Exploratory Datamorphic Testing of Classification Applications

Hong Zhu and Ian Bayley
Oxford Brookes University
Oxford OX33 1HX, UK
(hzhu|ibayley)@brookes.ac.uk

ABSTRACT

Testing has been widely recognised as difficult for AI applications. This paper proposes a set of testing strategies for testing machine learning applications in the framework of the datamorphism testing methodology. In these strategies, testing aims at exploring the data space of a classification or clustering application to discover the boundaries between classes that the machine learning application defines. This enables the tester to understand precisely the behaviour and function of the software under test. In the paper, three variants of exploratory strategies are presented with the algorithms as implemented in the automated datamorphic testing tool Morphy. The correctness of these algorithms are formally proved. The paper also reports the results of some controlled experiments with Morphy that study the factors that effect the test effectiveness of the strategies.

CCS CONCEPTS

• **Software and its engineering → Software notations and tools**; • **Computing methodologies → Artificial intelligence**; **Machine learning**.

KEYWORDS

Artificial intelligence, Software testing, Automated software testing, Test method, Testing tools, Datamorphic testing, Exploratory testing, Test strategies

ACM Reference Format:
Hong Zhu and Ian Bayley. 2020. Exploratory Datamorphic Testing of Classification Applications. In *Proceedings of AST '20: The First International Conference on Automation of Software Test (AST '20).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122334.1122456

1 INTRODUCTION

It is widely recognised that the generation of test data for AI applications is prohibitively expensive [11]. Checking the correctness of a test result is also notoriously difficult, if not completely impossible [7, 15]. Moreover, existing testing techniques for measuring test coverage and the automation of testing activities and processes are not directly applicable [20]. Testing AI applications is therefore a grave challenge for software engineering [2]. Developing novel approaches to test AI applications is highly desirable [4].

In [20, 21], we proposed a method called *datamorphic testing* for testing AI applications and reported a case study with face recognition applications. The method is further developed in [17, 18], in which the notion of test morphisms is introduced, an automated testing tools called *Morphy* is reported and a set of test strategies are formally defined and implemented. The case studies reported in [17] shows that test strategies can significantly improve the automation for testing AI applications. This paper presents another set of datamorphic test strategies specifically designed for testing the classification and clustering type of AI applications, which classify objects and entities according to their features and attributes. Classification and clustering are one of the largest categories of AI applications, and many AI techniques such as machine learning and data analytics generate applications of this category [1, 5, 8].

The proposed test strategies are based on the idea of exploratory testing in which testers interact with the application and use the information the application provides to change the course of testing in order to explore the application's functionality [12]. It is different from testing for verification and validation, which aims to confirm the correctness of the software under test with respect to a given specification. In contrast, exploratory testing treats the software under test as an object unknown and regards software testing as a series of experiments with the software aimed at discovering its functions and features. Moreover, the traditional verification and validation testing methods regard test cases as independent from each other. In contrast, exploratory testing uses the result of the previous test cases to guide its choice of the next test case in order to maximise its effectiveness in the process of searching for useful information.

The paper is organised as follows. Section 2 briefly reviews the datamorphic testing method, the automated testing tool Morphy, and the basic concepts of classification applications. Section 3 defines a set of three exploration strategies and illustrate their use with an example. Section 4 reports the experiments with these strategies on their performances. Section 5 concludes the paper with a discussion of related work and future work.

2 PRELIMINARIES

In this section, we briefly review the datamorphic testing method and the testing tool Morphy, and clustering and classification techniques to set the context of the paper.

2.1 Overview of Datamorphic Testing Method

In the datamorphic software testing method [17], software artefacts involved in testing are classified into two types: *entities* and *morphisms*.

AST '20, May 23–29, 2020, Seoul, South Korean

ACM ISBN 978-1-4503-XXXX-X/20/05…$15.00
https://doi.org/10.1145/1122334.1122456

Test entities are objects and data that are used and/or generated in testing. These include test cases, test suites/sets, the programs under test, and test reports, etc.

Test morphisms are mappings between entities. They generate and transform test entities to achieve testing objectives. They can be implemented as test code and invoked to perform test activities and composed to form test processes. The following are the test morphisms recognised by the datamorphic test tool *Morphy*.

- *Test set creators* create sets of test cases. They are called *seed test case makers* in [16, 21]. A typical example is random test case generators like fuzzers [10].
- *Datamorphisms* are mappings from existing test cases to new test cases. They are called data mutation operators in [9].
- *Metamorphisms* are mappings from test cases to Boolean values that assert their correctness. They are test oracles. Formal specifications and metamorphic relations in metamorphic testing [3, 7] can also be used as metamorphisms.
- *Test case metrics* are mappings from test cases to real numbers. They measure test cases giving, for example, the similarity of a test case to the others in the test set.
- *Test case filters* are mappings from test cases to truth values. They can be used, for example, to decide whether a test case should be included in a test set.
- *Test set metrics* are mappings from test sets to real numbers. They measure the test set quality, such as its code coverage [19].
- *Test set filters* are mappings from test sets to test sets. For example, they may remove some test cases from a test set for regression testing.
- *Test executers* execute the program under test on test cases and receive the outputs from the program. They are mappings from a piece of program to a mapping from input data to output. That is, they are functors in category theory.
- *Test analysers* analyse test sets and generate test reports. Thus, they are mappings from test sets to test reports.

A *test system* $\mathscr{T} = \langle \mathscr{E}, \mathscr{M} \rangle$ in datamorphic testing consists of a set $\mathscr{E}$ of test entities and a set $\mathscr{M}$ of test morphisms. In Morphy [17], a test system is specified as a Java class that declares a set of attributes as test entities and a set of methods as test morphisms.

Given a test specification, Morphy provides testing facilities to automate testing at three different levels. At the lowest level, various test activities can be performed by invoking test morphisms via a click of buttons on Morphy's GUI. At the medium level, Morphy implements various test strategies to perform complex testing activities through combinations and compositions of test morphisms. At the highest level, test processes are automated by recording, editing and replaying test scripts that consist of a sequence of invocations of test morphisms and strategies.

Test strategies are complex combinations of test morphisms designed to achieve test automation. Three sets of test strategies have been implemented in Morphy:

- *Mutant combination*: combining datamorphisms to generate mutant test cases; see [17].
- *Domain exploration*: searching for the borders between clusters/subdomains of the input space;
- *Test set optimisation*: optimising test sets by employing genetic algorithms.

This paper focuses on domain exploration strategies, which will be defined in Section 3. Those strategies that employ genetic algorithms to optimise test sets will be reported in another paper.

2.2 Classification Applications

Clustering as a data mining and machine learning problem is the partitioning of a given set of data points into groups containing similar data points. However, clustering does not only partition the data in the given data set, but also makes it possible to put new data into the right groups. The key concept of clustering is similarity between data points, which is defined formally in the form of a similarity or distance function on the data space. Two pieces of data that are similar to each other should be put into the same group, while the data that are dissimilar should be placed in different groups. Whereas clustering is unsupervised learning, classification is supervised learning. Given a number of examples of data points and their classifications, it learns how to assign data to groups [1, 5, 8].

