International Conference on Innovative Research in Engineering, Management and Sciences
International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2019 IJSRCSEIT | Volume 4 | Issue 9 | ISSN : 2456-3307

Multi-Objective Optimization Approach to Generate String Test Data

B. Swathi [*1]

[*1] Department of ISE, New Horizon College of Engineering, Bengaluru, India
baswarajuswathi@gmail.com [1]

ABSTRACT

String test cases are required by applications to recognize deformities and security issues. Nonetheless, its viability isn't tasteful. In this paper, discovery string test case generation techniques are investigated. Two objective functions are acquainted to produce compelling test cases. The optimization of the test cases is the primary objective, where it very well may be estimated through string distance functions. The second objective is controlling the string length distribution into a Benford distribution which suggests shorter strings have, all in all, a higher shot of discontent location. At the point when the two objectives are connected by means of a multi-objective advancement algorithm, predominant string test sets are delivered.

Keywords : Random Testing (RT), Benford distribution, String Test Cases

I. INTRODUCTION

In this paper, the objective is to produce a compelling set of test cases where each test case is a string. As clarified before, in light of exact investigations [1]– [5], blame areas regularly form nonstop locales in the info space. In light of this presumption, a various set of test cases has a more prominent possibility of distinguishing a blame. Thus, it is trusted that an assorted set of test cases is bound to deliver increasingly viable test cases. To accomplish this in the string space, we have characterized a wellness work that estimates the assorted variety of a test set. This enables an advancement procedure to be utilized to produce test cases dependent on the wellness work. To build a wellness capacity to gauge the decent variety, we use distance functions between strings. There are a few string distance functions accessible and subsequently, in this paper, the performance when utilized in test generation is considered. Diverse string distance capacity's performance is thought about as far as the viability of the produced test cases and their runtime. Since runtime performance is important in useful applications, further part expands the paper by applying a hash based distance work into the test generation strategies to improve the runtime productivity.

The distribution of the length of the created strings assumes an important job in disappointment recognition. The paper contends that littler strings have a higher possibility of distinguishing a disappointment. Since the primary wellness work is unfit to control the length distribution of the strings, second wellness work which demonstrates the nearness of the distribution of the lengths of the strings in a test set to the objective distribution is considered. A multi-objective improvement system is utilized to apply both wellness functions at the same time.

II. ADAPTIVE RANDOM STRING TEST CASE GENERATION

As examined in the past paper, to improve the poor adequacy of RT, ART strategies are presented. Chen et al. [6] first presented Fixed Size Candidate Set (FSCS) and afterward an assortment of other ART techniques[7] have been created by different scientists. The greater part of the ART techniques are intended for numerical test cases and they can't be utilized to produce string test cases. Among the ART techniques, the FSCS and ART for Object Oriented software (ARTOO) [8] strategies are prepared to do more mind boggling test case structures than fixed size vector of numbers and they can be connected to string test cases. Further, Mayer et al. [9] presumed that FSCS was a standout amongst the best ART techniques through an experimental investigation. Subsequently, adjusted FSCS and ARTOO to produce string test cases in this paper; these are checked on in the accompanying areas.

A. Fixed Size Candidate Set (FSCS)

FSCS technique [2] significantly reduces the computation time. In this paper, a string distance work is utilized in FSCS. FSCS has been at first presented for numerical test cases. Be that as it may, it tends to be connected to other test case structures like strings. The main prerequisite is that a distance work is characterized between the test cases.

To create test cases, FSCS utilizes a distance based method. The main string test case is created randomly, like RT. At that point, to create other test cases, a fixed size candidate set is utilized to deliver a test case. Therefore, K random strings are created as candidates (K=10 is utilized in the investigations dependent on the proposal of Chen et al. [7]). A string is chosen where it has the biggest distance from recently executed string test cases.

B. ART for Object Oriented Software (ARTOO)

ARTOO [9] is an ART strategy intended for object oriented software where it utilizes a distance work between objects to produce the test cases. The creators center around the particular issue of testing functions of an object-oriented program where test cases are input objects to the functions. ARTOO works like FSCS , it chooses a test case among the pool of candidates. The quantity of candidates for ARTOO is picked as 10 to match with the FSCS. The distinction among FSCS and ARTOO is the choice guideline among the candidates. The mean distance of every candidate to the recently chosen test cases is determined. At that point, a candidate with the biggest mean distance is picked as the champ (next test case).

III. EVOLUTIONARY STRING TEST CASE GENERATION

To produce string test cases, evolutionary algorithms can be utilized. Among the evolutionary algorithms, Genetic Algorithms (GA) [13] are the most ordinarily utilized pursuit algorithm in software building [14],GAs additionally fit great with our application which requires string controls. Two methodologies are utilized to create test sets dependent on GAs. In the first place, we use a GA with a solitary objective, where a decent variety based wellness work is utilized. At that point, a second wellness work is characterized to control the length distribution of the strings. Thus, in the second methodology, we utilize a Multi-Objective GA (MOGA) to advance both wellness functions at the same time.

