the edge to the center region, or from the center to the edge region.

$$f(x_i) = \begin{cases} x_i + d_i/2 & 0 < x_i \le d_i/2 \\ x_i - d_i/2 & d_i/2 < x_i \le d_i \end{cases} \quad (5)$$

1. Initialize E as an empty set.
2. Randomly choose a test case t. Add t to E.
3. Test the SUT using test case t.
4. If a failure is detected, testing is stopped and debugging may start. Otherwise go to step 5.
5. Randomly generate k candidates from the input domain to form a candidate set C, where k is a constant integer greater than 0. The value of k was set to 10 in our experiments.
6. Find $c \in C$ such that, among all the elements in C, c has the longest distance to its nearest neighbor in E.
7. Add c to E.
8. Map c to c' using Equation (5).
9. Test the SUT using test case c'.
10. If testing resources are not exhausted, go to step 4.

E. *Proportional Random Testing*

1. Initialize j to 0, *edgeCount* to 1, *centreCount* to 1, and *selectTestCaseInCentre* to *true*, where j is the number of test cases executed.
2. If *selectTestCaseInCentre* is *true*, randomly select and execute a test case in the centre region. Otherwise, randomly select and execute a test case in the edge region.
3. Increment j by 1. If *selectTestCaseInCentre* is *true*, increment *centreCount* by 1. Otherwise, increment *edgeCount* by 1. Update $R_{E,C}$.
4. If a failure is detected, testing is stopped and debugging may start. Otherwise go to step 5.
5. Update the projected failure rate as $\Theta_{projected} = 1/(j+1)$.
6. Calculate the ratio P_{centre}/P_{edge} based on $\Theta_{projected}$.
7. If $1/(R_{E,C}) < P_{centre}/P_{edge}$, set *selectTestCaseInCentre=true*. Otherwise, set *selectTestCaseInCentre=false*.
8. If testing resources are not exhausted, go to step 2.

IV. Experiment

The code provides a Main function that the user can interact with, allowing the user to select one or more ART algorithms to test.

```
How many ART algorithms do you want to test?
1.One 2.All
1
Choose the category of the ART algorithm you want to test:
1. Select-Test-From-Candidates Strategy(STFCS)
2. Partitioning-Based Strategy
3. Quasi-Random Strategy
4. Search-Based Strategy
5. Hybrid-Based Strategies
1
Choose the ART algorithm you want to test:
1. FSCS-ART
2. RRT
3. FSCS-ART-DNC
4. RRT-DNC
5. Inverted FSCS-ART
6. Farthest-CGConstraint-ART
7. Closest-CGConstraint-ART
8. DM-ARRT(DM-ART-2L)
1
===========================================================
artMethod.STFCS.fscs.FSCS_art

Dimension = 2   failure-rate = 0.01   faultZoneType = Block
    The average of Fm=: 67.063  Fart/Frt= 0.6706300000000001   Running Time= 2.374
===========================================================
```

Fig. 4. Running screenshots

A. Test Mode and Parameters

We use simulation tests, and for each ART method, there will be 2 input domain dimensions, 4 failure rates, and 3 failure domain types. So for each method, we get 24 trials. We run each 1000 times and compute the average F-measure, F-art/F-rt, as well as the running time.

实验参数	取值（集合）
输入域维度	{2，3}
失败率	{0.01, 0.005, 0.002, 0.001}
失败域类型	{Block, Strip, Point}

B. Test Results

The results are stored in 'ARTEmpirical-main/result', which contains a txt file with details for each method on 2D and 3D input fields, 0.001 failure rate, three failure field types, and an 'ART_Result_Summary.csv' file with all the results. In the 'ART_Result_Summary.csv' file, there are 24 pieces of data about each ART algorithm, for example for the DMART algorithm:

1	Algorithm	Dimensior	FailRate	FaultZone	Fm	Fr	RunningTime(
2	DMART	2	0.01	Block	91.37267	0.913727	0.348
3	DMART	2	0.01	Strip	98.228	0.98228	0.288
4	DMART	2	0.01	Point	97.73833	0.977383	0.326333
5	DMART	2	0.005	Block	154.662	0.77331	0.468
6	DMART	2	0.005	Strip	191.7297	0.958648	0.747
7	DMART	2	0.005	Point	197.9363	0.989682	0.783
8	DMART	2	0.002	Block	428.3133	0.856627	3.082333
9	DMART	2	0.002	Strip	461.0687	0.922137	2.927
10	DMART	2	0.002	Point	476.0647	0.952129	4.127333
11	DMART	2	0.001	Block	889.722	0.889722	11.20167
12	DMART	2	0.001	Strip	972.1523	0.972152	15.40867
13	DMART	2	0.001	Point	971.707	0.971707	15.59533
14	DMART	3	0.01	Block	83.89067	0.838907	0.254
15	DMART	3	0.01	Strip	97.176	0.97176	0.347333
16	DMART	3	0.01	Point	96.312	0.96312	0.358333
17	DMART	3	0.005	Bplock	172.96	0.8648	0.735333
18	DMART	3	0.005	Strip	198.341	0.991705	0.957333
19	DMART	3	0.005	Point	199.545	0.997725	0.973667
20	DMART	3	0.002	Block	456.5597	0.913119	5.717
21	DMART	3	0.002	Strip	491.1463	0.982293	7.269333
22	DMART	3	0.002	Point	498.3177	0.996635	8.517333
23	DMART	3	0.001	Block	858.0827	0.858083	25.648
24	DMART	3	0.001	Strip	961.3873	0.961387	31.18533
25	DMART	3	0.001	Point	1025.479	1.025479	32.95833

Fig. 5. DMART_Result_Summary.csv

C. Comparison of Different Algorithms

Figure 6 and 7 reveals the F-measure and running time of different algorithms.

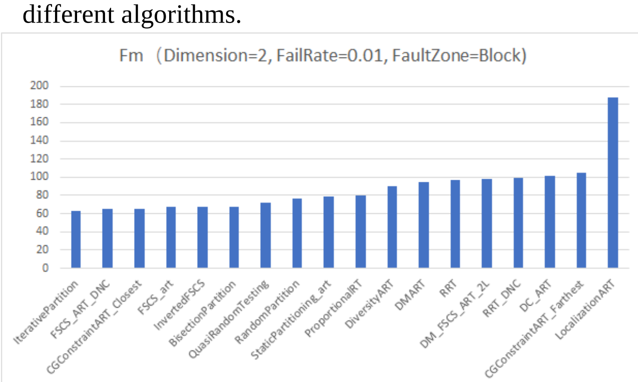


Fig. 6. F-measure of different algorithms

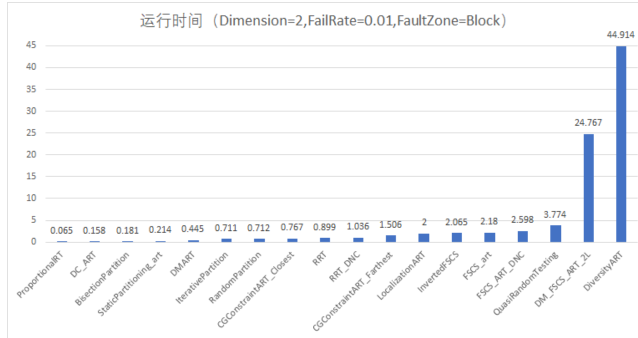


Fig. 7. Running time of different algorithms

D. Web-based running

We choose the single ART analysis as example. the algorithm is FSCS, the numerical program is Bessj, and the running time is about 15 minutes with the case of block-level failure domains.

Figure 7 and Figure 8 reveal the front-end page and back-end in the example experiment.

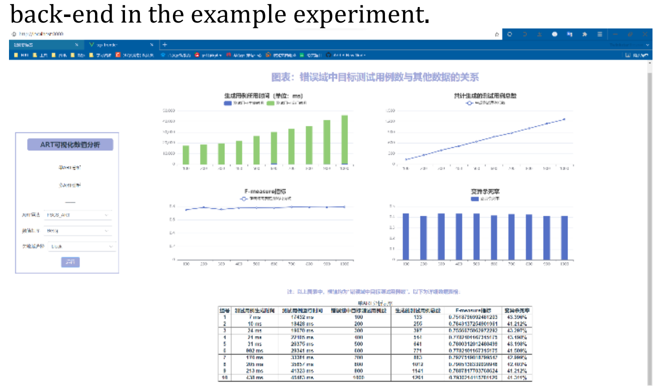
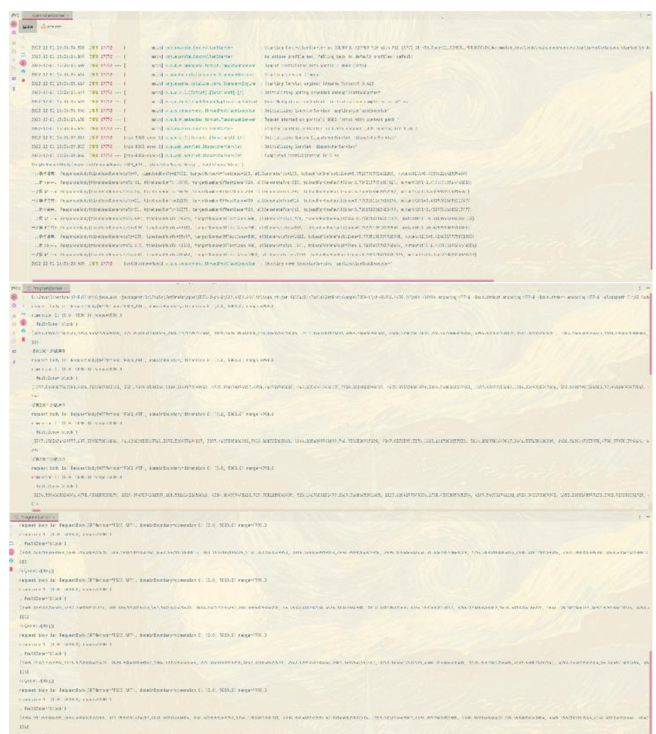


