Efficient and Effective Random Testing Using the Voronoi Diagram

T. Y. Chen and Robert Merkel
Faculty of Information and
Communication Technologies,
Swinburne University of Technology
John St., Hawthorn 3122 Australia
{tchen, rmerkel}@ict.swin.edu.au

Abstract

Adaptive Random Testing (ART) is a method for improving the fault-finding effectiveness of random testing. Fixed-Size Candidate Set ART is the most studied variant of this approach. However, existing implementations of FSCS-ART have had substantial selection overhead, with \( n \) test cases requiring \( O(n^2) \) time to generate. We describe the use of a geometric data structure known as the Voronoi Diagram to reduce this overhead to no worse than \( O(n\sqrt{n}) \) and, with further optimization, \( O(n \log n) \). We demonstrate experimentally that practical improvements in selection overhead can be gained using this improved implementation.

Keywords: random testing, adaptive random testing, software testing, Voronoi diagram.

1 Introduction

Random testing - testing programs with randomly selected inputs - is a basic and well-studied testing method. It is particularly useful as a method of assessing the reliability of the program under test. However, it is a less than optimally effective method of exposing errors in the program under test, as no advantage of information such as the program’s likely failure patterns is taken into account. For this reason, a number of methods have been devised to select sequences of tests that are more likely to detect program failures than a random sequence of the same length.

One simple approach is based on the observation that the parts of the program’s input domain in which failures occur - the failure region(s) - are very often contiguous. For example, Ammann and Knight [1] observed the existence of continuous failure regions in a missile launch control program, and Bishop [2] found that all the defects found in a study of real errors in a nuclear reactor trip function were continuous “blobs”. White and Cohen [3] provided an example of how contiguous failure regions can follow from the common program faults described as domain errors, where the conditions deciding the program’s execution path contain a fault. As contiguous failure regions seem to be very common, it will also be commonplace for regions in which failure will not be detected to be contiguous. If a test \( t_0 \) is conducted and does not reveal failure, a test spread away from \( t_0 \) is, in general, more likely to detect a failure than one close to it.

This desirability of achieving “wide spread” leads directly to an enhancement of random testing, termed Adaptive Random Testing (ART)[4]. Several methods based on the ART approach have been proposed [5, 6]. We present here the most straightforward method, the fixed-size candidate set method (FSCS-ART). In essence, to generate a test case, a fixed-size set of test case candidates is randomly generated, and the candidate that is at the furthest distance from previously executed test cases is chosen as the next test case. Distance, in this context, is defined as the distance between the candidate and the nearest previously executed test case. The algorithm is shown in Figure 1.

Chen et al. [4] compared the failure-finding effectiveness of FSCS ART with that of random testing using a sample of 12 programs seeded with various errors. The effectiveness was measured using the metric of the mean number of test cases required to detect the first failure of the program - termed the F-measure [7]. Depending on the program, the F-measure for FSCS-ART was at worst similar to RT and in the majority of programs tested was significantly smaller, in many cases close to 50% smaller.

We have shown mathematically that a testing strategy designed around the idea of contiguous failure regions, without additional information about the locations of such error will have an average F-measure no smaller than 50% of that of random testing [8]. FSCS-ART shows improvements in effectiveness approaching this magnitude. Therefore, rather than attempting to further improve the failure-finding effectiveness of ART, our attention has turned to reducing its computational overhead.
do
set \( M = 0 \)
randomly generate next \( k \) candidates \( c_1, \ldots, c_k \)
for each candidate \( c_n \)
calculate minimum distance \( m \) from
previously executed test cases
If \( m > M \)
set \( M = m \)
\( b = c_n \)
Add \( b \) to list of previously executed test cases
Test program using input \( b \)
while (no failure found)

Figure 1. Fixed-Size Candidate Set ART (FSCS-ART) algorithm

In FSCS-ART, the “find nearest previously executed test case” operation will require each candidate to be compared with each previously executed test case. Therefore, with \( k \) candidates, the overhead to generate the \( i \)th test case is directly proportional to \( ki \), and generating a sequence of \( n \) test cases takes time of the order \( O(kn^2) \). If \( n \) becomes large and the cost of actually running and evaluating the results of a test are small, this selection overhead can quickly become the dominant computational cost of the testing.