In both clustering and classification, the result is a program P that maps from the data space D into a number of groups G. We say that P is a *classification application*. We will write $P(x)$ to denote the output of P on an input $x \in D$, and call $P(x)$ the classification of x by P. We also assume that there is a function $dist : D \times D \to \mathbb{R}^+$ measuring the distances between any two points x and y in the data space D such that:

- $\forall x \in D(dist(x, x) = 0)$;
- $\forall x, y \in D(dist(x, y) \geq 0)$;
- $\forall x, y \in D(dist(x, y) = dist(y, x))$;
- $\forall x, y, z \in D(dist(x, y) + dist(y, z) \geq dist(x, z))$.

The distance function measures the similarity between data points in that the smaller the distance between two points the more similar they are.

For a classification program, it is crucial to classify data into correct classes. However, the borders between classes are often unknown if the classification program is obtained through machine learning and data mining. The goal of the exploration strategies proposed in this paper is to find a set of data pairs that represents the borders between classes. Thus, we introduce the notion of *Pareto front* of the classification as defined by the program P under test.

Definition 1. *(Pareto Front of Classification)*
Let $P : D \to G$ be a classification program, $dist : D \times D \to R$ be a distance metric defined on the input space D, and $\delta > 0$ be a given real number. A set $\{< a_i, b_i > | a_i, b_i \in D, i = 1, \cdots, n\}$ of data pairs is a Pareto front *of the classes of D according to P with respect to $dist$ and δ, if for all $i = 1, \cdots, n, P(a_i) \neq P(b_i)$ and $dist(a_i, b_i) \leq \delta$.* □

A Pareto front can show accurately the borders between the classes, thus help testers to determine whether the classification is correct or not.

2.3 Exploratory Test Systems

To apply an exploratory test strategy to a classification program $P : D \to G$ with a distance function $dist$, we assume that the test system $\mathscr{T} = \langle \mathscr{E}, \mathscr{M} \rangle$ has the following properties.

(1) The set $\mathscr{M}$ of morphisms contains a test executer $Exe_P(x)$ that executes the program P under test on a test case x and receives the output of P; that is $Exe_P(x) = P(x)$. In the sequel, we will write $P(x)$ for $Exe_P(x)$ for the sake of simplicity.
(2) There is a set $W \subseteq \mathscr{M}$ of unary datamorphisms defined on D. Informally, for each $w \in W$ and $x \in D$, $w(x), w^2(x), \cdots, w^n(x)$ generates a sequence of different data points in D, where $w^1(x) = w(x)$, $w^{n+1}(x) = w(w^n(x))$. These datamorphisms are called *traversal methods*.
(3) There is also a binary datamorphism $m \in \mathscr{M}$ such that for all $x, y \in D$, $dist(x, z) < dist(x, y)$ and $dist(y, z) < dist(x, y)$, where $z = m(x, y) \in D$. Informally, the datamorphism m calculates a point between x and y. It is called the *midpoint method*.

Note that, for all $x, y \in D$ and $z = m(x, y)$, we have:

$$(P(x) \neq P(y)) \Rightarrow (P(x) \neq P(z)) \vee (P(y) \neq P(z)). \quad (1)$$

Informally, if the program P under test classifies x and y into different classes, the midpoint between x and y must be either not in the same class as x or not in the same class as y.

2.4 The Running Example

In Section 3, we will use the following simple classification program as a running example to illustrate the exploration strategies. It classifies the points in a two-dimensional continuous space $[0, 2\pi] \times [-1, 1]$ into three classes: *red, black* and *blue* as illustrated in Figure 1. In this example, data points x and y is a pair of Pareto Front between *black* and *red* classes, if x is *red* and y is *black* and they are very close to each other. Such pairs can show accurately the borders between the classes, thus help testers to determine whether the classification is correct or not.

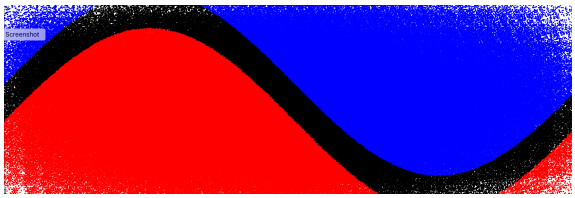


Figure 1: Data Space of the Running Example

The Figure 2 gives the traversal and midpoint methods in the Morphy test specification. The midpoint method $mid(x, y)$ calculates the geometric midpoint between x and y.

It is easy to see that the running example forms an exploratory test system with the following distance function.

$$Eucl(\langle x_1, x_2\rangle, \langle y_1, y_2\rangle) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \quad (2)$$

3 EXPLORATION STRATEGIES

This section presents the algorithms for three different exploratory strategies for testing clustering and classification applications. We also prove their correctness and illustrate their behaviour by using the running example given in the previous section.

```
@Datamorphism
public TestCase<TwoD, Colour> upward(TestCase<TwoD, Colour> seed){
    TestCase<TwoD, Colour> mutant = new TestCase<TwoD,Colour>();
    TwoD point = new TwoD(seed.input.x, seed.input.y + 0.2);
    mutant.input = point;
    return mutant;
}

@Datamorphism
public TestCase<TwoD, Colour> downward(TestCase<TwoD, Colour> seed){
    TestCase<TwoD, Colour> mutant = new TestCase<TwoD,Colour>();
    TwoD point = new TwoD(seed.input.x, seed.input.y - 0.2);
    mutant.input = point;
    return mutant;
}

@Datamorphism
public TestCase<TwoD, Colour> leftward(TestCase<TwoD, Colour> seed){
    TestCase<TwoD, Colour> mutant = new TestCase<TwoD,Colour>();
    TwoD point = new TwoD(seed.input.x-0.2, seed.input.y);
    mutant.input = point;
    return mutant;
}

@Datamorphism
public TestCase<TwoD, Colour> rightward(TestCase<TwoD, Colour> seed){
    TestCase<TwoD, Colour> mutant = new TestCase<TwoD,Colour>();
    TwoD point = new TwoD(seed.input.x+0.2, seed.input.y);
    mutant.input = point;
    return mutant;
}

@Datamorphism
public TestCase<TwoD, Colour> mid(TestCase<TwoD, Colour> x1,
        TestCase<TwoD, Colour> x2){
    TestCase<TwoD, Colour> mutant = new TestCase<TwoD, Colour>();
    TwoD point = new TwoD((x1.input.x + x2.input.x)/2,
            (x1.input.y + x2.input.y)/2);
    mutant.input = point;
    return mutant;
}
```

Figure 2: Datamorphisms of The Running Example

3.1 Random Target Strategy

Let's start with a simple exploration strategy based on random selection of known test cases in order to find the Pareto front of the classification groups between these two test cases. We call this strategy *random target strategy*.

The strategy starts by selecting a pair of two test cases x and y at random. If the outputs of the program P under test on these test cases are different, i.e. $P(x) \neq P(y)$, then a point z_1 between x and y are generated by using the binary datamorphism of the midpoint method $mid(x, y)$, i.e. $z_1 = mid(x, y)$. The program P is executed on this mutant test case z_1 to classify it. The classification of z_1 must be different from one of the original pair of test cases; say $P(z_1) \neq P(x)$. Thus, we can repeat the above steps with x and z_1 as the pair of test cases, and a further mutant z_2 can be generated. This process is iterated for a number of times to ensure the distance between the final pair of points is small enough. See Algorithm 1.