Fig. 7. Front-end page



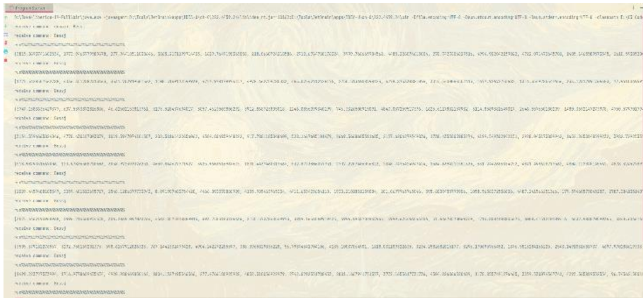


Fig. 8. backend page

V. Conclusion

Adaptive random testing emerges as a highly effective approach for test case generation, widely adopted in diverse fields such as fuzzing and AI testing. While random testing serves as a fundamental method in functional testing, its effectiveness is limited by the lack of utilization of additional information beyond the input domain. To address this constraint, adaptive random testing has been introduced to ensure the randomness and uniform distribution of test cases throughout the input domain.

In this article, we specifically focused on numerical programs, aiming to replicate classical adaptive random testing algorithms and assess their effectiveness. By examining the source code of the framework and conducting a comprehensive review of relevant academic literature, our team integrated their unique perspectives into the process of reconstructing the work of previous researchers. Subsequently, we developed a comprehensive framework that enabled customized data transmission, test case generation, execution, and evaluation processes. Furthermore, we leveraged echarts to create visually intuitive charts on the front-end, enhancing the presentation of our research findings.

References

[1] M. Abdelkarim and R. ElAdawi, TCP-Net: Test ase Prioritization using End-to-End Deep Neural Networks//2022 IEEE International

[2] Conference on Software Testing, Verification and Validation Workshops (ICSTW). Valencia, Spain, 2022: pp. 122-129.Antonia Bertolino, Antonio Guerriero, BrenoMranda, Roberto Pietrantuono, and Stefano Russo. Learning-to-rank vs ranking-to-learn:trategies for regression testing in continuous integration//Proceedings of theACM/IEEE 42nd International ew York, NY, USA. 2020. 1–12.

[3] Benjamin Busjaeger and Tao Xie. Learning for test prioritization: an industrial case study//Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016). Association for Computing Machinery.ew York, NY, USA, 2016: 975–980.

[4] J. Chen, Y. Bai, D. Hao, Y. Xiong, H. Zhang nd B. Xie, Learning to Prioritize Test Programs for Compiler Testing//2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). Buenos Aires, Argentina, 2017: pp. 700-711.

[5] E. A. Da Roza, J. A. P. Lima, R. C. Silva and S.Vergilio, Machine Learning Regression Techniques for Test Case Prioritization in Continuous Integration Environment//2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Honolulu, HI, USA, 2022: pp. 196-206.

[6] D. Di Nardo, N. Alshahwan, L. Briand and Y.Labiche, Coverage-Based Test Case Prioritisation:n Industrial Case Study//2013 IEEE Sixth International Conference on Software Testing,Verification and Validation. Luxembourg, 2013: pp.02-311.

