HKU CS Tech Report TR-2013-04

Postprint of article in The Symposium on Engineering Test Harness (TSETH '13),
Proceedings of the 13th International Conference on Quality Software (*QSIC '13*),
IEEE Computer Society, Los Alamitos, CA (2013)

The ART of Divide and Conquer

An Innovative Approach to Improving the Efficiency of Adaptive Random Testing

Cliff Chow
The University of Hong Kong
Pokfulam, Hong Kong
cliffchow@hku.hk

Tsong Yueh Chen*
Swinburne University of Technology
Australia
tychen@swin.edu.au

T.H. Tse
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

***Abstract*—Test case selection is a prime process in the engineering of test harnesses. In particular, test case diversity is an important concept. In order to achieve an even spread of test cases across the input domain, Adaptive Random Testing (ART) was proposed such that the history of previously executed test cases are taken into consideration when selecting the next test case. This was achieved through various means such as best candidate selection, exclusion, , and diversity metrics. Empirical studies showed that ART algorithms make good use of the concept of even spreading and achieve 40 to 50% improvement in test effectiveness over random testing in revealing the first failure, which is close to the theoretical limit. However, the computational complexity of ART algorithms may be quadratic or higher, and hence efficiency is an issue when a large number of previously executed test cases are involved. This paper proposes an innovative divide-and-conquer approach to improve the efficiency of ART algorithms while maintaining their performance in effectiveness. Simulation studies have been conducted to gauge its efficiency against two most commonly used ART algorithms, namely, fixed size candidate set and restricted random testing. Initial experimental results show that the divide-and-conquer technique can provide much better efficiency while maintaining similar, or even better, effectiveness.** *

***Keywords*—adaptive random testing, divide and conquer, efficiency, effectiveness, software testing, test harness**

I. INTRODUCTION

Test case selection is a major process in the development of test harnesses. Random testing [11] has been recognized as an important and useful method for test case selection. On the other hand, empirical studies have shown that failure-causing inputs, especially in numerical programs, tend to have contiguous failure regions [6]. Conceptually, if the current test cases cannot detect a failure, we should select the next test case to be far away from the test cases previously executed. Hence, evenly spreading the test cases across the input domain should provide a higher opportunity for revealing a failure.

Based on this intuition, Chen et al. [7] introduced the concept of *Adaptive Random Testing* (*ART*) to enhance the effectiveness of failure detection in random testing. The first ART method they proposed was the Fixed Size Candidate Set (FSCS) algorithm [6], [7], which generates a list of test cases as potential candidates, selects the best test case with the longest distance from a close neighborhood of previously executed test cases. Another proposed technique is the Restricted Random Testing (RRT), which selects the next test case as one that lies outside all the exclusive regions of previously executed test cases. Please refer to Section II for more details. Empirical studies show that ART algorithms, using the concept of even-spreading, achieve 40 to 50% improvement in effectiveness over random testing (in terms of the expected number of test cases required to detect the first failure), which is close to the theoretical limit [8].

However, ART is less efficient than random testing because of the extra task of ensuring even spreading of test cases, where the efficiency is measured in terms of the time to generate a test case. This extra task involves the computation of history information from previous test cases because the determination of the *i*-th test case is related to the first to the (*i*–1)-th previously executed test cases. The computational complexity of ART algorithms is generally quadratic or higher. Hence, ART may not be more cost-effective than random testing when efficiency is taken into consideration.

In order to enhance the efficiency of existing ART algorithms, two techniques were proposed, namely, *mirroring* by Kuo [12] and *forgetting* by Chan et al. [2]. Conceptually speaking, both techniques can be applied to all ART methods. In the mirroring technique, the ART algorithm is applied to part of the input domain, and then the generated test cases are copied like mirror images to the remaining parts. In the forgetting technique, not all previously executed test cases are used

© IEEE 2013. This material is presented to ensure timely dissemination of scholarly and technical work. Personal use of this material is permitted. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. Permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

* Corresponding author.

to guide the generation of the next test case. The maximum number of previously executed test cases to be retained is specified by a *memory parameter*. On the other hand, empirical studies show that there is a trade-off between computational complexity and effectiveness [16]. Experimental results in the forgetting paper [2], for instance, show that for a given memory parameter, a similar performance in effectiveness cannot be maintained across different failure rates.

The motivation of the present paper is to design a new and innovative approach, known as divide-and-conquer, for reducing the cost of test case generation for ART algorithms, as well as preserving the performance in effectiveness.

The present study serves as an introduction to an innovative technique that improves on the efficiency of ART while preserving its effectiveness as far as possible. We will focus our experimental comparison with the two most commonly used ART algorithms, namely FSCS and RRT. Evaluations with respect to other ART algorithms will be left as future work.

The contribution of our proposed technique is fourfold: (a) It is an innovative enhancement of ART algorithms that greatly improves the computational complexity from quadratic or higher order to linear order. (b) It preserves the even spreading property of ART and largely maintains the performance in effectiveness. (c) It can be applied to all ART algorithms because it is an add-on technique. (d) It is independent of the dimension of the input domain.

The paper is organized as follows: Section II provides the background information on FSCS and RRT. Section III describes the proposed divide-and-conquer approach and explains the differences between the new technique and two existing efficiency improvement techniques. A comparison of the efficiency and effectiveness between selected ART algorithms and the same algorithms enhanced with the divide-and-conquer strategy is presented in Section IV. Finally, we present the conclusion in Section V.

II. Background Information on FSCS and RRT

The *Fixed Size Candidate Set* (*FSCS*) algorithm for adaptive random testing, proposed by Chen et al. [6], [7], makes use of the simple intuition of selecting the best choice out of a fixed number of candidates to generate the next test case. Essentially, FSCS randomly generates k candidates and then calculates the distance $dist_i$ between each candidate c_i and its closest previously executed test case. The candidate c_{best} with the longest distance, defined as $c_{best} = \{c_i \mid dist_i = \max_{j\in\{1,2,\ldots,k\}} dist_j\}$, represents the best candidate. It is selected as the next test case, and other candidates are discarded. This best candidate selection procedure is repeated until the first failure is revealed or the appropriate time limit has been reached. Since the computing operation calculates the distance between the candidates and all previously executed test cases, the computational complexity of FSCS is $\Theta(n^2)$ [6], [7].