Let $n > 0$ be any given natural number. We write $RT(n) = \langle a, b\rangle$ to denote the results of executing Algorithm 1 with n as the parameter *steps* and $\langle a, b\rangle$ as the output.

Assume that the exploratory test system has the following properties.

(1) There is a constant $c > 1$ such that

$$\forall x, y \in D. \left(\frac{Max\{dist(x, z), dist(z, y)\}}{dist(x, y)}\right) \leq 1/c, \quad (3)$$

where $z = mid(x, y)$.
(2) There is a constant $d_m > 0$ such that

$$\forall x, y \in D.(dist(x, y) \leq d_m). \quad (4)$$

Then, we have the following theorem about the correctness of the random target strategy algorithm.

Algorithm 1 (Random Target Strategy)

Input:
testSet: Test Pool;
steps: Integer;
$mid(x, y)$: Binary datamorphism;
Output:
a, b: Test Case;
Begin
1: Select two different test cases x and y in *testSet* at random;
2: Execute program P on test cases x and y;
3: Check if a pair of Pareto front exits between x to y:
if $(x.output = y.output)$ **then return** $\langle null, null\rangle$
end if
4: Refinement:
for $i \leftarrow 1$ to *steps* **do**
$z = mid(x, y)$;
if $(x.output \neq z.ouptut)$ **then** $y = z$
else $x = z$;
end if
end for;
$a = x$; $b = y$;
return $\langle a, b\rangle$;
End

Theorem 1. *If $RT(n) = \langle a, b\rangle \neq \langle null, null\rangle$, then $\langle a, b\rangle$ is a pair of Pareto front according to P with respect to dist and δ, if $d_m/c^n < \delta$.*

Proof.

If $RT(n) = \langle a, b\rangle \neq \langle null, null\rangle$, then, the condition of the If-statement in step (3) is **false**. Thus, the loop is executed. It is easy to see that the For-loop in *Step 4* in the algorithm terminates.

We now proof that the following is a loop invariant of the loop by induction on the number i of iterations of the loop body.

$$dist(x, y) \leq \frac{d_m}{c^i} \wedge P(x) \neq P(y).$$

When entering the loop, by assumption (4), the distance between the data points stored in variable x and y satisfies the following inequality.

$$dist(x, y) \leq d_m$$

Since the condition of the If-statement is false, we have that

$$P(x) = x.output \neq y.output = P(y).$$

Therefore, the loop invariant is true for $i = 0$.

Assume that the loop invariant is true for $i = n \geq 0$.

After the excution of the loop body one more time (i.e. $i = n + 1$), by applying the Hoare logic of the If-statements in the loop body, the distance d'_x between the data points stored in variables x and y will become either $dist(x, z)$ or $dist(z, y)$, where $z = mid(x, y)$. By assumption (3), in both cases we have that

$$d'_x \leq Max\{dist(x, z), dist(z, y)\} \leq dist(x, y)/c \leq d_m/c^{n+1}.$$

By the condition of the If-statement in the loop body and the property (1), applying Hoare logic we have that, after the execution of the loop body, the data points stored in variables x and y have the property that $P(x) \neq P(y)$. Therefore, the condition is a loop invariant according to Hoare logic.

When the loop exits, $i = steps = n$. By Hoare logic, after executing the assignment statements $a = x$ and $b = y$, we have that

$$dist(a, b) \leq d_m/c^n \wedge P(a) \neq P(b).$$

Therefore, the theorem is true by Definition 1. □

The algorithm of random target strategy can be run multiple times to generate a number of pairs for the Pareto front.

Example 1. *For example, applying the random target strategy to the running example, we can obtain a test set shown in Figure 3 when 1000 pairs of test cases are selected at random from a test set of 300 random test cases. A total of 641 pairs of Pareto front test cases were generated. The success rate in generating a pair for the Pareto front is 64.1%. The set of Pareto front pairs shows clearly the boundary between the subdomains classified by the software.*

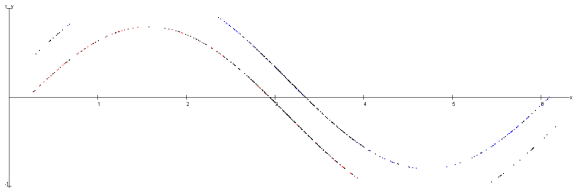


Figure 3: Pareto Front Generated by Random Target

In this example, the number of steps n is 20. Since the data space $D = [0, 2\pi] \times [-1, 1]$, if the distance function $dist(x, y)$ is $Eucl(x, y)$, we have that $d_m = 2\sqrt{\pi^2 + 1}$. By the definition of $mid(x, y)$, we have that

$$\frac{Max(\{dist(x, z), dist(y, z)\})}{dist(x, y)} = 1/2.$$

So, $c = 2$. By Theorem 1, for the distance δ between each pair in the Pareto front, we have that

$$\delta \leq \frac{d_m}{c^{20}} = \frac{\sqrt{\pi^2 + 1}}{2^{19}}.$$

Note, the distance between the test cases in each pair of Pareto front is so small that they are not visually distinguishable in Figure 3. □

3.2 Directed Walk Strategy

A variation of the random target strategy is to start with one test case (rather than a pair) and apply a unary datamorphism repeatedly until a test case of different classification is found. Then, the Pareto front between these two test cases is searched for in the same way as for the random target strategy. In this strategy, the unary datamorphism (i.e. a mutation operator) is the traversal method. The repeated application of the mutation operator makes a 'walk' in one direction until a test case in a different class is found or gives up the exploration if we have gone too far (i.e. too many iterations).

Note that, a walk in one direction may not be able to find a data point in a different class. In that case, the algorithm returns $\langle null, null\rangle$. Let $m, n > 0$ be any given natural numbers. We write $DW(m, n) = \langle a, b\rangle$ to denote the results of executing Algorithm 2 with m as the walking distance and n as the number of *steps* and $\langle a, b\rangle$ as the output. Assume that the exploratory test system satisfies assumption (3) and has the following property.

Algorithm 2 (Directed Walk)

Input:
TestSet: test set;
walkDistance: integer;
steps: Integer;
$d(x)$: Unary datamorphism;
$mid(x, y)$: Binary datamorphism;
Output:
a, b: Test Case;
Begin
1: Select a test cases x in *testSet* at random;
2: Execute program P on test case x;
3: Walk in one direction as follows:
Bool found = **false**;
for $i \leftarrow 1$ to *walkingDistance* **do**
$y = d(x)$;
Execute software on test case y;
if ($x.output \neq y.output$) **then**
$found$ = **true**; break;
else $x = y$;
end if
end for
4: Check if a Pareto front can be found:
if ($\neg found$) **then return** $\langle null, null\rangle$;
end if
5: Refinement
for $i \leftarrow 1$ to *steps* **do**
$z = mid(x, y)$;
if ($x.output \neq z.ouptut$) **then** y = z;
else x = z;
end if;
end for
$a = x$; $b = y$;
return $\langle a, b\rangle$;
End

There is a constant $d_s > 0$ such that

$$\forall x \in D.\, (dist(x, d(x)) \leq d_s)\,. \tag{5}$$

where d_s is called the step size of the traversal method $d(x)$. Then, we have the following correctness theorem for the directed walk algorithm.

