A Condition Coverage-Based Black Hole Inspired Meta-Heuristic for Test Data Generation

Derya Yeliz Ulutaş [1,2], Ayşe Tosun [2]

[1] *ASELSAN Inc., Ankara, Turkey*
[2] *Istanbul University, Faculty of Computer and Informatics Engineering, Istanbul, Turkey*

Abstract

Combinatorial Testing (CT) strategy is one of the well-known methods to achieve high code coverage rates during testing for safety critical systems. While generating test data in CT, we often encounter the problem of test case explosion, especially for the systems with multiple parameters and values. To overcome this challenge, search-based CT (CSST) strategies are introduced. In this study, we propose a new algorithm that inspires from a binary variant of Black Hole Algorithm (BBH) in CSST and adopt BBH according to the CT challenges in an industrial context. The proposed BBH version, *BH-AllStar*, aims the following: (1) obtaining higher condition coverage, (2) avoiding being stuck in local minima and (3) handling discrete input values. We finalize the solution space of *BH-AllStar* by reassessing the previously removed stars and incorporating the useful ones into it. We evaluate our approach on a real-life software project in the safety-critical domain with respect to condition coverage, number of test cases and execution time. Compared to BBH, *BH-AllStar* generates more test cases which achieve up to 43% increase in condition coverage.

Keywords

Combinatorial Search-based Software Testing (CSST), Test Data Generation, Meta-heuristic

1. Introduction

The complexity and the criticality of the safety critical systems in the defense industry are growing [1, 2], hence, these systems require more thorough and efficient testing technologies, such as test automation and regression test selection/prioritization [3] and continuous integration. The software testing field in the defense industry needs to develop more efficient and qualified techniques to meet security-critical requirements. [1]. For instance, various test techniques such as Boundry-Value Analysis and Equivalence Partitioning are used to meet the DO-178 Standard to ensure safety-critical requirements, which has been imposed by the U.S. Federal Aviation Administration (FAA). [4, 5]. To assess the effectiveness of these methods, different coverage criteria are employed [6]. During functional testing, Combinatorial Testing (CT) methods are widely utilized especially for software systems in which there exist many parameters and conditions that take a combination of multiple input values [7, 8]. CT method such as t-way testing executes the software using test cases with the interaction of the possible input values [9]. The radar software in our industrial context, for instance, takes 12 input parameters each takes two to six values. If we would like to test this software with full input coverage, we would end up executing 63.34.24.5=1,399,680 test cases. This is an impossible goal to reach, and instead in practice, we follow t-way testing with the IPOG strategy [10]: select the t value as high as possible with the cost of increasing the number of test cases. The current limitation of the employed t-way testing in our industrial setting is the lack of relationship between the selected test cases and their code coverage, because the strategy of selecting the test inputs is not dependent on the covered branches/conditions of the software system.

QuASoQ'21: 9th International Workshop on Quantitative Approaches to Software Quality, December 06, 2021, Taipei, Taiwan
EMAIL: dyulutas@aselsan.com.tr (D.Y. Ulutas); tosunay@itu.edu.tr (A. Tosun)

To address this challenge, combinatorial search-based software testing (CSST) approaches are suitable [11]. One of the studies by Al-Sammarraie and Jawawi [9] inspired our work. The authors propose three new approaches namely Binary Black Hole (BBH), Multiple Black Hole (MBH) and Binary Multiple Black Hole (BMBH) based on the Black Hole Algorithm (BHA), which is first introduced in [12] as an alternative to Particle Swarm Optimization (PSO). Their aim was generating test inputs with respect to the number of test cases and code coverage. The authors also aim to avoid being stuck in local minima and they propose a novel energy mechanism for MBH that helps to choose useful black hole populations only. The results show that MBH outperforms PSO and BHA in terms of the number of test cases, consistency and high coverage rates.