Chan et al. [3] introduced *Restricted Random Testing* (*RRT*), which is based on the notion of exclusion. All exclusion zones are circular and equal in size. Let A be the target exclusion zone and n be the number of previously executed test cases. Each exclusion area is A/n, and the radius of each exclusion zone is $\sqrt{A/(n\pi)}$. For each round of test case generation, an exclusion zone has to be defined for every test case previously executed. The potential test cases are randomly generated until there is a potential test case outside all exclusion regions. It will be used as the next test case. The size of the exclusion zone will decrease when the number of previously executed test case increases. Mayer and Schneckenburger [15] estimated that the computational complexity of RRT is $\Theta(n^2 \log(n))$, assuming that the number of candidate test cases is logarithmic in the number of previously executed test cases. They validated their assumption empirically for the situation where the exclusion zones are circular, the total areas of the exclusion zones is 150% of the area of the input domain, and the number of test cases is no more than 500.

III. Divide-and-Conquer Approach

A. Background

The proposed approach makes use of the standard concept of “divide and conquer” for breaking up a large problem into smaller sub-problems. It applies the exact ART algorithm to each of the sub-problems until the computational operation is too expensive for applying the ART algorithm again. It then further breaks up the sub-problems and repeats the procedure recursively until either a first failure is revealed or the time limit is reached.

While the target of the proposed divide-and-conquer approach is in alignment with two existing techniques for reducing the computational complexity of ART algorithms, namely, mirroring and forgetting, the concept of the new approach is different. In mirroring [12], the input domain is partitioned into disjointed sub-domains, and then the process selects a sub-domain as a *source domain* while the others are called *mirror domains*. The ART algorithm will only be applied to the *source domain*, and then the process uses a function to map the generated test cases from the *source domain* to all *mirror domains*. This mirroring procedure is repeated until a first failure is identified.

In forgetting [2], the researchers make use of the forgetting principle, which refers to “*Human learning is often characterised by inaccurate retention or recall, termed forgetting*” [2], to tackle the problem of the number of computational operations when the number of previously executed test cases grows. For a given memory parameter, the process forgets some previously executed test cases in order to improve the efficiency in generating the next test case.

The proposed divide-and-conquer approach uses bisectional division to break up the input domain into smaller subdomains, and generates a next test case from each sub-domain. The divide-and-conquer approach does not forget any historical information from previously executed test cases. It only ignores test cases outside the sub-domain under consideration in order to improve the efficiency of ART algorithms. This is a new and innovative approach for efficiency improvement in ART.

B. *Proposed Approach*

The approach starts with any ART algorithm and a specified integer known as the *threshold*, denoted by $\boldsymbol{\lambda}$. The determination of the *threshold* will be discussed later. Let $\boldsymbol{D}$ denote the input domain and $\boldsymbol{T}$ denote a set of previously executed test cases. After $\boldsymbol{\lambda}$ test cases have been generated in $\boldsymbol{T}$ from the input domain $\boldsymbol{D}$, the divide-and-conquer process is triggered. The process bisectionally divides each dimension of the input domain into equal-sized sub-domains $\{\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_s\}$ such that $\cup_{i=1}^{s}\boldsymbol{D}_i = \boldsymbol{D}$, where $s = 2^{dn}$ denotes the number of sub-domains after each division process and *dn* denotes the dimension of the input space. The next step is to allocate test cases into their relevant sub-domains. This is achieved by putting all the sub-domains thus generated into a global queue and selecting the least populated sub-domain $\boldsymbol{D}_i$ such that $|\boldsymbol{T}_i| = \min_{j\in\{1,2,\ldots,s\}} |\boldsymbol{T}_j|$, where |.| denotes the size of a set and $\boldsymbol{T}_i$ denotes the set of selected test cases in the sub-domain $\boldsymbol{D}_i$, that is, $\boldsymbol{T}_i = \boldsymbol{T} \cap \boldsymbol{D}_i$, from which the next test case will be generated. The process of selecting the currently least populated sub-domain is repeated until the numbers of test cases for all sub-domains reach $\boldsymbol{\lambda}$ again. Then, the process of bisectional division of the sub-domains will be recursively applied.

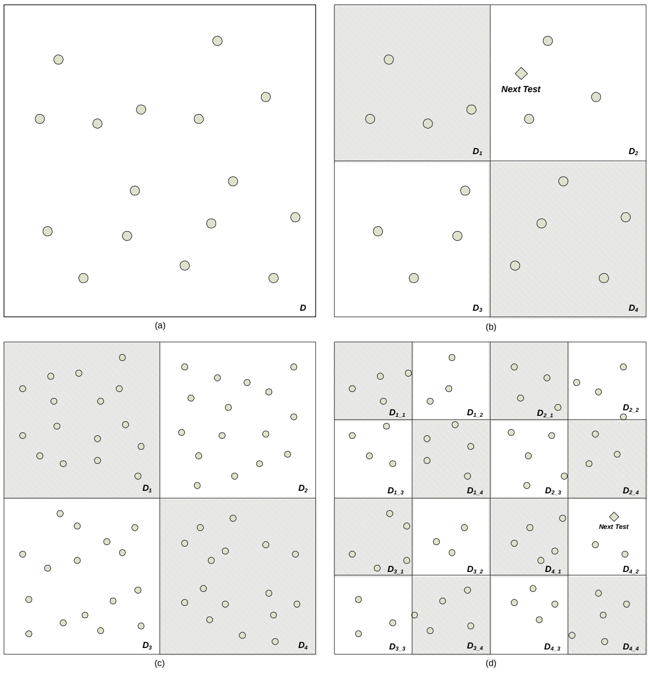

Fig. 1. The operation of ART with divide-and-conquer.