Theorem 2. *If $DW(m, n) = \langle a, b\rangle \neq \langle null, null\rangle$, then, $\langle a, b\rangle$ is a pair in the Pareto front according to P with respect to $dist$ and δ, if $d_s/c^n < \delta$, where n is the number of steps.*

Proof. If $DW(m, n) = \langle a, b\rangle \neq \langle null, null\rangle$, then the condition of the If-statement in step (4) is **false**. Thus, the For-loop of Step (5) is executed. It is easy to see that the For-loop in *Step 5 Refinement* in the algorithm terminates.

Similar to the proof of Theorem 1, by the definiton of d_s and assumption (5), the following is a loop invariant of the loop by induction on the number i of iterations of the loop body.

$$dist(x, y) \leq \frac{d_s}{c^i} \wedge P(x) \neq P(y).$$

When the loop exits, $i = steps = n$. By Hoare logic, after executing the assignment statements $a = x$ and $b = y$, we have that

$$dist(a, b) \leq d_s/c^n \wedge P(a) \neq P(b).$$

Therefore, the theorem is true by Definition 1. □

Example 2. *For example, starting from 1000 random test cases using the directed walk strategy with the $upward(x)$ datamorphism as the unary traversal method, a set of 161 pairs of Pareto front were generated; shown in Figure 4. The set of Pareto front pairs also shows clearly parts of the boundaries between classes. The success rate of finding a pair of Pareto front on one test case is 16.1%.*

In this example, the number n of steps is also 20. By the definition of $upward(x)$ traversal method, we have that $d_s = 0.2$, if the distance function $dist(x, y)$ is $Eucl(x, y)$. As in Example 1, by the definition of $mid(x, y)$, we have that $c = 2$. By Theorem 2, for the distance δ between each pair of Pareto front, we have that

$$\delta \leq \frac{d_s}{c^{20}} = 0.2 \times \frac{1}{2^{20}}.$$

Again, the distance between the test cases in each pair of Pareto front is so small that they are not visually distinguishable, so they appear as one dot in Figure 4. □

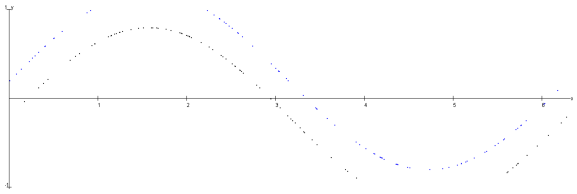


Figure 4: Pareto Fronts Generated by Directed Walk

3.3 Random Walk Strategy

If multiple traversal methods are available, a random walk can be performed by selecting the direction of the next step at random. This is similar to the random walk testing in hyperlink/web GUI test. The algorithm is given below.

We write $RW(m, n) = \langle a, b\rangle$ to denote the results of executing Algorithm 3 with m as the walking distance and n as the *steps* and $\langle a, b\rangle$ as the output. Assume that the exploratory test system satisfies assumption (3) and has the following property. There is a constant $d_s > 0$ such that

$$\forall x \in D. \forall d_i \in W. (dist(x, d_i(x)) \leq d_{sm}). \tag{6}$$

where d_{sm} is called the maximal step size of the traversal methods $d_i(x) \in W$. Then, we have the following correctness theorem for the algorithm of random walk strategy.

Theorem 3. *If $RW(m, n) = \langle a, b\rangle \neq \langle null, null\rangle$, then, $\langle a, b\rangle$ is a pair of Pareto front according to P with respect to $dist$ and δ, if $d_{sm}/c^n < \delta$, where n is the steps.*

Proof. If $RW(m, n) = \langle a, b\rangle \neq \langle null, null\rangle$, then, the condition of the If-statement in step (4) is **false**. Thus, the For-loop of Step (5) is executed. It is easy to see that the For-loop in *Step 5 Refinement* in the algorithm terminates.

Algorithm 3 (Random Walk Strategy)

Input:
testSet: Test Set;
walkingDistance: Integer;
steps: Integer;
$d_1(x), \cdots, d_k(x)$: Unary datamorphism; $k > 1$
$mid(x, y)$: Binary datamorphism;
Output:
a, b: Test Case;
Begin
1: Select a test case x in *testSet* at random;
2: Execute program P on test case x;
3: Walking at random to search for test case in a different class:
Bool $found$ = **false**;
for $i \leftarrow 1$ to *walkingDistance* **do**
Get a random integer r in the range $[1, k]$
$y = d_r(x)$;
Execute program P on test case y;
if $(x.output \neq y.output)$ **then**
$found$ = **true**; **break**;
else x=y;
end if
end for
4: Check if a Pareto front can be found:
if $(\neg found)$ **then return** $\langle null, null \rangle$;
end if
5: Refinement:
for $i \leftarrow 1$ to *steps* **do**
$z = mid(x, y)$;
if $(x.output \neq z.ouptut)$ **then** $y = z$;
else x = z;
end if
end for
$a = x$; $b = y$;
return $\langle a, b \rangle$;
End

Similar to the proof of Theorem 1, by the definiton of d_{sm} and assumption (6), we can prove that the following is a loop invariant of the loop by induction on the number i of iterations of the loop body.

$$dist(x, y) \leq \frac{d_{sm}}{c^i} \wedge P(x) \neq P(y).$$

When the loop exits, $i = steps = n$. After executing the assignment statements $a = x$ and $b = y$, the following is true by Hoare logic.

$$dist(a, b) \leq d_{sm}/c^n \wedge P(a) \neq P(b).$$

Therefore, the theorem is true by Definition 1. □

Example 3. *For example, by applying the random walk strategy on a test set containing 300 random test cases, 1000 random walks generated 805 pairs of Pareto front test cases shown in Figure 5, where the walking distance was 20 steps.*

In this example, the number n of steps is also 20. By the definition of upward(x), downward(x), leftward(x) and rightward(x) traversal methods, we have that $d_s = 0.2$, if the distance function $dist(x, y)$ is $Eucl(x, y)$. As in Example 1 and 2, by the definition of $mid(x, y)$, we have that $c = 2$. By Theorem 3, the distance δ between each pair of Pareto front satisfies the following inequality.

$$\delta \leq \frac{d_s}{c^{20}} = 0.2 \times \frac{1}{2^{20}}. \quad \square$$

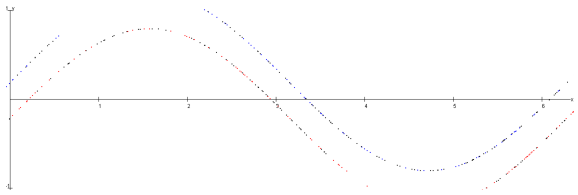


Figure 5: The Pareto Fronts Generated by Random Walk

4 EXPERIMENTS

Controlled experiments with the exploratory test strategies have been conducted using the automated datamorphic testing tool Morphy to study their test effectiveness. This section report the results of the experiments.

4.1 Design of the Experiments

4.1.1 Objectives of the Experiments. As discussed in the previous sections, exploration strategies are designed to test classification applications. They aim to find the borders between subdomains of the classifications. The goal of the experiments is to study the factors that have effect on the effectiveness of these test strategies in terms to their capability of finding the Pareto fronts between subdomains. The measurement of test effectiveness is the number of test executions per border points found by the test strategy.