A. Genetic Algorithm (GA)

In the accompanying, we first quickly clarify GA's essential phrasing and after that, fitting wellness functions and GA's parameters are examined. Different chromosomes form a populace where a chromosome is a candidate arrangement. At every generation, a few chromosomes are chosen (by the choice instrument) and posterity are produced by means of a hybrid administrator. At long last, the transformation administrator is used to roll out random little improvements to the created posterity bringing about a lower likelihood of getting to be caught in a nearby ideal point.

IV. METHODOLOGY

To produce the test cases the accompanying steps are performed by NSGA-II.
Step 1: The underlying populace with size N is produced randomly.
Step 2: The populace is sorted.
Step 3: A posterity populace with size N is made utilizing determination components, hybrid, and change.
Step 4: A joined populace of posterity and guardians is created with size 2N.
Step 5: The new populace is sorted and the main N chromosomes are chosen to form the people to come.
Step 6: A verify whether the halting foundation have been met is performed. On the off chance that the rule isn't met, at that point we come back to step 3.

V. STRING DISTANCE FUNCTIONS

A distance work between two strings is required in ART and evolutionary test case generation strategies. A few string distance functions are presented in the writing [9], [11], [12]. In spite of the fact that we can't afford to investigate every one of them, a great portion of them, particularly those that typically perform well in software testing examines, are shrouded in this paper. As needs be, we performed the tests with six string distance functions. Four of which are Levenshtein , Hamming, Cosine, Manhattan [12], and Euclidian distance functions that are more than once utilized in software testing thinks about [9], [11].Further, we likewise utilized Locality-Sensitive Hashing (LSH) method as a quick gauge of string distance in our work.

A. LEVENSHTEIN DISTANCE

The Levenshtein Distance is an alter put together distance that works based with respect to three alter activities, "erase", "insert", and "update". Every task has a related cost where each string can be converted to the next string dependent on these alter activities. The distance is the base expense of an arrangement of alter tasks that converts one string into the other string . The Levenshtein distance doles out a unit cost to all alter activities. Numerically, the Levenshtein distance between two strings, Str1 and Str2, is equivalent to lev(Length(Str1), Length (Str2)) where it very well may be determined recursively by

$$lev(i,j)=\begin{cases}\max(i,j) & \text{if } \min(i,j)==0\\ \min\begin{Bmatrix} lev(i-1,j)+1\\ lev(i,j-1)+1\\ lev(i-1,j-1)+cost(i,j)\end{Bmatrix} & \text{otherwise}\end{cases}$$

$$cost(i,j)=\begin{cases}0 & \text{if } Str1_i == Str2_j\\ 1 & \text{otherwise}\end{cases},$$

whereStr1i denotes the ith character of Str1, and Str2j denotes the jth character of Str2.

B. HAMMING DISTANCE

The Hamming distance [12] was at first acquainted as a measure with figure the distance of good for nothing streams. In any case, it has been adjusted to be utilized for strings [12]. The Hamming distance of two strings, as "abcd" and "anfd", is the quantity of characters diverse in two strings. At the end of the day, each character in the principal string is contrasted and a character in the proportionate position in the second string. In this precedent, the distance is two. In cases where the sizes of two strings are not equivalent, invalid characters (ASCII code of zero) are added as far as possible of the littler string until the two strings have an equivalent size. For instance, the distance among "abdominal muscle" and "acdb" is three[15].

C. MANHATTAN DISTANCE

The Manhattan distance [12] is regularly utilized for vectors of numbers. It likewise can be connected to strings as

$$\text{Manhattan distance}=\sum_{i=1}^{n}\left|Str1_i - Str2_i\right|$$

Where Str1i and Str2i are ASCII codes of the ith character. Like the Hamming distance, when the size of the two strings isn't equivalent, invalid characters are added to the shorter string.

D. EUCLIDIAN DISTANCE

The Euclidian distance is like the Manhattan distance. It very well may be connected to strings as

$$\text{Cartesian distance} = \sqrt{\sum_{i=1}^{n} (Str1_i - Str2_i)^2}$$

Again, invalid characters are added to the shorter string until the two strings have an equivalent size.

E. COSINE DISTANCE

The Cosine comparability ascertains the similitude of two vectors as a cosine of the edge of two vectors. The Cosine comparability can be determined as pursues where ASCII codes are utilized as a number.

$$\text{Cosine similarity} = \frac{\sum_{i=1}^{n} Str1_i \times Str2_i}{\sqrt{\sum_{i=1}^{n} Str1_i^{\,2}} \times \sqrt{\sum_{i=1}^{n} Str2_i^{\,2}}}.$$

Like The Hamming distance, when the size of the two strings isn't equivalent, invalid characters are added to the shorter string. At long last, to ascertain the distance, 1-Cosine comparability is utilized.