As an example for illustration, consider a 2-dimension input domain $\boldsymbol{D}$. Fig. 1 shows the proposed divide-and-conquer approach in action. Suppose $\boldsymbol{\lambda}$ is 16. In Fig. 1(a), when the total number of test cases $|\boldsymbol{T}|$ in the input domain $\boldsymbol{D}$ reaches $\boldsymbol{\lambda}$, the divide-and-conquer process is triggered. In Fig. 1(b), input domains are bisectionally divided into four equal-sized sub-domains. After the test case reassignment process, the distribution of test cases are $|\boldsymbol{T}_1| = 4$, $|\boldsymbol{T}_2| = 3$, $|\boldsymbol{T}_3| = 4$, and $|\boldsymbol{T}_4| = 5$ in sub-domains $\boldsymbol{D}_1$, $\boldsymbol{D}_2$, $\boldsymbol{D}_3$, and $\boldsymbol{D}_4$ respectively. In accordance with the selection criterion, which chooses the least populated sub-domain to proceed, the next test case generation process is applied to sub-domain $\boldsymbol{D}_2$. After generating one test case in $\boldsymbol{D}_2$, the next test case should be generated from one of the three sub-domains $\{\boldsymbol{D}_1, \boldsymbol{D}_2, \boldsymbol{D}_3\}$, which have the same number of test cases previously executed (that is, 4). Then, the selection criterion is repeated until the numbers of test cases in all the sub-domains reach the *threshold* value $\boldsymbol{\lambda}$ again. In Fig. 1(c), the numbers of previously executed test cases in all sub-domains $\{\boldsymbol{D}_1, \boldsymbol{D}_2, \boldsymbol{D}_3, \boldsymbol{D}_4\}$ reach the *threshold* value of 16, and the process starts the next bisectional division of all sub-domains. In Fig. 1(d), after the said bisectional division procedure, the process allocates all the sub-domains with a *depth* of 2 into a global queue (where *depth* denotes the depth of the bisectional division) and then repeats the selection criteria in order to find the least populated sub-domain $\boldsymbol{D}_{4\_2}$ where $|\boldsymbol{T}_{4\_2}|$ is 2.

The algorithm for adaptive random testing with divide-and-conquer is as follows:

Algorithm 1. Adaptive Random Testing with Divide-and-Conquer (DC)

```
$\boldsymbol{\lambda}$ = threshold for the divide-and-conquer process;
    /* The threshold is set by the human tester */
$\boldsymbol{D}$ = the entire input domain;
$\boldsymbol{T}$ = {};  /* a list to store the generated test cases in the input domain $\boldsymbol{D}$ */
depth = 0;
d_queue = {$\boldsymbol{D}$};  /* a list to store $\boldsymbol{D}$ and $\boldsymbol{D}_i$ */
t_queue = {$\boldsymbol{T}$};  /* a list to store $\boldsymbol{T}$ and $\boldsymbol{T}_i$ */
tmp_d_queue = {};  /* a temporary storage for $\boldsymbol{D}$ and $\boldsymbol{D}_i$ */
tmp_t_queue = {};  /* a temporary storage for $\boldsymbol{T}$ and $\boldsymbol{T}_i$ */
while (stopping criteria not reached)
    select the least populated sub-domain $\boldsymbol{D}_i$ from d_queue;
        /* that is, $\boldsymbol{D}_i$ such that $|\boldsymbol{T}_i| = \min_{j\in\{1,2,\ldots,|t\_queue|\}} |\boldsymbol{T}_j|$ */
    $t$ = ART($\boldsymbol{D}_i$, $\boldsymbol{T}_i$) /* generate a new test case by applying ART to $D_i$
        with $T_i$ as the set of all test cases previously executed */
    test the program using $t$ as a test case;
    if $t$ reveals a failure
        exit;
    else
        add $t$ to $\boldsymbol{T}_i$;
    end if;
    if $|\boldsymbol{T}_i| = \boldsymbol{\lambda}$
        remove $\boldsymbol{D}_i$ from d_queue;
        remove $\boldsymbol{T}_i$ from t_queue;
        start the bisectional division process on $\boldsymbol{D}_i$ and create $s$ sub-domains
            $\boldsymbol{D}_{i\_j}$ such that $\cup_{j=1}^{s}\boldsymbol{D}_{i\_j} = \boldsymbol{D}_i$, where $s$ denotes the number of sub-
            domains after each division process;
        create $|\boldsymbol{T}_{i\_j}|$ lists to store the generated test cases in each sub-domain
            $\boldsymbol{D}_{i\_j}$;
        for each $t_k$ in $\boldsymbol{T}_i$ /* test case reassignment process */
            if $t_k$ is in $\boldsymbol{D}_{i\_j}$
                add $t_k$ to $\boldsymbol{T}_{i\_j}$;
            end if;
        end for;
        add all $\boldsymbol{D}_{i\_j}$ to tmp_d_queue;
        add all $\boldsymbol{T}_{i\_j}$ to tmp_t_queue;
        delete $\boldsymbol{D}_i$ and $\boldsymbol{T}_i$;
    end if;
    if t_queue is empty
        d_queue = tmp_d_queue;
        t_queue = tmp_t_queue;
        tmp_d_queue = {};
        tmp_t_queue = {};
        depth ++;
    end if;
end while;
```

C. Determination of threshold

The major concept of the proposed divide-and-conquer approach is to partition a large problem into smaller sub-problems when the number of previously executed test cases reaches the threshold λ, in the sense that it tackles the smaller sub-domains individually rather than the entire input domain. If λ is set to be large, previously executed test cases should probably be evenly distributed, which is the property of ART, and there will be more or less the same number of test cases in each of the resultant sub-domains. However, the computational complexity will be high in this scenario. On the other hand, if λ is set to be too small, such as 4 in a 2-dimensional space, there will be a high chance that one of its 4 sub-domains may have no test case. This means that the next test case will be directly generated by a purely random generation process, thus losing any relationship with other previously executed test cases and defeating the target of even spreading. For this reason, it would be better to avoid too small a value for λ so as to increase the probability that each sub-domain contains at least one previously executed test case.