We share the same goal with [9] of generating test data using a search-based meta-heuristic, which avoids local minima and achieves higher coverage than BBH. The approach in [9] could be well fitted in our industrial problem because of two main motivations: (1) their problem domain and data set are similar to our project, i.e., radar software, (2) they innovatively apply BHA, which is originally a clustering algorithm, to generate test data for functional testing. However, considering the scale of our radar software under test, BBH approach does not fully address our challenge. More specifically, BBH concentrates on generating minimum number of test cases rather than increasing the coverage criteria. Due to this, it lacks assessing the generated stars except the black hole in terms of their coverage. The term star here corresponds to the test case (e.g., [3, 5, 7] is an input set, also a test case and named as 'star'), as depicted in Table 2. This in turn eliminates some of the useful stars during the operations of BBH. In this study, we propose a new algorithm called BH-AllStar by modifying several strategies in BBH, such as elimination mechanism based on distance between stars, moving stars in discrete space, and selecting the best black hole population from the history. Our new approach generates relatively more test cases with higher condition coverage than BBH, although we cover only 0.14% of 1,399,680 test cases in our industrial context. We assess the applicability of BBH in [9] and its new variant BH-AllStar in an industrial context with respect to the number of test cases, coverage rate and execution time.

The remainder of this paper is organized as follows: Section 2 presents the related work. We report the details of our approach BH-AllStar and research methodology in Section 3. Section 4 shows the results and discussion; Section 5 explains the threats to validity. Finally, Section 6 concludes this study with future directions.

2. Related work

Combinatorial Search-based Software Testing (CSST) is a testing method based on Combinatorial Testing (CT). CT aims at generating input vectors combining possible input values of all input parameters of the Software Under Test (SUT) [10]. The SUT can have n input parameters: c_i (i = 1,2,3…n). Every parameter can take a set of values V_i where i can be from 1 to n. For example, V_1 is the value set of c_1 and contains m different values. (v_1,v_2,…,v_m). One test case is composed of a set of values from all V_i of all c_i (e.g. $v_x \in V_1$, $v_y \in V_2$, $v_z \in V_3$) [10]. There are different methods to build all combinations of input parameters for CT such as t-way testing or randomization-based methods in the literature [13]. CSST is one of the search-based approaches to reduce the large solution space [9]. To find the optimum solution of a problem, especially for the problems, which contain very large sets of solutions and have computational constraints, meta-heuristic search techniques are commonly implemented [14]. These meta-heuristic search techniques are based on initially generated random solution space and narrowing down the search space based on the evaluation criteria.

A recent systematic mapping study in [14] highlights 260 relevant studies in CSST, and narrows down to 42 primary studies to investigate the proposed approaches and their test case generation accuracy. The authors summarize well-known meta-heuristic search techniques as follows: Genetic Algorithm (GA), Particle Swarm Optimization Algorithm (PSO), Simulated Annealing Algorithm (SA), and Ant Colony Optimization (ACO). Their mapping strategies show that GA is the most popularly applied technique, whereas some combine multiple algorithms such as GA and SA. They conclude that the primary studies provide benefits in terms of code coverage and execution time.

In Table 1, we list a sample of studies obtained from [14] that we examine in terms of the approach, dataset on which the approaches are evaluated, and the fitness criteria. Among the

studies, we selected eight studies targeting a similar objective to our industry problem, and varying in terms of fitness functions and datasets. The studies [15-19] utilize nature inspired optimization methods for CSST, whereas [9, 12, 20] use a heuristic based on black hole phenomenon. All studies in Table 1 show that the improved versions of nature inspired optimization methods outperform original methods with respect to code coverage or execution time. All the prior studies validate the success of their proposed approach on simple functions like the source code that calculates the greatest common divisor or checking the validity of the date, etc. Unfortunately these projects are not representative of real programs, and achieving high coverage is relatively easier with few test cases. Except one study [15], all studies focus on branch coverage as one of their evaluation criteria, although in practice wıth more complex algorithms, multiple conditions need to be taken into account with a condition converage criterion. Below, we give more description on our baseline algorithm BHA [12] and its application on test case generation [9].