We have recently proposed two new methods that take advantage of the same intuition as ART, providing comparable failure-finding effectiveness with significantly lower overheads. One method [6] takes advantage of the concept of mirroring - using the FSCS-ART technique to generate a subset of the desired test case, and then using a relatively cheap method to transform each member of this subset into additional test cases. This method was shown empirically to have similar effectiveness to FSCS-ART, but with much lower overheads. However, the fundamental, expensive comparison operation still remains part of the method.

An alternative approach was to discard the use of distance comparisons entirely and generate candidates using an algorithm that implicitly widely spreads candidates. Quasi-Random Testing [9] takes advantage of a type of sequence known as a “quasi-random sequence”, developed for efficient multidimensional quadrature, that inherently possesses the “even spread” qualities desired. This method has an extremely low selection overhead, but, as the name implies, the sequence of test cases is no longer in any way random, potentially reducing the robustness of the method.

In this paper, we show how an efficient implementation of the “find the nearest” operation can be achieved with an adaptation of an appropriate data structure and algorithm; the Voronoi Diagram. We begin with a discussion of the “post office problem”, a classical problem in multidimensional search. We then show empirically that the incremental construction of the Voronoi Diagram can significantly reduce the overhead of FSCS-ART.

2 The post-office problem

If we make the assumption that a test can be represented in terms of a fixed-length vector, the “find the nearest” problem is essentially the simplest case of the “post-office problem” described by Knuth ([10], p. 555). Knuth described the problem as follows:

Suppose, for example, that we wish to handle queries like “What is the nearest city to point \( x \)?”, given the value of \( x \).

At the time, Knuth noted that “no really nice data structures seem to be available” for this problem, and described a simple approach using a tree representation by McNutt and Pring. Shamos [11] was the first to identify that the Voronoi diagram was an efficient method for solving the post-office problem. The Voronoi diagram is a fundamental data structure in computational geometry. It has been developed independently in various areas of the natural sciences; see [12] for an extensive survey. The following description of the Voronoi diagram is from Bowyer [13]: Consider a set of distinct points \( P = \{P_1, \ldots, P_m\} \) in the plane. For each point, we define the territory of that point as the area of the plane which is closer to that point than any other point in \( P \). The set of resulting territories will form a pattern of convex polygons covering the whole plane. This definition trivially extends to arbitrary numbers of dimensions. Figure 2 shows a Voronoi diagram in the plane for a small number of random points. If all point pairs which share a boundary segment are joined, the result is a triangulation of the convex hull of the points, termed the Delaunay triangulation. The Voronoi and the Delaunay are duals, and one can very straightforwardly be computed from the other.

Given the existence of a Delaunay triangulation of such a point set, Green and Sibson [14] pointed out one straightforward way to locate the closest neighbour of a point - simply perform a walk along the points of the triangulation. They further suggested a point close to the centroid of the point set would be a good place to start. For a random point set in \( \mathbb{R}^d \), such a walk should take \( O\left(\left\lfloor \left(\frac{n}{2}\right)^{\frac{1}{d}} \right\rfloor\right) \) time. There are several ways to post-process a Voronoi diagram to allow a post-office query to be answered in \( O(\log n) \) time.

Shamos [11] published an \( O(n \log n) \) algorithm for constructing a Voronoi diagram in \( \mathbb{R}^2 \). This algorithm uses a classical divide-and-conquer approach, sorting the points in one dimension, constructing a number of small Voronoi diagrams for subsets of points in sections of the range and then
describing a procedure for merging the resulting diagrams. This algorithm is totally unsuitable for the task at hand, as it does not allow for the incremental addition of points to the diagram.