F. LOCALITY-SENSITIVE HASHING (LSH)

LHS [24] is a method that can be utilized as a quick estimation of the distance between two strings. The essential thought is to hash strings to such an extent that comparative strings are mapped into an equivalent hash code with a high likelihood. Random projections are center components used to outline input information to an esteem. In this paper, we utilized a sort of random projection that is utilized to gauge cosine distances. This projection is characterized as

$$h^x(\vec{v}) = \begin{cases} 1 & \vec{x} \cdot \vec{v} \geq 0 \\ 0 & \vec{x} \cdot \vec{v} < 0 \end{cases}$$

Where v is the information vector, x is a random vector produced from a Gaussian distribution, and h x (v) is a bit speaking to the area of v contrasted with x. P random projections are utilized to build a hash esteem where it demonstrates the area of the information vector contrasted with the P random vectors. Therefore, we have P bits as a hash esteem; P=32 is utilized in this exploration.

At long last, the Hamming distance is utilized between two hash bit strings which prompts an estimation of the cosine distance of the first strings. LSH improves the runtime request as the Hamming distance between two 32 bit streams is autonomous of the sizes of the strings. A far reaching runtime request investigation is exhibited in the following area.

Cosine and LSH distances are normally standardized against the length of the strings and subsequently, we don't have to standardize them. Be that as it may, the other talked about distances are not normally standardized. To standardize them, the outcome is partitioned by Length(Str1)+Length(Str2).

VI. CONCLUSION

In this paper, black-box string test case generation is examined. Two objectives are acquainted with produce viable string test cases. The principal objective controls the decent variety of the test cases inside a test set. As indicated by different observational examinations, blames as a rule happen in blunder precious stones or disappointment locales. Subsequently, controlling the decent variety of the test cases is an important part of black-box test case generation. The second objective is in charge of controlling the length distribution of the string test cases. The Benford distribution is utilized as an objective distribution. In like manner, a Kolmogorov– Smirnov test is used to develop the wellness work. At the point when the two objectives are enforced, utilizing a multi-objective advancement strategy, prevalent test cases are created.

VII. REFERENCES

[1] P. E. Ammann and J. C. Knight, “Data diversity: an approach to software fault tolerance,” Comput. IEEE Trans., vol. 37, no. 4, pp. 418–425, Apr. 1988.

[2] G. B. Finelli, “NASA Software failure characterization experiments,” Reliab. Eng. Syst. Saf., vol. 32, no. 1–2, pp. 155–169, 1991.

[3] L. J. White and E. I. Cohen, “A Domain Strategy for Computer Program Testing,” Softw. Eng. IEEE Trans., vol. SE-6, no. 3, pp. 247–257, May 1980.

[4] P. G. Bishop, “The variation of software survival time for different operational input profiles (or why you can wait a long time for a big bug to fail),” in Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on, 1993, pp. 98–107.

[5] C. Schneckenburger and J. Mayer, “Towards the determination of typical failure patterns,” in Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting, 2007, pp. 90–93.

[6] T. Y. Chen, T. H. Tse, and Y. T. Yu, “Proportional sampling strategy: a compendium and some insights,” J. Syst. Softw., vol. 58, no. 1, pp. 65–81, 2001.

[7] T. Y. Chen, H. Leung, and I. K. Mak, “Adaptive Random Testing,” in Advances in Computer Science - ASIAN 2004, vol. 3321, M. Maher, Ed. Springer Berlin / Heidelberg, 2005, pp. 3156–3157.

[8] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, “ARTOO: Adaptive Random Testing for Object-Oriented Software,” in Software Engineering, 2008. ICSE ’08. ACM/IEEE 30th International Conference on, 2008, pp. 71–80.

[9] J. Mayer and C. Schneckenburger, “An empirical analysis and comparison of random testing techniques,” in Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, 2006, pp. 105–114.

[10] C. Durtschi, W. Hillison, and C. Pacini, “The effective use of Benford’s law to assist in detecting fraud in accounting data,” J. forensic Account., vol. 5, no. 1, pp. 17–34, 2004.

[11] H. Hemmati, A. Arcuri, and L. Briand, “Achieving Scalable Model-based Testing Through Test Case Diversity,” ACM Trans. Softw. Eng. Methodol., vol. 22, no. 1, pp. 6:1–6:42, Mar. 2013.

[12] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, “Prioritizing test cases with string distances,” Autom. Softw. Eng., vol. 19, no. 1, pp. 65–95, 2012.

[13] D. Whitley, “A genetic algorithm tutorial,” Stat. Comput., vol. 4, no. 2, pp. 65–85, 1994.

[14] M. Harman and B. F. Jones, “Search-based software engineering,” Inf. Softw. Technol., vol. 43, no. 14, pp. 833–839, 2001.

[15] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, “A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation,” Softw. Eng. IEEE Trans., vol. 36, no. 6, pp. 742–762, Nov. 2010.