Table 1
Summary of studies in the literature

Study	Baseline Method	Dataset/Project	Fitness/Evaluation Criteria
[15]	ACO	triangleType, gcd, calday, isValidDate, cal	Branch coverage
[16]	SA	unknown	Condition coverage
[19]	PSO	triangleType, gcd, calday, isValidDate, cal	Branch coverage
[17]	GA	triangle classification and nextDate	Branch distance
[18]	SA	triangle classification	Branch Coverage, # of Detected Mutants/Defects
[12]	BHA	Six-benchmark dataset from ML databases	Distance between blackhole and the star
[20]	BHA	Well-known mathematical functions	
[9]	BHA	pizza ordering, smart mobile, heart disease	k number of test cases

2.1. Black hole algorithm

BHA has been first introduced in the study [12] as a new heuristic algorithm inspired by the black hole phenomenon. Based on Newton's law, scientists invented a concept about the stars with high power and strong gravitational field [21]. They name this star as "black hole". The reason of this name is that any object moving around this special star cannot escape and is absorbed into the black hole if the objects cross the Schwarzschild radius [21]. This special star is described with the adjective 'black' because light cannot be reflected and make this star invisible [21, 22]. With the inspiration of the nature of black holes, BHA is proposed for data clustering [12]. Fig.1 presents the pseudocode of BHA adopted from [9, 12].

```
Algorithm 1: Black Hole Algorithm
  Result: Solution space consists of stars
  P = initializePopulation();
  while iteration do
    P = moveStars(P);
    BH = updateFitness(P); //select the best star as black hole;
    for each star S in P do
      if star(i crosses the R)) then
        replace(P, star(i)); //destroy current star and generate new one;
      end
    end
    BH = updateFitness(P); //select the best star as black hole;
  end
```

Figure 1: BHA pseudocode adopted from [9, 12]

According to the terminology used in [9], a population with random stars is initially generated, and a fitness value is calculated for every star in this population. The best star with the best fitness value is selected as the black hole. Later, all stars in the population are moved towards the black hole as described in Eq.1. The new location of the current star x(t+1) is calculated by multiplying a random number with the difference between the BH Star and the current star (BH-currentStar), and adding the result to the previous location x(t). The current star is moved closer to the BH Star as a result of this operation. Movement of the stars towards the black hole provides a grouping process and force all the other stars to converge to the best fitness value. If a star crosses the event horizon (R) of the black hole defined in Eq.2, it is destroyed, and a new random star is generated and added into the population. The R value is calculated as the ratio of the fitness value of BH Star and sum of the fitness function outcomes of all stars in the population. This step leads to adding various stars to the population and increases the probability of convergence to the desired solution.

$$x(t+1) = x(t) + \text{rand x (BH-currentStar)} \quad (1)$$

$$R = f(BHStar) / \sum_{i=1}^{n} f(Star(i)) \quad (2)$$

The searching process stops if the stopping criterion is met. This criterion can be the maximum number of iterations or a sufficiently good fitness criterion [9, 12].

Hatamlou conducted an experiment on the six-benchmark dataset [12] and highlights two

advantages of BHA: (1) simple and easy to implement, (2) applicable to other problem domains, and (3) outperforming other clustering methods. The performance of BHA is later tested in [21] on mathematical functions to reach their minimum and maximum values. The authors in [21] compare BHA with GA and PSO, and report that BHA outperforms others and can be applied to other problem domains.

2.2. The baseline study

Based on the BHA proposed in [12], Al-Sammarraie and Jawawi [9] propose three variations of BHA for test case generation: BBH, MBH and BMBH. Terms and function names used in the study [9] are described in Table 2. For the inputs that has discrete values like binary values (0,1), *moveStars* operation (Eq. 1) does not work because it causes a shift in the continuous domain due to the multiplication of the distance with a random value. To handle this issue, a binary variant of black hole is proposed. This approach generates a random value r_d between 0 and 1. This random value is compared against the constant value, p_r (Eq. 3). If it is greater than r_d, the new location of the star is going to be equal to that of black hole. Otherwise, new location of is equal to its own old value [9].

Table 2
Terminology of BBH used in [9]

Term	Meaning
Parameter	x: can take values x_1, x_2, x_3 y: can take values $y_1, y_2, y_3, \ldots$
Star	$[x_1\ y_2\ z_3\ t_4]$
Population	$[star_1, star_2 \ldots star_n]$
updateFitness()	Calculate the coverage of every star in the population and determine the best star as the black hole.
moveStars()	Change location of stars towards the blackhole.
updateRadius()	Update the radius value of black hole.
replace()	Remove the current star and generate a new one.
R	Radius of the black hole (eq.2)

$$xi(t+1) = \begin{cases} BH(t)\ if\ r_d < pr \\ xi(t)\ otherwise \end{cases} \quad (3)$$

The authors reach consistent and high coverage by extending solution space with the help of multiple swarm approach called MBH [9]. The main idea in MBH is to generate more than one population and in turn, more than one black hole in a single iteration to avoid being stuck in local minima. At the end of the iterations, final list of populations become the solution space. The findings showed that their approach suffers from being stuck in local minima since there is no mutation operation. The authors point that the population initialization and *moveStars* operations could be improved.