For the case of pure random testing, since the sub-domains are in equal-sized partitions, the determination problem can be viewed as a dice problem [9]: "*How many dice must be rolled to have at least a* 95*% chance of rolling a one, a two, a three, a four, a five and a six?*" The approach can be extended even further in accordance with the inclusion-exclusion principle [10], which is a mathematic counting technique to obtain the total number of elements in the union of two or more finite sets (such as $|A \cup B \cup C| = |A| + |B| + |C| - |B \cap C| - |C \cap A| - |A \cap B| + |A \cap B \cap C|$, where A, B, and C are finite sets). The probability $p(n)$ for having at least one test case residing in every resultant sub-domain in 2-dimensional space is

$$\begin{aligned} p(n) &= \sum_{j=0}^{3} (-1)^j \binom{4}{j} \left(\frac{4-j}{4}\right)^n \\ &= 1 - 4\left(\frac{3}{4}\right)^n + 6\left(\frac{1}{2}\right)^n - 4\left(\frac{1}{4}\right)^n \end{aligned} \qquad (1)$$

This probability exceeds 0.95 when $n \geq 16$. Table I shows some of the probabilities for various values of n. In other words, there is 95% chance that at least one test case falls into every resultant sub-domain after the 16th test case generation in 2-dimensional input domain. Rewriting (1) in a general form in dn-dimensional input space, we have

$$p(n) = \sum_{j=0}^{2^{dn}-1} (-1)^j \binom{2^{dn}}{j} \left(\frac{2^{dn}-j}{2^{dn}}\right)^n \qquad (2)$$

Equation (2), however, is not applicable to adaptive random testing because ART does not produce independent and uniformly distributed random values. Having said that, (2) provides a rough guideline for determining the *threshold* for divide and conquer. More discussions and experimental results will be described in the next section.

TABLE I. PROBABILITIES $p(n)$ OF GENERATING AT LEAST ONE TEST CASE IN EVERY SUB-DOMAIN IN 2-DIMENSIONAL SPACE

n	***p*(*n*)**	***n***	***p*(*n*)**	***n***	***p*(*n*)**
4	0.093750	10	0.780602	16	0.960001
5	0.234375	11	0.833988	17	0.969978
6	0.380859	12	0.874759	18	0.977472
7	0.512695	13	0.905703	19	0.983098
8	0.622925	14	0.929094	20	0.987321
9	0.711365	15	0.946729	21	0.990489

IV. Analysis and Experimental Results

A. Efficiency improvement

The determination of the *threshold* is an important decision. On one hand, the reduction of computational complexity will not be significantly improved if the *threshold* is set to be too high. On the other hand, if the *threshold* is assigned too small a value, the overall process will be dominated by sub-divisions of the input domain. Putting it to the extreme, suppose we set the *threshold* to be 1. The bisectional division process will be executed whenever one test case is generated by random testing in the sub-domains. In other words, the ART algorithm is never applied in this scenario.

It is easy to see that the computational complexity of the divide-and-conquer approach contains two major components: The first component is the operation steps of the distance calculation for a specific ART algorithm in all input sub-domains, as shown in Figs. 1(a) and (c). The second is the process of reassigning of all the test cases into newly created sub-domains by the bisectional division procedure, as displayed in Figs. 1(b) and (d).

Consider the example in Fig. 1 that executes the FSCS-ART algorithm [6], [7] with divide-and-conquer technique in a 2-dimensional space such that the *threshold* λ is set to a constant value of 16. Referring to Section II, the complexity of distance computation in FSCS is $\Theta(n^2)$. Since divide-and-conquer is triggered whenever that number of previously executed test cases has reached the *threshold* λ, the asymptotic value of n will not exceed λ, which is a constant. Hence, the complexity of distance computation in FSCS is also constant. Thus, after applying the divide-and-conquer approach, the computational complexity of ART algorithms becomes a linear order of the number of previously executed test cases.

On the other hand, another overhead is related to the process of dividing a large domain into smaller sub-domains and the assignment of test cases to each resultant sub-domain. Let m_{first} be the number of test cases to detect the first failure and s be the number of resultant sub-domains after dividing a domain. For instance, $s = 4$ in a 2-dimensional input space. The *depth*, which means the number of iterations of the division process, is given by $m_{\text{first}} = \lambda s^{depth-1}$. Rearranging the equation, we obtain:

$$depth = \begin{cases} \left\lfloor \dfrac{\log(m_{\mathrm{first}}/\boldsymbol{\lambda})}{\log(s)} \right\rfloor + 1 & m_{\mathrm{first}} \geq \boldsymbol{\lambda} \\ 0 & m_{\mathrm{first}} < \boldsymbol{\lambda} \end{cases} \quad (3)$$

Considering the scenario in a 2-dimensional space, where $s = 4$. Let the *threshold* $\boldsymbol{\lambda}$ be 100. A *depth* of 0 will result when the number of test cases is in the range of [0, 100), a *depth* of 1 will take place when the range is [100, 400), a *depth* of 2 will result when the range is [400, 1600), and so on. In general, since the *depth* can be found in (3), the overhead related to the bisectional division process and the assignment of test cases to each resultant sub-domain and the can be computed.

A comparison analysis was conducted on two commonly used ART algorithms, namely FSCS and RRT, with and without enhancement by the divide-and-conquer technique. In FSCS, the parameter for the number of the potential candidates was set to a constant $k = 10$ after Chen et al. [6], [7]. Similarly, the target exclusion ratio for the RRT was assigned to 150% after Chan et al. [2], [3]. For the divide-and-conquer approach, *threshold*s of 10, 50, and 100 were applied. The simulation exercises were repeated 1,000 times and the experimental results of the computation times of the algorithms are reported in Table II. In addition to computational complexity, diversity measurement metrics, namely, discrepancy and dispersion [5], were also used to measure the effectiveness of the divide-and-conquer technique.