It is worth noting that the experiments are not for comparison of the strategies, which each has its own suitable applications.

4.1.2 Subject applications. The experiments are carried out with ten classification applications shown in Figure 6. These applications are on the same input domain, i.e. two-dimensional real numbers in the range of $[0, 2\pi] \times [-1, 1]$.

4.2 Experiment process and the results

For each subject application, three exploration strategies are used with various parameters. Each test is repeated for 10 times using the testing tool Morphy and the average of the data is used to analyse the results.

4.2.1 Experiments with the directed walk strategy. The experiments used various numbers of random test cases from 200 to 1200 as shown in Table 1; here, the column *#Seed TCs* is the number of seed test cases in the experiment. These seed test cases are generated at random from the uniform distribution. From each seed test case, one walk in one direction is made for up to 20 steps. The experiments used the upward datamorphism. The column *Avg #Runs* in Table 1 gives the average number of test executions of the subject program under test. The column *Avg #mutant TC* gives the average number of mutant test cases generated; these are test cases on the borders of the clusters.

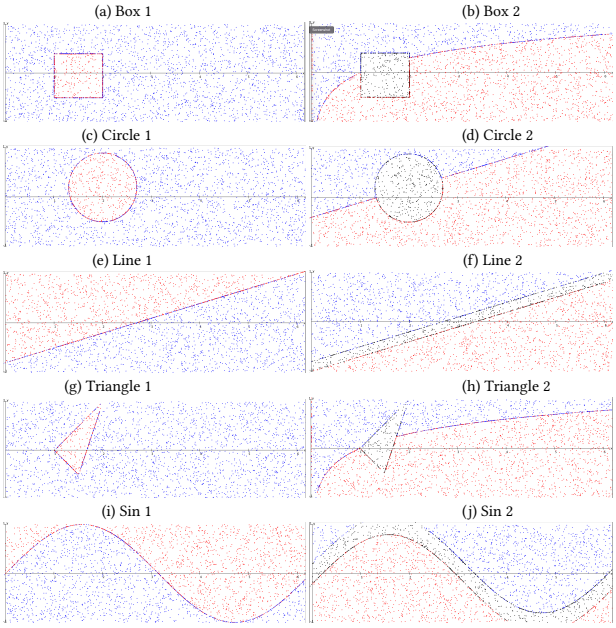


Figure 6: Illustration of the sample applications

Table 1: Experiments Date of The Directed Walk Strategy

Subject	#Seeds (=#Walks)	Avg #Runs	Avg #Mutants	Avg #Runs/Mutant	Subject	#Seeds (=#Walks)	Avg #Runs	Avg #Mutants	Avg #Runs/Mutant
Box 1	200	4205.70	11.40	368.92	Box2	200	4223.40	46.80	90.24
	400	8413.80	27.60	304.85		400	8442.20	84.40	100.03
	600	12620.80	41.60	303.38		600	12668.80	137.60	92.07
	800	16827.60	55.20	304.85		800	16891.40	182.80	92.40
	1000	21033.40	66.80	314.87		1000	21108.00	216.00	97.72
	1200	25236.70	73.40	343.82		1200	25339.80	279.60	90.63
Circle 1	200	4207.50	15.00	280.50	Circle 2	200	4218.20	36.40	115.88
	400	8416.40	32.80	256.60		400	8442.20	84.40	100.03
	600	12624.50	49.00	257.64		600	12657.70	115.40	109.69
	800	16835.60	71.20	236.46		800	16883.90	167.80	100.62
	1000	21046.90	93.80	224.38		1000	21102.50	205.00	102.94
	1200	25255.30	110.60	228.35		1200	25319.70	239.40	105.76
Line 1	200	4221.20	42.40	99.56	Line 2	200	4237.80	75.60	56.06
	400	8437.00	74.00	114.01		400	8476.80	153.60	55.19
	600	12657.60	115.20	109.88		600	12712.00	224.00	56.75
	800	16877.50	155.00	108.89		800	16956.20	312.40	54.28
	1000	21099.60	199.20	105.92		1000	21188.80	377.60	56.11
	1200	25312.00	224.00	113.00		1200	25426.20	452.40	56.20
Sin 1	200	4216.90	33.80	124.76	Sin 2	200	4233.90	67.80	62.45
	400	8435.00	70.00	120.50		400	8465.10	130.20	65.02
	600	12651.70	103.40	122.36		600	12698.80	197.60	64.27
	800	16869.40	138.80	121.54		800	16927.10	254.20	66.59
	1000	21088.20	176.40	119.55		1000	21160.00	320.00	66.13
	1200	25300.90	201.80	125.38		1200	25398.20	396.40	64.07
Triangle 1	200	4205.20	10.40	404.35	Triangle 2	200	4221.60	43.20	97.72
	400	8411.70	23.40	359.47		400	8444.20	88.40	95.52
	600	12618.50	37.00	341.04		600	12672.50	145.00	87.40
	800	16822.80	45.60	368.92		800	16888.70	177.40	95.20
	1000	21028.90	57.80	363.82		1000	21112.70	225.40	93.67
	1200	25232.80	65.60	384.65		1200	25341.40	282.80	89.61

The experimental data shows that the number of mutant test cases (i.e. the pairs of test cases in the Pareto front) generated by using the directed walk strategy increases linearly with the number of walks; see Figure 7. Similarly, the number of test executions is also linear with respect to the number of walks. In Figure 7, the X axis is the number of random seed test cases, which equals number of walks, and the Y axis of (a) and (b) are the average numbers of mutant test cases and test executions, respectively. The average numbers of test executions on various subject programs are so close to each other that they are not visually separable in Figure 7(a).

The test effectiveness is measured in term of the number of test executions per mutant test case generated. It is fairly invariant for each subject while the number of random seed test cases varies

Figure 7: Results of The Directed Walk Strategy

from 200 to 1200; see Figure 7(c), where the Y axis is the average test effectiveness. The experiment data also show that the test effectiveness varies significantly for different subject programs; see Figure 7(d), which gives the overall average effectiveness of testing various subject programs.

4.2.2 Experiments with the random walk strategy. There are two parameters in the random walk strategy: (1) the number of seed test cases, and (2) the number of walks starting from the seed test cases. Two sets of experiments were designed and conducted. The first is with a fixed number of seed test cases (200 test cases) but variable numbers of random walks (range from 200 to 1200). The second is with a fixed number of random walks (800 walks) but variable numbers of random seeds (range from 200 to 1200).

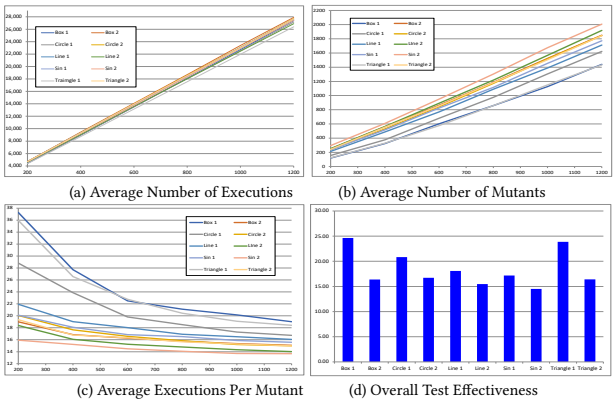
Table 2 gives the result data of the first set of experiments.