Our work complements the previous study [9] in several ways: (1)We propose a condition coverage-based elimination mechanism instead of a distance-based one to keep the test cases that likely contribute to total coverage. (2) We prioritize achieving higher coverage rates than restricting the number of test cases. (3) We propose two new approaches within BHA to assess all the generated stars and black holes and to extend the selected population without being stuck in local minima. (4) We apply the BHA method to a real-life software system for test case generation.

3. Our methodology

The software under test in this study is part of a large-scale radar software developed for the defense industry. Due to its confidentiality constraints, we are not able to give the details of the algorithm or its pseudocode. We can briefly describe the complexity of our SUT: it takes 12 input parameters with different possible values, and outputs an integer value. It contains two functions with 17 if and nested if statements with up to three levels, two for loops, two switch case statements and 72 branches in total. We built a technological setup, in order to conduct our analysis on BBH and its variants including BH-AllStar. Our analysis starts by selecting the baseline algorithm as shown in Fig 2, namely BBH, to generate test cases, i.e., a black hole (BH) surrounded with stars, and our software is executed against this test case. Then the system collects the information of missed conditions using Jacoco testing tool. Iteratively, the condition coverage information is stored in a file, and used in the elimination decision mechanism. Based on the decisions, the algorithm generates new test cases (stars) and the cycle continues. The algorithm terminates after K (10-500) number of iterations.

We modify several operations in [9] and propose our approach: BH-AllStar. We use *initializePopulation()* and *updateFitness()* methods as is, but modify *moveStars()*, and created a new star elimination criterion. We also propose two new methods to select the final population both of which will be described below. Our motivation is also determining the best BH in

reaching as high coverage rates as possible, and shaping the other stars around it without being stuck in local minima. During our analysis, we noticed that the distance criteria defined in [9] removes some stars although they contribute to the coverage ratio. It also prioritizes to keep the number of test cases as few as possible. Increasing the number of test cases is preferable for our industrial setting than having low coverage rates. Fig. 3 shows the pseudocode of our approach. Our contributions are presented in the below subsections.

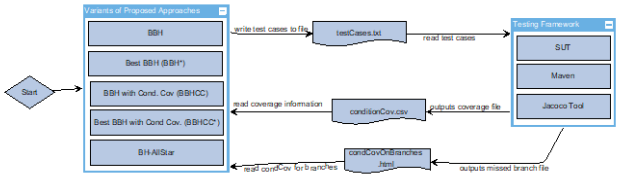

Figure 2: Our experiment setup

```
Algorithm 1: BH-AllStar
  Result: Solution space consists of stars
  P = initializePopulation();
  BH = updateFitness(P);
  while iteration do
      P = moveStars(P);
      BH = updateFitness(P);
      for each alive star S in P do
          if not isBestAtLeastOnOneBranch() then
              ratio = calculateBranchRatio();
              if ratio is less than 1 then
                  S.isAlive = false;
                  S = generateNewStar(); //generate new star different from black hole
                    P.add(S);
              end
          end
      end
      BH = updateFitness(P);
      calculateTotalCoverage();
      bestPopulation = keepPopulationWithMaxTotalCoverege();
  end
  solutionSpace = allStar(P, bestPopulation);
```

Figure 3: Pseudocode of our proposed approach

3.1. Condition coverage-based elimination mechanism

We calculate condition coverage for each branch existing in our testing code. For example, an "if" statement with one condition, e.g. x<5, contains two branches, true or false, and the condition coverage for this branch can be 0, 50% or 100%. If an "if" statement has two conditions, e.g. x<5 && y>10, then it has 4 conditions (TT, TF, FT, FF). With the input set (x=3, y=11; x=3, y=9; x=6, y=9), the condition coverage for this branch is calculated as 75%. Only traditional branch coverage is not enough for our study since we need to cover all the conditions of each branch in our testing code. Therefore, we replace the distance based mechanism, which calculates to what extent the distance between current star and black hole exceeds the R value to decide whether to destroy the star or keep it alive, with our condition coverage based mechanism, as shown in Fig.3 described as *isBestAtLeastOnOneBranch*() and *calculateBranchRatio*() methods. With this condition coverage-based mechanism for all branches of our SUT, we give chance to the stars with lower coverage ratios then the BH Star to cover uncovered branches. We have two steps in our decision mechanism:

isBestAtLeastOnOneBranch We control if the current star is the best among all stars in the population in terms of covering at least one more branch. This operation would keep an efficient star, even if it has a smaller coverage ratio compared to other stars.

calculateBranchRatio We compare current star and black hole in terms of covering all branches one by one and calculate the ratio of the number of branches that current star covers better than black hole, and the number of branches that current star covers worse than black hole. We use here a coefficient=3 to multiply the number of better covered branches in order to give a star more chance (see Eq. 4). We have made several tests with different coefficients and chosen the value 3 as the optimum value for deciding the stars to be accepted or not.

$$\frac{\#ofBetterCoveredBranches*3}{\#ofWorseCoveredBranches} \quad (4)$$

Fig. 4 illustrates a sample of stars: An eliminated star (purple), a previously eliminated but later revived star (yellow), and an always alive star (orange) with their coverage values in the timeline, when our proposed mechanism is applied. The total coverage significantly increases at the end of the iterations as we let a previously killed star (yellow) to be included into the final population. Eliminating a star (purple) also increase total coverage during 18th iteration.

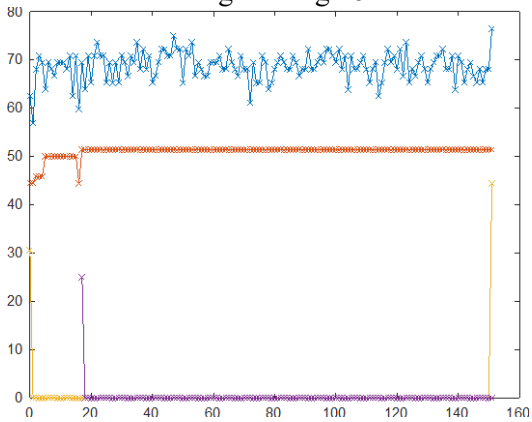

Figure 4: The x-axis is the number of iterations and the y-axis is condition coverage rate.

3.2. Moving Discretely

In *moveStars()* operation, to shift the stars towards the black hole, we propose an index-based method for discrete-valued parameters, similar to 'Binary Variants' in [9]. We store our input parameters in an array and we generate a random index value between the input value index of the current star and the BH. Then we move current star to this index and assign the value to the current star as the updated input parameter.

3.3. Choosing the best population over history

In *keepPopulationWithMaxTotalCoverage*(), we keep the population generated in all iterations of BBH including coverage rates and alive/dead states of each star. At the end of the iterations, we select the population that has the highest coverage ratio, not the last population according to BH.

3.4. All Star operation

In *allStar*() method, we compare all the stars between t_0 and tend against the BH at t_n, and add the best stars in terms of condition coverage into our final population. Although this causes an increase in the number of test cases, it also increases the coverage of the final population.

The variants of BBH are as follows: (1) Best BBH (**BBH***): We incorporate "Choosing Best Population from History" approach into BBH. (2) BBH with Condition Coverage (**BBHCC**): We use "Condition Coverage-Based Elimination Mechanism". (3) Best BBH with Condition Coverage (**BBHCC***): We add "Choosing Best Population over History" onto BBHCC. (4) **BH-AllStar**: We add "AllStar Operation" onto BBHCC*. We assess all according to condition coverage of the population, number of test cases, and execution time.

4. Results and discussion

The results over different iterations and with variants of BBH, reported in Table 3, show that higher code coverage rates are obtained compared to BBH. For instance, with 500 iterations, coverage values are 69.4% in BBH, whereas 75% in BBH*, 70.8% in BBHCC, 73.6% in BBHCC* and 76.8% in BH-AllStar. We observe that same coverage rates can be achieved with fewer test cases but higher execution time for different number of iterations. The maximum code coverage rate is approximately 76% in iterations 150 and 500 with BH-AllStar. Solution space consists of 190 test cases for 150 iterations and executes in 327 secs. However, for 500 iterations, only 19 test cases are used but execution time is 815 secs.