The *discrepancy* metric indicates whether different regions in the input domain $\boldsymbol{D}$ have equal densities in test cases. The metric is formulated as:

$$discrepancy = \max_{r \in \{1,2,\ldots,1000\}} \left| \left|\frac{\boldsymbol{T}_r}{\boldsymbol{T}}\right| - \left|\frac{\boldsymbol{D}_r}{\boldsymbol{D}}\right| \right|$$

where $\boldsymbol{D}_r$ denotes a rectangular sub-domain of $\boldsymbol{D}$ whose size and location are randomly defined, and $\boldsymbol{T}_r$ denotes the set of test cases randomly selected from sub-domain $\boldsymbol{D}_r$, that is, $\boldsymbol{T}_i = \boldsymbol{T} \cap \boldsymbol{D}_r$. Thus, a low discrepancy value indicates that the test cases are equal in densities across all the sub-domains in $\boldsymbol{D}$.

The *dispersion* metric indicates whether there is a large empty region in input domain $\boldsymbol{D}$, which is reflected by the maximum distance of any test case from its nearest neighbor. The metric is formulated as:

$$dispersion = \max_{p \in \{1,2,\ldots,|T|\}} \min_{q \in \{1,2,\ldots,|T|\}} dist(t_p, t_q)$$

where the function *dist* denotes the Euclidean distance between two test cases. Thus, a low dispersion value indicates small empty regions in the input domain $\boldsymbol{D}$.

The empirical results in Figs. 2 and 3 demonstrate that significant efficiency improvements are achieved after applying the divide-and-conquer technique to either ART algorithm. The computational complexity is reduced. The reduction in computational complexity depends on the value of the *threshold* and the nature of the ART algorithm. Table II further shows that FSCS-DC and RRT-DC can perform better in terms of the discrepancy metric than the respective basic ART algorithms, FSCS and RRT. As expected, the dispersion diversity also indicates that there are larger empty regions for FSCS-DC and RRT-DC compared with the basic FSCS and RRT, respectively.

TABLE II. COMPARISONS OF COMPUTATIONAL COMPLEXITY, DISCREPANCY, AND DISPERSION BETWEEN FSCS AND FSCS WITH DC AND BETWEEN RRT AND RRT WITH DC

Computational Complexity (in Seconds)								
n	***FSCS***	***FSCS-DC,*** $\boldsymbol{\lambda = 100}$	***FSCS-DC,*** $\boldsymbol{\lambda = 50}$	***FSCS-DC,*** $\boldsymbol{\lambda = 10}$	***RRT***	***RRT-DC,*** $\boldsymbol{\lambda = 100}$	***RRT-DC,*** $\boldsymbol{\lambda = 50}$	***RRT-DC,*** $\boldsymbol{\lambda = 10}$
50	0.02	0.02	0.02	0.01	0.02	0.02	0.02	0.01
100	0.07	0.07	0.04	0.01	0.08	0.07	0.04	0.01
200	0.27	0.12	0.09	0.02	0.29	0.12	0.09	0.02
400	1.04	0.33	0.15	0.05	1.23	0.33	0.15	0.05
800	4.10	0.54	0.36	0.10	5.07	0.55	0.37	0.10
1600	16.30	1.35	0.59	0.19	20.75	1.36	0.60	0.19
Discrepancy								
50	0.114	0.114	0.114	0.104	0.119	0.133	0.120	0.104
100	0.079	0.085	0.068	0.071	0.083	0.085	0.072	0.066
200	0.055	0.050	0.053	0.045	0.057	0.053	0.045	0.046
400	0.038	0.029	0.031	0.030	0.042	0.032	0.030	0.028
800	0.026	0.019	0.020	0.018	0.029	0.022	0.021	0.019
1600	0.018	0.014	0.012	0.013	0.021	0.014	0.013	0.012
Dispersion								
50	0.179	0.177	0.179	0.198	0.177	0.177	0.175	0.177
100	0.126	0.128	0.131	0.134	0.123	0.126	0.129	0.129
200	0.092	0.092	0.095	0.097	0.088	0.094	0.090	0.093
400	0.066	0.066	0.068	0.070	0.063	0.064	0.067	0.068
800	0.047	0.048	0.048	0.049	0.045	0.047	0.047	0.048
1600	0.034	0.034	0.034	0.036	0.032	0.033	0.034	0.036

B. *Comparisons of Effectiveness Using F-measure*

In this paper, the expected number of test cases to detect the first failure, usually referred to as the F-measure, is used as the effectiveness measure [4], [6]. F-measure is equal to $1/\theta$ or $|\boldsymbol{D}|/m$ for random testing with replacement, where m denotes the number of failures and θ denotes the failure rate, such that $m = |\boldsymbol{D}|\theta$. A lower value of the F-measure indicates a more effective testing approach. It is generally agreed that the F-measure is a more appropriate effectiveness measure for random testing and ART compared with other effectiveness metrics [1], [6].

There are three most typical failure patterns [4], [13], [14], namely, block, strip, and point patterns, as illustrated in Figs. 4a to 4c, respectively. In these figures, the bounding box indicates the input domain, and the colored block, strip, and dots represent the locations of failure-causing inputs.

The effectiveness for applying divide-and-conquer technique with respect to these typical failure patterns was studied via simulation. A 2-dimensional input domain was used, with randomly generated failure patterns. For the block pattern, a square was randomly chosen within the input domain such that its size yields the desired failure rate. For the strip pattern, two points on adjacent sides were picked, and a strip connecting the two points determined the failure region. For the point pattern, 50 non-overlapping circular regions were randomly chosen with equal radii.