Table 2: Experiments Data of The Random Walk Strategy

Subject	#Walks	Avg #Runs	Avg #Mutants	Avg #Runs/Mutant	Subject	#Walks	Avg #Runs	Avg #Mutants	Avg #Runs/Mutant
Box 1	200	4429.40	118.80	37.28	Box2	200	4704.40	247.60	19.00
	400	8950.60	323.00	27.71		400	9457.20	561.00	16.86
	600	13526.00	601.20	22.50		600	14067.60	864.60	16.27
	800	18126.60	858.00	21.13		800	18691.30	1186.20	15.76
	1000	22706.50	1126.60	20.15		1000	23397.70	1523.40	15.36
	1200	27386.00	1439.80	19.02		1200	27900.20	1850.60	15.08
Circle 1	200	4484.60	155.60	28.82	Circle 2	200	4735.40	236.00	20.07
	400	8976.00	376.00	23.87		400	9330.30	528.40	17.66
	600	13491.80	681.00	19.81		600	13939.40	841.40	16.57
	800	18069.60	975.60	18.52		800	18551.60	1171.80	15.83
	1000	22567.20	1305.40	17.29		1000	23094.90	1517.80	15.22
	1200	27152.80	1622.20	16.74		1200	27685.00	1846.40	14.99
Line 1	200	4677.00	213.20	21.94	Line 2	200	4638.30	252.20	18.39
	400	9281.90	487.60	19.04		400	9074.50	564.40	16.08
	600	13860.30	769.00	18.02		600	13590.90	891.60	15.24
	800	18464.00	1090.40	16.93		800	18017.30	1218.20	14.79
	1000	22929.80	1388.20	16.52		1000	22466.60	1567.40	14.33
	1200	27491.00	1711.80	16.06		1200	26891.00	1917.80	14.02
Sin 1	200	4731.10	235.20	20.12	Sin 2	200	4703.10	295.20	15.93
	400	9342.90	516.20	18.10		400	9241.60	606.60	15.24
	600	13891.00	824.40	16.85		600	13765.30	950.60	14.48
	800	18436.60	1114.60	16.54		800	18346.50	1303.00	14.08
	1000	23084.50	1454.00	15.88		1000	22946.20	1674.00	13.71
	1200	27613.20	1772.20	15.58		1200	27440.00	2006.80	13.67
Triangle 1	200	4380.70	121.80	35.97	Triangle 2	200	4694.60	242.60	19.35
	400	8728.70	328.60	26.56		400	9318.20	554.80	16.80
	600	13146.50	577.20	22.78		600	13955.60	854.20	16.34
	800	17561.20	859.40	20.43		800	18530.40	1182.60	15.67
	1000	21984.90	1150.80	19.10		1000	23015.10	1505.80	15.28
	1200	26387.20	1432.40	18.42		1200	27635.90	1844.40	14.98

The results of the experiments show that the average number of test executions and the average number of mutant test cases generated is linear in the number of random walks; see Figure 8.

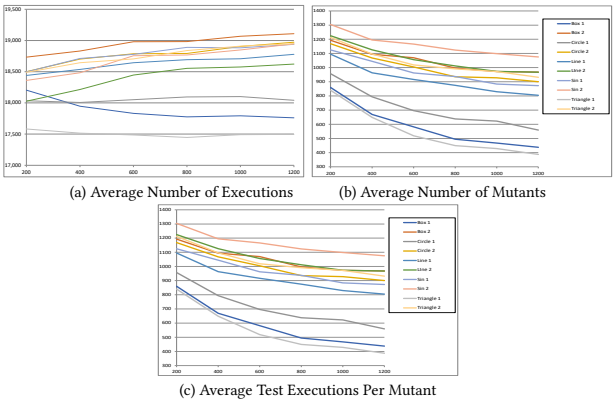
The test effectiveness increases with the number of walks; see Figure 8. Although the overall average test effectiveness varies



(a) Average Number of Executions (b) Average Number of Mutants

(c) Average Executions Per Mutant (d) Overall Test Effectiveness

Figure 8: Results of The Random Walk Strategy



(a) Average Number of Executions (b) Average Number of Mutants

(c) Average Test Executions Per Mutant

Figure 9: Results of the Variable Number of Seeds

between subject programs, the differences on test effectiveness are much smaller than the directed walk strategy. As shown in Figure 8(c) and (d), the test effectiveness for testing subject programs Box 1, Triangle 1 and Circle 1 are poorer than those for the other subjects.

The second set of experiments were with fixed number of walks but variable numbers of seed test cases. Table 3 shows the results of the experiment in which 800 walks were run with variable number of seeds.

Table 3: Experiments Data of Variable Number of Test Cases

Subject	#Seeds	Avg #Runs	Avg #Mutants	Avg #Runs/Mutant	Subject	#Seeds	Avg #Runs	Avg #Mutants	Avg #Runs/Mutant
Box 1	200	18204.30	860.20	21.16	Box2	200	18733.40	1193.20	15.70
	400	17946.90	669.40	26.81		400	18831.90	1094.00	17.21
	600	17833.10	582.00	30.64		600	18979.10	1069.80	17.74
	800	17775.10	494.40	35.95		800	18983.50	997.00	19.04
	1000	17794.20	468.00	38.02		1000	19067.10	972.20	19.61
	1200	17761.10	437.80	40.57		1200	19109.10	968.20	19.74
Circle 1	200	18034.30	957.20	18.84	Circle 2	200	18500.70	1168.80	15.83
	400	18007.00	794.20	22.67		400	18703.00	1067.80	17.52
	600	18051.80	696.20	25.93		600	18786.90	1004.60	18.70
	800	18095.10	638.00	28.36		800	18793.40	935.80	20.08
	1000	18101.30	622.80	29.06		1000	18904.50	928.20	20.37
	1200	18042.40	559.40	32.25		1200	18972.40	900.60	21.07
Line 1	200	18440.20	1095.20	16.84	Line 2	200	18027.10	1225.20	14.71
	400	18536.10	963.00	19.25		400	18214.20	1125.60	16.18
	600	18644.00	916.20	20.35		600	18445.90	1056.60	17.46
	800	18691.70	874.60	21.37		800	18553.70	1012.00	18.33
	1000	18706.30	830.20	22.53		1000	18574.90	973.00	19.09
	1200	18778.40	804.80	23.33		1200	18622.20	967.80	19.24
Sin 1	200	18497.70	1125.00	16.44	Sin 2	200	18360.40	1304.40	14.08
	400	18712.90	1044.60	17.91		400	18483.70	1195.40	15.46
	600	18776.70	962.80	19.50		600	18754.80	1166.20	16.08
	800	18888.80	937.60	20.15		800	18767.80	1124.00	16.70
	1000	18883.30	883.60	21.37		1000	18848.20	1098.20	17.16
	1200	18938.40	872.80	21.70		1200	18946.30	1075.40	17.62
Triangle 1	200	17582.10	840.40	20.92	Triangle 2	200	18489.80	1212.80	15.25
	400	17514.60	648.60	27.00		400	18644.30	1096.00	17.01
	600	17485.20	518.80	33.70		600	18705.20	1018.80	18.36
	800	17446.70	449.80	38.79		800	18840.00	991.60	19.00
	1000	17490.80	428.40	40.83		1000	18910.40	972.80	19.44
	1200	17495.20	387.40	45.16		1200	18948.30	931.80	20.34

For the second set of experiments, as shown in Figure 9, the number of test executions increases as the number of seed test cases increases, while the number of mutant test cases generated decreases. Therefore, the test effectiveness in terms of average number of test executions per mutant generated decreases as the number of seed test cases increases as shown in Figure 9(b).