In addition, the baseline algorithm, BBH has a better coverage (72.2%) than our condition coverage based BBH approach (65.3%) for 300 iterations. However, our additions help to increase the coverage from 65.3% to 75% in BH-AllStar. BBH* always reaches better coverage rates than BBH. When the coverage criterion is set as condition (BBHCC), we do not always see a better convergence than BBH. But choosing the best population and adding all the "good" stars eventually converge to much higher coverage ratios. Instead of defining multiple black hole as in [9], we stick to the single black hole phenomenon but we give a certain flexibility to the star selection and achieve a similar improvement like in [9]. We can confirm the prior finding that a black hole inspired meta-heuristic could be successful at generating test data for software systems with too many parameters.

Table 3
Performance of BBH Variants and BH-Allstar

Iteration		BBH	BBH*	BBH CC	BBH CC*	BH-AllStar
10	**Stars (#)**	10	10	10	10	12
	Cov. (%)	51.4	70.8	68.1	72.2	73.6
	Time	19	19	22	22	22
150	**Stars (#)**	10	10	10	10	190
	Cov. (%)	68.1	73.5	68.7	75	76.8
	Time	320	320	372	372	372
300	**Stars (#)**	10	10	10	10	14
	Cov. (%)	72.2	75.0	65.3	73.6	75
	Time	460	460	476	476	476
500	**Stars (#)**	10	10	10	10	19
	Cov. (%)	69.4	75	70.8	73.6	76.4
	Time	782	782	815	815	815

Table 4
Coverage for different number of iterations

		BH	Alive Stars			BH-AllStar
Coverage	#iter	Star	Min	Max	Median	
	10	51.4	25	51.4	37.5	73.6
	150	51.4	23.6	51.4	43	76.8
	300	51.4	23.6	51.4	37.5	75
	500	51.4	25	51.4	38.8	76.4

Considering the number of test cases, we observe that BH-AllStar may take higher values

with respect to the positions of the stars, condition coverage ratios of each star and the black hole star. In 150 iterations, we generate 190 test cases, which is a higher value compared to the other experiment results. Since we pick the best stars in terms of coverage during *All-Star* operation, there may be many "best" stars in the population compared to the selected black hole. We did not perform any filtering on these best stars like pruning, but it might be possible to reduce those that cover similar conditions as a final step.

Table 4 presents the coverage rate of the BH Star, Alive Stars and BH-AllStar in the final solution space. In both BBH and our proposed approach, the black hole has the same coverage rate (51.4%). Alive stars in the solution space have a minimum of 23.6%, a maximum of %51.4 (BH itself) and a median of 43% coverage ratios. A star with less coverage ratio than the BH Star is acceptable in our study because it might have a higher coverage on any branch compared to the BH star.

Fig. 5 presents the coverage rates of all the branches according to 190 stars in the solution space for 150 iterations. As depicted in Fig.5, there are still some branches (e.g., 19th to 27th) which are not sufficiently covered although we increase the total coverage ratio. This is partly due to the fact that BH algorithm essentially revolves around the black star that is picked. So, some of the areas in the search space might be missed during the iterations.

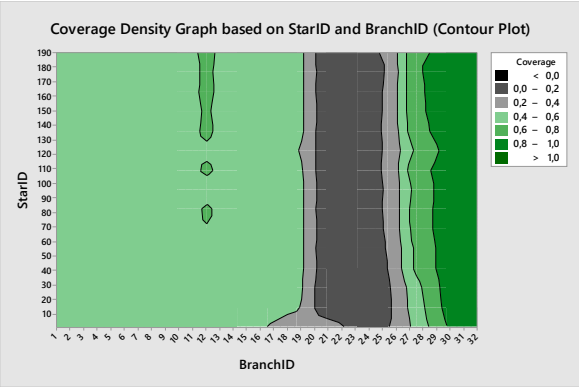

Figure 5 Coverage Density Graph based on Star ID and Branch ID

Fig. 6 illustrates the change in the total coverage ratio of BBH and BH-AllStar for 150 iterations. It is seen that one or more populations were found with BBH (around 60th and 100th iterations) that reach a higher coverage, but there were later lost. The final coverage ratio was also smaller at the end. On the other hand, in our approach, a peak can be seen just after all the iterations are executed, at t=151. The population with the best coverage has been kept and the good stars over the history are added to that population.