TABLE III. COMPARISONS OF F-MEASURES BETWEEN FSCS AND FSCS WITH DIVIDE-AND-CONQUER FOR BLOCK, STRIP, AND POINT PATTERNS

Failure rate	Expected F-measure of random testing (F_{rt})	**FSCS**		**FSCS-DC** (λ = 100)		**FSCS-DC** (λ = 50)		**FSCS-DC** (λ = 10)		**FSCS-DC** (λ = 4)	
		Mean of FSCS (F_{fscs})	(F_{fscs}/F_{rt})	Mean of FSCS-DC ($F_{fscs-dc}$)	($F_{fscs-dc}/F_{rt}$)	Mean of FSCS-DC ($F_{fscs-dc}$)	($F_{fscs-dc}/F_{rt}$)	Mean of FSCS-DC ($F_{fscs-dc}$)	($F_{fscs-dc}/F_{rt}$)	Mean of FSCS-DC ($F_{fscs-dc}$)	($F_{fscs-dc}/F_{rt}$)
Block Pattern											
0.01	100	68.4	68.43%	69.3	69.32%	69.2	69.16%	73.3	73.26%	79.7	79.70%
0.005	200	134.6	67.30%	137.5	68.77%	136.3	68.15%	146.5	73.26%	161.4	80.72%
0.002	500	325.1	65.01%	335.5	67.11%	334.7	66.95%	351.7	70.35%	388.2	77.63%
0.001	1000	650.2	65.02%	661.5	66.15%	665.5	66.55%	715.4	71.54%	781.4	78.14%
0.0005	2000	1290.4	64.52%	1309.3	65.46%	1324.5	66.23%	1398.5	69.92%	1517.8	75.89%
Strip Pattern											
0.01	100	92.1	92.13%	93.6	93.56%	94.1	94.14%	95.7	95.73%	104.4	104.41%
0.005	200	188.2	94.09%	190.1	95.07%	189.8	94.92%	192.3	96.17%	209.6	104.82%
0.002	500	475.3	95.06%	486.0	97.21%	484.7	96.93%	488.7	97.73%	522.7	104.54%
0.001	1000	944.9	94.49%	952.1	95.21%	976.0	97.60%	963.2	96.32%	1037.0	103.70%
0.0005	2000	1932.5	96.63%	1922.4	96.12%	1901.8	95.09%	1928.6	96.43%	2043.2	102.16%
Point Pattern											
0.01	100	100.6	100.57%	99.7	99.74%	100.4	100.45%	100.9	100.85%	108.3	108.33%
0.005	200	200.3	100.17%	199.9	99.93%	201.6	100.78%	201.8	100.89%	218.8	109.42%
0.002	500	496.8	99.37%	495.9	99.18%	494.7	98.95%	501.2	100.25%	525.8	105.17%
0.001	1000	991.7	99.17%	978.7	97.87%	993.9	99.39%	995.4	99.54%	1070.5	107.05%
0.0005	2000	1978.7	98.94%	1965.7	98.29%	1982.6	99.13%	1973.9	98.69%	2124.6	106.23%

TABLE IV. COMPARISONS OF F-MEASURES BETWEEN RRT AND RRT WITH DIVIDE-AND-CONQUER FOR BLOCK, STRIP, AND POINT PATTERNS

Failure rate	Expected F-measure of random testing (F_{rt})	**RRT**		**RRT-DC** (λ = 100)		**RRT-DC** (λ = 50)		**RRT-DC** (λ = 10)		**RRT-DC** (λ = 4)	
		Mean of RRT (F_{rrt})	(F_{rrt}/F_{rt})	Mean of RRT-DC (F_{rrt-dc})	(F_{rrt-dc}/F_{rt})	Mean of RRT-DC (F_{rrt-dc})	(F_{rrt-dc}/F_{rt})	Mean of RRT-DC (F_{rrt-dc})	(F_{rrt-dc}/F_{rt})	Mean of RRT-DC (F_{rrt-dc})	(F_{rrt-dc}/F_{rt})
Block Pattern											
0.01	100	66.1	66.14%	67.7	67.68%	68.0	68.01%	70.9	70.94%	72.8	72.79%
0.005	200	127.6	63.78%	133.1	66.56%	135.1	67.57%	141.6	70.82%	146.0	73.02%
0.002	500	310.0	62.00%	323.0	64.59%	330.0	66.00%	346.1	69.22%	360.3	72.07%
0.001	1000	614.9	61.49%	645.4	64.54%	659.1	65.91%	698.5	69.85%	727.2	72.72%
0.0005	2000	1218.1	60.91%	1289.2	64.46%	1289.7	64.49%	1365.2	68.26%	1440.5	72.03%
Strip Pattern											
0.01	100	91.4	91.36%	92.9	92.91%	93.6	93.63%	94.1	94.09%	95.0	94.99%
0.005	200	184.6	92.28%	190.7	95.35%	189.9	94.93%	189.8	94.91%	192.2	96.11%
0.002	500	479.1	95.81%	479.3	95.86%	482.4	96.47%	479.8	95.96%	484.3	96.85%
0.001	1000	947.6	94.76%	971.1	97.11%	960.5	96.05%	948.7	94.87%	962.6	96.26%
0.0005	2000	1906.8	95.34%	1898.6	94.93%	1939.0	96.95%	1933.0	96.65%	1933.0	96.65%
Point Pattern											
0.01	100	102.5	102.55%	101.1	101.13%	100.9	100.87%	100.0	99.96%	112.1	112.13%
0.005	200	201.9	100.96%	200.7	100.35%	202.3	101.13%	198.8	99.41%	225.3	112.67%
0.002	500	502.2	100.44%	496.3	99.26%	499.7	99.95%	496.9	99.38%	554.1	110.83%
0.001	1000	995.5	99.55%	987.9	98.79%	997.9	99.79%	1004.0	100.40%	1111.3	111.13%
0.0005	2000	1974.6	98.73%	1975.0	98.75%	1990.4	99.52%	2010.0	100.50%	2227.6	111.38%

TABLE V. COMPARISONS OF F-MEASURES BETWEEN RANDOM TESTING (RT) AND RANDOM TESTING WITH DIVIDE-AND-CONQUER (RT-DC) FOR BLOCK, STRIP, POINT PATTERNS