[7] D. Di Nardo, N. Alshahwan, L. Briand and Y.Labiche, Coverage-Based Test Case Prioritisation:n Industrial Case Study//2013 IEEE Sixth International Conference on Software Testing,Verification and Validation. Luxembourg, 2013: pp.02-311.

[8] J. A. P. Lima and S. R. Vergilio, A Multi-Armed Bandit Approach for Test Case Prioritization in Continuous Integration Environments. IEEE Transactions on Software Engineering, 2022. vol. 48,no. 2: pp. 453-465, 1.

[9] V. H. S. Durelli et al., Machine Learning Appliedto Software Testing: A Systematic Mapping Study.EEE Transactions on Reliability, 2019. vol. 68, no., pp. 1189-1212.

[10] Sebastian Elbaum, Alexey Malishevsky, and GreggRothermel. Incorporating varying test costs and faultseverities into test case prioritization//Proceeding of he 23rd International Conference on SoftwareEngineering(ICSE). Toronto, ON, Canada. 2001: pages 329–338.

[11] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan,Chunrong Fang, and Zhenyu Chen.DeepGini:prioritizing massive tests to enhance the robustnessof deep neural networks//Proceedings of the29th ACM SIGSOFT International Symposiumon Software Testing and Analysis (ISSTA 2020).Association for Computing Machinery. New York,NY, USA, 2020: 177–188.

[12] Y. Huang, T. Shu and Z. Ding, A Learn-to-Rank Method for Model-Based Regression Test CasePrioritization. IEEE Access, 2021,vol. 9, pp. 16365-16382.

[13] Jahan, Hosney et al. Version Specific Test CasePrioritization Approach Based on Artificial Neural Network, Intelligent Fuzzy Systems, 2019,vol. 36, no. 6, pp. 6181-6194.

[14] Kandil, P., Moussa, S., and Badr, N. Cluster-based test cases prioritization and selection techniquefor agile regression testing. ournal Of Software-evolution And Process, 2017, 29: e1794.

[15] Z. Khalid and U. Qamar, Weight and Cluster BasedTest case Prioritization Technique//2019 IEEE10th Annual Information Technology, Electronicsand Mobile Communication Conference (IEMCON).Vancouver, BC, Canada, 2019: pp. 1013-1022.

[16] R. Lachmann, S. Schulze, M. Nieke, C. Seidl and I.Schaefer, System-Level Test Case PrioritizationUsing Machine Learning// 2016 15th IEEEInternational Conference on Machine Learningand Applications (ICMLA). Anaheim, CA, USA,2016: pp. 361-368.

[17] Jackson A. Prado Lima, Willian D. F. Mendonça,Silvia R. Vergilio, and Wesley K. G. Assunção.Learning-based prioritization of test cases incontinuous integration of highly-configurablesoftware//Proceedings of the 24th ACM Conferenceon Systems and Software Product Line: Volume A- Volume A (SPLC ’20). Association for ComputingMachinery. New York, NY, USA, 2020: Article 31,1–11.

[18] C. -T. Lin, S. -H. Yuan and J. Intasara, A Learning-to-Rank Based Approach for Improving RegressionTest Case Prioritization//2021 28th Asia-PacificSoftware Engineering Conference (APSEC). Taipei,China. 2021: pp. 576-577.

[19] A. Da Roza, J. A. P. Lima, R. C. Silva and S.Vergilio, Machine Learning Regression Techniques for Test Case Prioritization in Continuous Integration Environment//2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Honolulu, HI, USA, 2022: pp. 196-206.

[20] Mahdieh M, Mirian-Hosseinabadi S H, EtemadiK, et al. Incorporating fault-proneness estimationsinto coverage-based test case prioritization methods.Information and Software Technology, 2020, 121:106269.

[21] N. Medhat, S. M. Moussa, N. L. Badr and M. F.Tolba, A Framework for Continuous Regression andIntegration Testing in IoT Systems Based on DeepLearning and Search-Based Techniques. IEEE Access,2020, vol. 8, pp. 215716-215726.

[22] Francis Palma, Tamer Abdou, Ayse Bener, JohnMaidens, and Stella Liu. An Improvement to TestCase Failure Prediction in the Context of Test CasePrioritization//Proceedings of the 14th InternationalConference on Predictive Models and Data Analyticsin Software Engineering (PROMISE’18). Associationfor Computing Machinery. New York, NY, USA,2018: 80–89.

[23] Pan, R., Bagherzadeh, M., Ghaleb, T.A. et al. Test case selection and prioritization using machine learning: a systematic literature review. EmpiricalSoftware Engineering. 2022. 27, 29.