The first incremental algorithm for constructing a Voronoi diagram was described by Green and Sibson [14]. To add a point \( P_n \) to an existing diagram, they first identify the point \( P_c \) nearest to the new point, then make local modifications to the diagram (a constant-time operation). They use the triangulation walk described above to identify \( P_c \). Therefore, their version takes \( O(\sqrt{n}) \) time to add a point to the structure. In Green and Sibson’s tests, they found that this optimisation was unnecessary because the dominant cost was the modifications to the data structure.

Green and Sibson’s primary focus was the actual generation of the Voronoi diagram, which as previously noted is useful for a number of purposes other than searching. As an enhancement specifically to speed up searching, they suggested a simple modification - at regular intervals, duplicate the Delaunay triangulation, and after each duplication retain the older versions unmodified. Then, when performing the walk, start with the version containing the fewest nodes, and when the closest node in the earliest version is found, switch to searching the next version using the closest node found so far as the starting point. This would reduce the cost of a search over random points to \( O(\log n) \), at the cost of the time and memory required to create the duplicate partial triangulations. In our application, multiple searches are conducted for each insertion into the diagram, so the enhancement would offer a larger improvement.

Green and Sibson’s algorithm only works for Voronoi diagrams in the plane. Bowyer [13] and Watson [15] were the first to describe algorithms suitable for use in \( d \geq 3 \) dimensions. Bowyer’s algorithm, in particular, is a reasonably straightforward conceptual extension of the ideas used in Green and Sibson. Using it, it is possible to add the \( n \)th point to the diagram in \( O\left(\frac{n^2}{d}\right) \) time, and find the nearest neighbour amongst \( n \) points in the same order of magnitude time. Again, the non-constant cost comes from the walk through the points, so the same optimisation described for Green and Sibson’s method should reduce the time complexity of searching or adding a random point to \( O(\log n) \).

Incrementally-generated Voronoi diagrams and Delaunay triangulations are used in a number of applications throughout computer science and other disciplines; Zheng and Lee [16] describe the use of this data structure for efficiently answering location-based queries in wireless data services. Aurenhammer [12] provides a comprehensive survey on the topic.

In this paper, we examine whether the improved asymptotic performance of the Voronoi-diagram based search over...
a naive linear search translates into an empirical reduction in ART overhead.

3 Implementing ART using the Voronoi diagram

An implementation of Bowyer’s algorithm is available in the freely downloadable Svis solid modelling library [17]. Amongst many other functions, this library enables the incremental construction of a Voronoi diagram for points in \( \mathbb{R}^3 \), and the use of such for post-office queries. Svis’s implementation does not feature the optimised walk described in section 2, and was clearly not optimised for speed. Several simple implementation modifications (for instance, speeding up array initialisations in inner loops) were made and produced substantial linear speed improvements. The library is implemented in C++.

The naive linear search method was also implemented straightforwardly in C++, using the Svis’s point class and the C++ list implementation to store the data. Svis’s point manipulation and comparison functions were used here, also, to localise the differences between the linear search and the Voronoi method.

All experiments were performed on a machine with an Intel Pentium III processor running at 733 MHz with 512 megabytes of RAM. The operating system used was a development snapshot of Debian GNU/Linux [18], including the Linux kernel version 2.4.18. The Gnu Compiler Collection [19], version 3.2.3, was used to compile Svis and both of the ART implementations. Timing measurements were performed by using the system’s time command.

The candidate test cases were generated using the Mersenne Twister random number generator, as implemented in the GNU Scientific Library [20]. The same random number generator and seed were used for all experiments.

In testing, the time taken to run automated tests is the sum of:

- the time taken to generate the tests,
- the time to execute the program using the generated tests as inputs, and
- the time to evaluate the results of testing.

Of these, only the generation time is affected by the use of the Voronoi method or the linear search method. Therefore, any comparison of the relative efficiency of the methods should be restricted to the time to generate sequences of test cases.