Failure rate	Expected F-measure of RT (F_{rt})	**RT-DC** (λ = 100)		**RT-DC** (λ = 50)		**RT-DC** (λ = 10)		**RT-DC** (λ = 4)	
		Mean of RT-DC (F_{rt-dc})	(F_{rt-dc}/F_{rt})	Mean of RT-DC (F_{rt-dc})	(F_{rt-dc}/F_{rt})	Mean of RT-DC (F_{rt-dc})	(F_{rt-dc}/F_{rt})	Mean of RT-DC (F_{rt-dc})	(F_{rt-dc}/F_{rt})
Block Pattern									
0.01	100	98.5	98.54%	98.0	98.01%	90.0	90.04%	83.9	83.91%
0.005	200	195.7	97.87%	192.3	96.15%	179.8	89.90%	166.7	83.33%
0.002	500	491.1	98.22%	482.6	96.51%	449.9	89.97%	421.4	84.28%
0.001	1000	978.1	97.81%	978.6	97.86%	899.8	89.98%	834.9	83.49%
0.0005	2000	1942.2	97.11%	1950.5	97.53%	1789.5	89.47%	1688.3	84.41%
Strip Pattern									
0.01	100	106.0	106.01%	102.9	102.88%	99.9	99.89%	99.5	99.46%
0.005	200	204.3	102.15%	201.3	100.64%	197.5	98.76%	197.6	98.81%
0.002	500	492.7	98.54%	499.5	99.90%	488.1	97.62%	487.0	97.40%
0.001	1000	1003.3	100.33%	998.5	99.85%	986.4	98.64%	978.8	97.88%
0.0005	2000	1976.8	98.84%	1994.0	99.70%	1965.2	98.26%	1972.9	98.64%
Point Pattern									
0.01	100	100.0	99.95%	99.3	99.28%	99.7	99.72%	98.2	98.20%
0.005	200	201.6	100.78%	199.0	99.48%	197.3	98.66%	199.7	99.83%
0.002	500	502.1	100.42%	502.3	100.45%	497.6	99.52%	498.5	99.69%
0.001	1000	1004.4	100.44%	1011.4	101.14%	1007.8	100.78%	990.4	99.04%
0.0005	2000	2001.7	100.09%	2006.7	100.34%	1977.9	98.90%	1988.1	99.41%

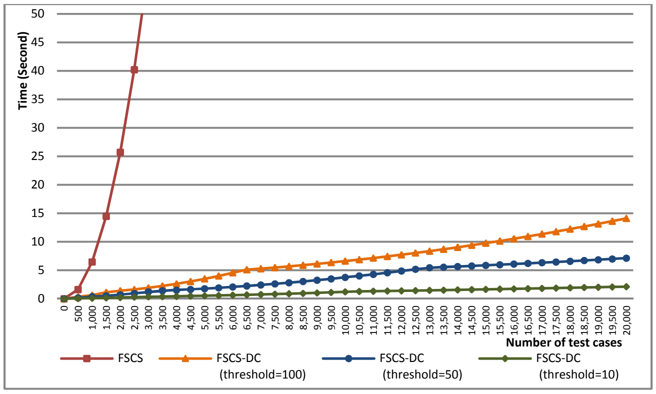

Fig. 2. Comparisons of computational complexity (in seconds) between FSCS and FSCS with divide-and-conquer with thresholds of 10, 50, and 100.

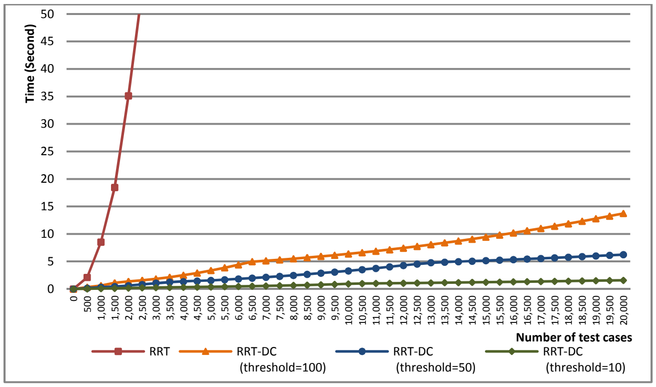

Fig. 3. Comparisons of computational complexity (in seconds) between RRT and RRT with divide-and-conquer with thresholds of 10, 50, and 100.

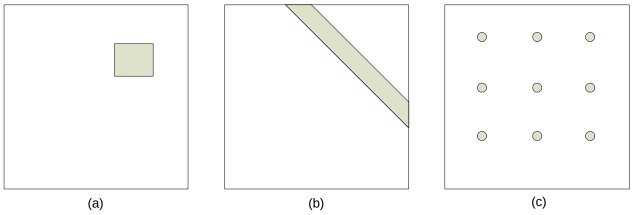

Fig. 4. Block, Strip, and Point failure patterns in 2-dimensional space.

The simulations were conducted with the following variables and values:

- Failure rates: 0.01, 0.005, 0.002, 0.001, and 0.0005
- Failure patterns: block, strip, and point
- Algorithms: FSCS, RRT, and random testing
- *Thresholds*: 100, 50, 10, and 4

For each combination of failure rate, failure pattern, algorithms, and *threshold*, 20,000 test runs were executed. The average F-measure for each exercise was collected. The same parameters for both the FSCS and RRT algorithms were used as per Section IV-A.

The experimental results in Tables III to IV demonstrate that the divide-and-conquer technique on top of ART performs similar, or even better, effectiveness than the original ART algorithms for the point failure pattern. The results also show that there is only a slight decline of 1–3% in effectiveness of the divide-and-conquer approach for the block and strip failure patterns.

Figs. 2 and 3 show that the F-measure slightly increases with efficiency improvement by reducing the *threshold* from 100 to 10. Tables III to IV show that the F-measure further increases if a small value of *threshold* such as 4 is assigned. These results imply that the effectiveness in FSCS-DC and RRT-DC becomes slightly poorer for smaller *threshold* values.

We also have studied the effect of divide-and-conquer technique to random testing. Table V illustrates that improvements in the F-measure for conducting random testing with divide-and-conquer can be significant. Future careful review will be required to verify whether divide-and-conquer can be used as a technique for improving the even spread of random test cases without the use of basic ART algorithms.

C. Threats to Validity

1) Internal validity.

This paper mainly serves to introduce an innovative efficiency improvement technique, namely divide-and-conquer. The experiments focus on simulations and may not represent a full variety of possible execution patterns of real-life faulty programs. However, in order to provide an "unbiased" cover, the experimental evaluations have been conducted on the three typical failure patterns, namely, block, strip, and point patterns, with wide ranges of failure rates. Various previous studies [3], [7], [16] have demonstrated that simulations of the three classic failure patterns can provide an objective validation of comparative merits.