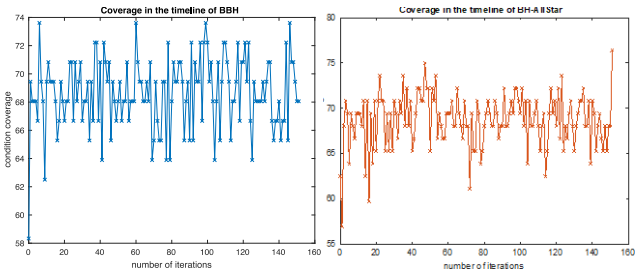

Figure 6 Coverage in the timeline of BBH

The results show that our Choosing Best Population from History approach always provides better coverage rates. Since the nature of the movement to the black hole and random generation of stars might cause a decrease in total coverage in BBH, we handle this problem by keeping the population with maximum coverage. The main reason that BH-AllStar outperforms BBH is that the latter eliminates stars because it is too close to the black hole star. However, it does not mean that these stars do not contribute to total coverage.

5. Threats to validity

We highlight several issues that might jeopardize the validity of our findings, and discuss these in this section. The first issue is related to the construct defined as the fitness criteria of our proposed algorithm. We aim to improve the final population of the algorithm by assessing the condition coverage ratio, which is calculated through Jacoco tool for each branch of the tested algorithm. Jacoco gives a condition coverage ratio for each branch, but it does not specifically say which conditions are satisfied in that branch. When two stars having 50% condition coverage for the same branch are evaluated, we consider those the same. But we do not know whether both cover different conditions and complement each other. This might have caused to falsely eliminate some stars from the population. We plan to work on this issue in our future studies by checking other unit testing tools.

The second issue is related to the binary strategy applied to turn BHA to BBH algorithm. In Section III, we describe how the movement of the stars happens on a discrete scale.

Unfortunately, the description in [9] was not detailed enough and we had to make some assumptions. This might cause a deviation between BBH in [9] and BBH we developed here. It does not affect our comparisons between BBH, its variants and BH-AllStar on the software under test in this paper.

The third issue is related to the internal validity of the experimental design. We are aware that choosing a different coefficient value in Eq.4 in BH-AllStar would impact our results, in fact, we tried multiple values and observed the convergence of the algorithm in terms of condition coverage to pick the final value as 3. We also think the randomness in the initial population, and the selection of best stars from all the iterations could affect the final results. We plan to work on these as future works as both require empirical analysis of different strategies. Finally, our results are limited to a single industrial context because we chose to solve the problem of our industrial partner through an improved CSST method. The prior works often assess their approaches on simple programming exercises, which suffer from generalization of the results on real life projects. We believe our study validates the real application of Black Hole inspired approaches adopted to condition coverage criterion.

6. Conclusion

In this study, we present a new approach to generate test data by implementing a previously proposed meta-heuristic searching technique BBH on a real-life engineering problem that we face for our safety critical systems in the industry. Our BBH variants and new algorithm BH-AllStar perform better than BBH in terms of condition coverage up to 43%. Although BH-AllStar generates more test cases compared to BBH, we achieve extending search space and obtaining higher condition coverage rates in the same execution time. As a future work, we plan new experiments to compare Multiple Black Hole Approach reported in [9] and our BH-AllStar. We also plan to decrease the number of test cases in our current approach, the selection of best stars for each branch in BH-AllStar can be limited by selection only one among the best stars. We also intend to conduct further research on every condition in a decision to be covered of the SUT, rather than having higher condition coverage, which is known as MC/DC (Modified Conditon/Decision Coverage). We think that our proposed approach can be applied to other problems, such as testing algorithms with different level of complexity, nested conditions and line of codes. Initialization of the population, which is the first step of BBH can also be optimized to start with a better population rather than a random start as in its current setting.

7. References

[1] J. Park, H. Ryu, H.-J. Choi, and D.-K. Ryu, *A survey on software test maturity in Korean defense industry*. 2008, pp. 149-150.