We therefore compared the execution time that our two implementations took to generate specified numbers of test cases. As the Voronoi diagram implementation available worked in \( \mathbb{R}^3 \) without modification, the test cases generated were vectors in \( \mathbb{R}^3 \). All elements in the vectors in the range \([0,1]\).

Figure 3 shows the computation time taken to generate the specified number of test cases using the two methods, and normalised plots of \( n^2 \) and \( n^4 \) for comparison.

4 Discussion and Conclusion

As shown by Figure 3, the FSCS-ART implementation using the Voronoi diagram is clearly faster than the linear search implementation under the testing conditions given, both in terms of the absolute time taken and the asymptotic behaviour displayed. The time increase over the size range trialled was very close to that predicted by the theoretical asymptotic analysis. Even the suboptimal \( O\left(n^{1+\frac{1}{d}}\right) \) complexity of the implementation tested provides substantial performance improvements over the quadratic algorithm, such as to make practical test sequences of a length that would not otherwise be feasible.

This study did not cover the effect of the number of dimensions of the test vectors on the relative overheads of the method. The constant factors in the algorithm depend mainly on the number of vertices per point in the Voronoi diagram, which clearly increases swiftly in higher dimensions. In fact, the most expensive operation in the current implementation takes time proportional to the square of the number of vertices per node. Bowyer [13] states that for the two-dimensional case, the Euler-Poincare formula shows that the mean number of vertices per point will be about 6. Bowyer also quotes an upper bound from Miles for the three-dimensional case of 27.07, and notes that “Unfortunately there is, as yet, no general expression for the number in \( k \) dimensions.” Clearly, however, the effect of increased dimensionality will limit the range of problems to which the enhancements may be profitably applied. Detailed tuning of the implementation could significantly reduce other linear-time constant factors, and is currently under investigation. If such speedups are obtained, the input space dimensionality at which increased linear constant factors outweigh the asymptotic speedup may be increased. An alternative, reportedly faster, and freely downloadable open source implementation is CGAL [21].

The optimisation of the Voronoi walk would be particularly advantageous for sequences in \( \mathbb{R}^2 \), as the walk is actually more computationally expensive for those sequences compared with higher-dimensional ones. A more efficient implementation would also decrease the number of tests in the sequence for which the Voronoi method empirically outperforms the linear search method.

Bowyer’s method is not the only method for generating the necessary Delaunay triangulations available. Watson
[15] describes another equivalent method, and it would be worthwhile to compare the performance of the two in our application. Green and Sibson’s method is actually recommended by Bowyer as more suitable for applications in $\mathbb{R}^2$. Additionally, there exists another major method for solving the post-office problem using a data structure termed a Randomised Post Office tree (usually abbreviated to simply RPO tree) [22]. Whilst having worse asymptotic performance than the Voronoi algorithms in the construction of the data structure, Clarkson claims that for many distributions of points their performance is likely considerably better than the worst case bound. Clarkson’s algorithm, as described in the reference, is not designed for incremental insertion of new data. Like most search tree algorithms without explicit balancing, it requires the insertion of data in randomised order for performance to be close to optimal. Therefore, Clarkson requires that the input order of data should be randomly permuted before the data structure is built. With incremental insertion, this is not possible. However, with appropriate choices of supplementary algorithms (the nodes of an RPO tree are themselves Voronoi diagrams), an RPO tree could be built incrementally, though with no guarantee of efficiency. In our application, the data will be inserted in an order that may be sufficiently close to random to enable good performance.

In any case, we have demonstrated that, with the aid of the Voronoi diagram data structure, FSCS-ART can be used to generate sequences of test cases more efficiently than previously known. We recommend its use if the overhead of test case generation presently poses a barrier to applying ART.

Acknowledgements

This project was partially supported by an Australian Research Council Discovery Project (DP0557246).

References