Another threat to internal validity of the experiment is the choice of basic ART algorithms for the evaluation exercise. Since the two ART algorithms, namely Fixed Size Candidate Set (FSCS) and Restricted Random Testing (RRT), are commonly used and most involved in the comparisons of various random testing techniques [15], [16], they stand a representative position.

A further study covering real-life faulty programs and including other ART algorithms will alleviate the threats to internal validity.

2) External validity

There is one issue that may affect the external validity of the experiment. The Math::Random module, which is the Perl port of the C version of Randlib, is applied as the pseudorandom number generator to produce for the experiment a sequence of numbers that are approximately random. Randlib has been chosen because it comprises a large library of C routines for generating random numbers with uniform distribution under the UNIX and Linux environments. The use of other pseudorandom number generators may produce slightly different results.

3) *Construct validity*

There are two commonly used metrics to evaluate the effectiveness performance of testing algorithms, namely F-measure and P-measure, which refer to the expected number of test cases to detect the first failure and the probability of detecting at least one failure, respectively. To serve the purpose of introducing an innovative technique in this paper, F-measure is more appropriate as an effectiveness performance metric in order to synchronize and compare with various previous studies [3], [7], [15]. Furthermore, empirical studies [1], [16] have consistently shown that ART outperforms RT with respect to the P-measure. For these reasons, F-measure rather than P-measure has been used for the effectiveness performance metric in the experimental evaluation.

V. Conclusion

Previous empirical studies demonstrated that ART makes better use of the even-spreading concept than random testing and needs fewer test cases to reveal the first failure. In contrast, the computation cost for generating test cases in ART has been reported to be high. We propose a divide-and-conquer technique to address this problem. We make use of the intuition of breaking up a large problem into smaller sub-problems and specify a threshold to limit the computational growth when a large number of previously executed test cases are involved in an ART algorithm. On the other hand, although the use of the threshold hides some of the history information from previously executed test cases, the sub-domains after the bisectional division process can be treated as temporary space to store the past history and further build up an even spreading of test cases. In addition, the divide-and-conquer technique has an equal-number-equal-size characteristic to ensure that each sub-domain is equal in size and contains the same number of test cases, so that every sub-domain has the same probability in revealing the first failure. Consequently, the divide-and-conquer technique can significantly reduce the computational complexity of common ART algorithms such as FSCS and RRT while largely maintaining the effectiveness.

For future work, real-life programs will be utilized to enrich the evaluation. A more comprehensive comparison analysis between divide-and-conquer and other efficiency improvement techniques will also be conducted.

Acknowledgment

This work is supported in part by a linkage grant of the Australian Research Council (project no. LP100200208) and grants of the General Research Fund of the Research Grants Council of Hong Kong (project nos. 717811 and 716612).

References

[1] S. Anand, E. Burke, T.Y. Chen, J. Clark, M.B. Cohen, W. Grieskamp, M. Harman, M.J. Harrold, and P. McMinn, “An orchestrated survey on automated software test case generation,” A. Bertolino, J.J. Li, and H. Zhu, editors/orchestrators, *Journal of Systems and Software*, 2013, doi: 10.1016/j.jss.2013.02.061.

[2] K.P. Chan, T.Y. Chen, and D.P. Towey, “Forgetting test cases,” *Proceedings of the 30th Annual International Computer Software and Applications Conference* (*COMPSAC 06*), vol. 1, IEEE Computer Society, 2006, pp. 485–494.

[3] K.P. Chan, T.Y. Chen, and D.P. Towey, “Restricted random testing: adaptive random testing by exclusion,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 4, pp. 553–584, 2006.

[4] T.Y. Chen, G. Eddy, R.G. Merkel, and P.K. Wong, “Adaptive random testing through dynamic partitioning,” *Proceedings of the 4th International Conference on Quality Software* (*QSIC 04*), IEEE Computer Society, 2004, pp. 79–86.

[5] T.Y. Chen, F.-C. Kuo, and H. Liu, “Adaptive random testing based on distribution metrics,” *Journal of Systems and Software*, vol. 82, no. 9, pp. 1419–1433, 2009.

[6] T.Y. Chen, F.-C. Kuo, R.G. Merkel, and T.H. Tse, “Adaptive random testing: the ART of test case diversity,” *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[7] T.Y. Chen, H. Leung, and I.K. Mak, “Adaptive random testing,” *Advances in Computer Science: Proceedings of the 9th Asian Computing Science Conference* (*ASIAN 04*), Lecture Notes in Computer Science, vol. 3321, Springer, 2004, pp. 320–329.

[8] T.Y. Chen and R.G. Merkel, “An upper bound on software testing effectiveness,” *ACM Transactions on Software Engineering and Methodology*, vol. 17, no. 3, article no. 16, 2008.

[9] M.M. Conroy, A collection of dice problems with solutions and useful appendices, 2013, http://www.matthewconroy.com.

[10] R. Fernndez, J. Frohlich, and A.D. Sokal, *Random Walks, Critical Phenomena, and Triviality in Quantum Field Theory*, Springer-Verlag, 1992.

[11] R. Hamlet, “Random testing,” *Encyclopedia of Software Engineering*, J.J. Marciniak, ed., John Wiley, 2002.

[12] F.-C. Kuo, “An indepth study of mirror adaptive random testing,” *Proceedings of the 9th International Conference on Quality Software* (*QSIC 09*), IEEE Computer Society, 2009, pp. 51–58.

[13] J. Mayer, “Adaptive random testing by bisection with restriction,” *Proceedings of the 7th International Conference on Formal Methods and Software Engineering* (*ICFEM 05*), Springer, 2005, pp. 251–263.

[14] J. Mayer, “Adaptive random testing by bisection and localization,” *Proceedings of the 5th International Conference on Formal Approaches to Software Testing* (*FATES 05*), Springer, 2006, pp. 72–86.

[15] J. Mayer and C. Schneckenburger, “An empirical analysis and comparison of random testing techniques,” *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering* (*ISESE 06*), ACM, 2006, pp. 105–114.

[16] A. Shahbazi, A.F. Tappenden, and J. Miller, “Centroidal voronoi tessellations: a new approach to random testing,” *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 163–183, 2013.