Figure 9(c) also confirms the observations on test effectiveness made in the first set of experiments. That is, the test effectiveness for subject programs Triangle 1, Box 1 and Circle 1 are obviously poorer than the other subjects. The reason for this phenomenon will be discussed in Subsection 4.3.

4.2.3 Experiments with the random target strategy. The random target strategy only has one parameter: the number of pairs of test cases selected at random. The experiments are conducted with this parameter ranging from 200 to 1200. The experiment data are given in Table 4 below.

Table 4: Experiments Data of The Random Target Strategy

Subject	#Seeds (=#Walks)	Avg #Runs	Avg #Mutants	Avg #Runs/Mutant	Subject	#Seeds (=#Walks)	Avg #Runs	Avg #Mutants	Avg #Runs/Mutant
Box 1	200	728.70	55.60	13.11	Box2	200	2240.50	206.60	10.84
	400	1133.80	93.60	12.11		400	4203.60	400.80	10.49
	600	1755.20	155.60	11.28		600	6355.00	615.60	10.32
	800	2083.90	188.40	11.06		800	8145.90	794.60	10.25
	1000	2790.00	259.00	10.77		1000	10146.00	994.60	10.20
	1200	3518.00	331.80	10.60		1200	12370.00	1217.00	10.16
Circle 1	200	1037.10	86.40	12.00	Circle 2	200	2090.80	191.80	10.90
	400	1724.30	152.80	11.28		400	3903.00	370.60	10.53
	600	2675.50	247.60	10.81		600	5891.50	569.20	10.35
	800	3444.00	324.40	10.62		800	7843.90	764.40	10.26
	1000	4436.00	423.60	10.47		1000	9748.00	954.80	10.21
	1200	5292.00	509.20	10.39		1200	11412.00	1121.20	10.18
Line 1	200	2088.10	191.60	10.90	Line 2	200	2506.50	233.60	10.73
	400	4114.10	391.80	10.50		400	4876.10	468.00	10.42
	600	6235.70	603.60	10.33		600	7039.80	684.00	10.29
	800	8044.00	784.40	10.25		800	9321.90	912.20	10.22
	1000	10182.00	998.20	10.20		1000	12056.00	1185.60	10.17
	1200	11904.00	1170.40	10.17		1200	14116.00	1391.60	10.14
Sin 1	200	2189.90	201.80	10.85	Sin 2	200	2651.30	248.00	10.69
	400	4129.10	393.20	10.50		400	5197.80	500.20	10.39
	600	6243.50	604.40	10.33		600	7727.60	752.80	10.27
	800	8394.00	819.40	10.24		800	10172.00	997.20	10.20
	1000	10186.00	998.60	10.20		1000	12596.00	1239.60	10.16
	1200	12076.00	1187.60	10.17		1200	15192.00	1499.20	10.13
Triangle 1	200	522.70	34.80	15.02	Triangle 2	200	2016.30	184.40	10.93
	400	830.20	63.40	13.09		400	4147.10	395.00	10.50
	600	971.40	77.20	12.58		600	5783.60	558.40	10.36
	800	1403.80	120.40	11.66		800	7573.80	737.40	10.27
	1000	1835.90	163.60	11.22		1000	9791.90	959.20	10.21
	1200	1872.00	167.20	11.20		1200	11170.00	1097.00	10.18

The data show that the average number of test executions and the average number of mutant test cases generated are linear in the number of walks for all subject programs as shown in Figure 10.

The test effectiveness increases with the number of walks since the average number of test executions needed to generate a mutant test cases decreases with the number of walks increases. The test effectiveness of the random target strategy is given in Figure 10(b). The data show that the test effectiveness for subjects Triangle 1, Box 1 and Circle 1 are significantly poorer than those for the other subjects. This is also shown clearly in Figure 10(d).

4.3 Discussion

From the experiments, we observed the following phenomena.

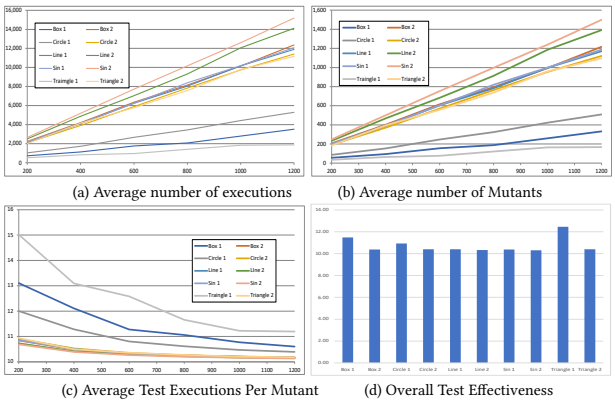


Figure 10: Results of the Random Target Strategy

4.3.1 Factors influencing Test Effectiveness. The test effectivgess of the strategies on various subject programs are summarised in Table 5 and depicted in Figure 11, where the larger the number, the lower the test effectiveness.

Table 5: Summary of Test Effectiveness

Subject	Directed walk	Random walk	Random target
Box 1	323.45	24.63	11.49
Box 2	93.85	16.39	10.38
Circle 1	247.32	20.84	10.93
Circle 2	105.82	16.72	10.41
Line 1	105.82	18.08	10.41
Line 2	55.76	15.48	10.33
Sin 1	122.35	17.18	10.38
Sin 2	64.75	14.52	10.31
Triangle 1	370.38	23.88	12.46
Triangle 2	93.19	16.40	10.41
Avg	**158.27**	**18.41**	**10.75**

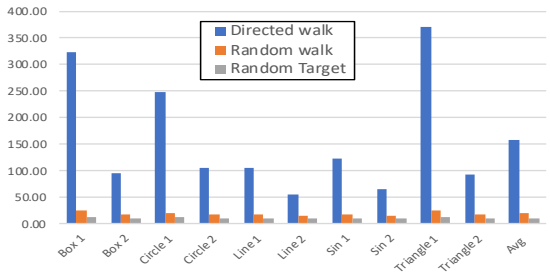


Figure 11: Test Effectiveness on Subject Programs

The data show that for each strategy, the test effectiveness varies significantly according to the different subject programs. However, for each strategy, the experimental data show that the test effectiveness for Box 1 is lower than that for Box 2. The effectiveness for Circle 1 is lower than that for Circle 2, and so on. This phenomenon is not an coincidence.

Theoretically speaking, the test effectiveness for the directed walk strategy is determined by the probability that there is a border between two subdomains in the right direction from a test case and within the walking distance. For the random target strategy, the test effectiveness is determined by the probability that two random test cases fall in two different subdomains. For the random walk strategy, the test effectiveness is determined by the probability that there is a border nearby to a randomly selected test case. These properties have a number of implications.