[2] R. Kenett, F. Ruggeri, and F. Faltin, "Analytic Methods in Systems and Software Testing," 07/01 2018, doi: 10.1002/9781119357056.

[3] V. Garousi, R. Ozkan, and A. Betin-Can, "Multi-objective regression test selection in practice: An empirical study in the defense software industry," *Information and Software Technology,* 06/01 2018, doi: 10.1016/j.infsof.2018.06.007.

[4] S. Nidhra, "Black Box and White Box Testing Techniques - A Literature Review," *International Journal of Embedded Systems and Applications,* vol. 2, pp. 29-50, 06/30 2012, doi: 10.5121/ijesa.2012.2204.

[5] W. Youn, S. Hong, K. Oh, and O. Ahn, "Software Certification of Safety-Critical Avionic Systems: DO-178C and Its Impacts," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 30, pp. 4-13, 04/01 2015, doi: 10.1109/MAES.2014.140109.

[6] M.-H. Chen, M. Lyu, and W. Wong, "Effect of code coverage on software reliability measurement," *Reliability, IEEE Transactions on,* vol. 50, pp. 165-170, 07/01 2001, doi: 10.1109/24.963124.

[7] L. Hu, W. Wong, D. Kuhn, and R. Kacker, "How does combinatorial testing perform in the real world: an empirical study," *Empirical Software Engineering,* vol. 25, 07/01 2020, doi: 10.1007/s10664-019-09799-2.

[8] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Introduction to Combinatorial Testing* (Chapman & Hall/CRC Innovations in Software Engineering and Software

Development Series). Taylor & Francis, 2013.

[9] H. Al-Sammarraie and D. Jawawi, "Multiple Black Hole Inspired Meta-Heuristic Searching Optimization for Combinatorial Testing," *IEEE Access,* vol. PP, pp. 1-1, 02/13 2020, doi: 10.1109/ACCESS.2020.2973696.

[10] y. Lei, R. Kacker, D. Kuhn, V. Okun, and J. Lawrence, *IPOG: A general strategy for T-way software testing*. 2007, pp. 549-556.

[11] C. Nie and H. Leung, "A Survey of Combinatorial Testing," *ACM Comput. Surv.,* vol. 43, p. 11, 01/01 2011, doi: 10.1145/1883612.1883618.

[12] A. Hatamlou, "Black hole: A new heuristic optimization approach for data clustering," *Information Sciences,* vol. 222, pp. 175–184, 02/01 2013, doi: 10.1016/j.ins.2012.08.023.

[13] K. Zamli, "T-Way Strategies and Its Applications for Combinatorial Testing," *International Journal of New Computer Architectures and their Applications (IJNCAA),* vol. 2, pp. 459-473, 01/01 2011.

[14] D. Y. Ulutas, "The Generation of Optimized Test Data: Preliminary Analysis of a Systematic Mapping Study," 2020.

[15] C. Mao, X. Yu, J. Chen, and J. Chen, *Generating Test Data for Structural Testing Based on Ant Colony Optimization*. 2012, pp. 98-101.

[16] N. Tracey, J. Clark, K. Mander, and J. McDermid, "An automated framework for structural test-data generation," in *Proceedings 13th IEEE International Conference on Automated Software Engineering (Cat. No.98EX239)*, 13-16 Oct. 1998 1998, pp. 285-288, doi: 10.1109/ASE.1998.732680.

[17] A. El-Serafy, G. El Sayed, C. Salama, and A. Wahba, *Enhanced Genetic Algorithm for MC/DC test data generation*. 2015, pp. 1-8.

[18] K. Wang, Y. Wang, and L. Zhang, *Software testing method based on improved simulated annealing algorithm*. 2014, pp. 418-421.

[19] C. Mao, X. Yu, and J. Chen, *Swarm Intelligence-Based Test Data Generation for Structural Testing*. 2012.

[20] M. Farahmandian and A. Hatamlou, "Solving optimization problem using black hole algorithm," *Journal of Advanced Computer Science & Technology,* vol. 4, 12/25 2015, doi: 10.14419/jacst.v4i1.4094.

[21] E. Curiel, "The many definitions of a black hole," *Nature Astronomy,* vol. 3, 01/08 2019, doi: 10.1038/s41550-018-0602-1.

[22] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. 2009.