First, given a classification application, one should select the most effective strategy to explore the Pareto fronts betweem subdomains based on the understanding of the application. The data obtained from our experiments are not sufficient to compare the strategies on their effectiveness. This is because the probability of finding a pair in the Pareto front heavily depends on the size and location of the subdomains of the classification application. There is no benchmark on such parameters in real applications as far as we know. Our subjects in the experiments may not be representative of the distribution of the parameters in real applications.

Second, it provides a good explanation of the observations made in the previous sections that the number of pairs generated for the Pareto front is a linear function of the number of seed test cases or number of walks since they are independent.

Moreover, although the test effectiveness is mostly determined by the size, shape and location of the subdomains that the program classifies, for directly walk and random walk strategies, it is also affected by the number of steps walked and the number of iterations in the refinement. The number of steps walked influences the probability of finding two points in different subdomains and also the total number of test executions. The longer the walk, the more likely one is to find two points in different subdomains, but this requires more test executions. Thus, a balance between these two contradictory factors of test effectiveness must be made to achieve the best test effectiveness.

Finally, the number of iterations in the refinement loop controls the distance betwen the pair of test cases in the Pareto fronts generated. It has no impact on the probablity of finding two data points in different subdomains, but does have an affect on test effectiveness. The shorter distance mutations requires more iterations, thus more test executions, and therefore, is less effective. For random walk and directed walk strategies, the number of iterations can be selected for correctness theorems proved in this paper. For the random target strategy, usually more iterations are required than the other two strategies.

4.3.2 Validity of the Experiments. As pointed out at the beginning of the section, the experiments are designed to determine which factors have an effect on the test effectiveness of the strategies. The subject programs used in the controlled experiments have subdomains that are of typical shapes in data mining and machine learning applications [1, 5, 8]. As discussed above, the conclusion that we draw from the experiments are not depending on specific features of subdomains such as the size and location. However, as discussed above, they do provide insight on the factors that affect test effectiveness. Therefore, we are confident that the conclusions drawn from the experiments are valid.

5 CONCLUSION

5.1 Related Work

Exploratory testing was originally proposed for improving GUI-based manual testing of web-based applications, which also often lacks a clear definition of software correctness [12]. The name is given to a common practice in industry that existed for many years without guidance until recently. The notion of exploratory strategy was first defined by Whittaker [12] as guidance on how to manually explore the software in the most effective way.

Research on testing AI applications has been active in recent years [2, 4, 6]. It is interesting to observe that datamorphisms are actually used to testing AI applications like driverless vehicles [11, 15]. Our case study with face recognition shows that test automation can be improved by reusing datamorphisms if they are explicitly defined and supported by a testing tool [17, 18, 21]. However, the testing of classification applications has not been studied intensively. An interesting work by Xie *et. al.* is the development of a set of metamorphic relations as test oracles for the clustering and classification applications [13]. In [14], a case study is reported to use these metamorphic relations to test a clustering function generated by the data mining tool Weka, in which datamorphisms are also used although they are not explicitly defined.

5.2 Main Contributions

The main contribution of this paper is the adaptation of the notion of exploratory strategy to the testing of AI applications. We demonstrated that such strategies can be formally defined in the datamorphic testing framework. They have also been implemented in the automated datamorphic testing tool Morphy [17].

In this paper, we studied the theoretical properties of three exploratory strategies for the discovery of the Pareto front of classification applications. We formally proved the correctness of the algorithms that implement the strategies.

We have also conducted controlled experiments with the exploration strategies. Experimental data demonstrated the factors that have impact on test effectiveness of these strategies. The observations obtained from experiments provide a guidance to the selection of the strategies for a given classification application and the choices of parameters to apply the strategies.

5.3 Future Work

The data spaces of the running example and the subjects of the experiments have fixed dimensions on continuous values. This is for the purpose of easily visualising the results. The strategies are independent to the continuity and dimensions of the data space, thus they are also applicable to other types of classification applications. The proofs of their correctness are also independent to these features, thus the correctness theorems also hold for such data spaces. It is interesting to conduct experiments using different types of data spaces, such as image, audio, video and text values. We are conducting case studies with real machine learning applications to evaluate the practical usability of the strategies.

There are also many possible variations of the strategies proposed and studied in this paper. In particular, the algorithms in this paper do not need a test morphisms that measure the distances between two test cases. If such a test morphism is available, the termination of the refinement loop can be determined by the distances between the pair of test cases.

The analysis of the phenomena observed in the experiments suggested that the test effectiveness depended on the probability of finding two test cases that are in different classification subdomains. A formal proof of this property will give a solid foundation for understanding these strategies and providing precise guidance to the selection of the parameters of the strategies. Thus, it is worth further research.

REFERENCES

[1] C. Aggarwal. 2015. *Data Mining: The Textbook.* Springer.

[2] X. Bai, J. Li, and A. Ulrich (Eds.). 2018. *Proc. of IEEE/ACM 13th International Workshop on Automation of Software Test (AST 2018),.* IEEE Computer Society, Gothenburg, Sweden.

[3] T. Y. Chen, F-C. Kuo, H. Liu, P-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou. 2018. Metamorphic testing: A review of challenges and opportunities. *ACM Comput. Surv.* 51, 1, Article 4 (Jan 2018), 27 pages. https://doi.org/10.1145/3143561

[4] A. Gotlieb, M. Roper, and P Zhang (Eds.). 2019. *Proc. of The First IEEE International Conference on Artificial Intelligence Testing (AITest 2019).* IEEE Computer Society, Los Alamitos, CA, USA. https://doi.org/10.1109/AITest.2019

[5] M. Mohri, A. Rostamizadeh, and A. Talwalkar. 2012. *Foundations of Machine Learning.* The MIT Press.

[6] M. Roper and Z. Q. Zhou (Eds.). 2020. *Proc. of The Second IEEE International Conference on Artificial Intelligence Testing (AITest 2020).* IEEE Computer Society, Los Alamitos, CA, USA. (In Press) pages.

[7] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen. 2018. Metamorphic testing: testing the untestable. *IEEE Software* (2018), 1–1. https://doi.org/10.1109/MS.2018.2875968

[8] S. Shalev-Shwartz and S. Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press.

[9] L. Shan and H. Zhu. 2009. Generating structurally complex test cases by data mutation: A case study of testing an automated modelling tool. *Comput. J.* 52, 5 (Aug 2009), 571–588.

[10] M. Sutton, A. Greene, and P. Amini. 2007. *Fuzzing: Brute Force Vulnerability Discovery.* Addison-Wesley.

[11] Y. Tian, K. Pei, S. Jana, and B Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proc. of The 40th IEEE/ACM Int'l Conf. on Software Engineering (ICSE 2018).* IEEE Computer Society, Gothenburg, Sweden, 303–314.

[12] J. A. Whittaker. 2009. *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design.* Pearson Education. https://books.google.co.uk/books?id=BsB0NpkcdgIC

[13] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen. 2011. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software* 84 (2011), 544–558.

[14] S. Yang, D. Towey, and Z. Zhou. 2019. Metamorphic exploration of an unsupervised clustering program. In *Proc. of IEEE/ACM 4th International Workshop on Metamorphic Testing (MET 2019).* IEEE Computer Society, 48–54.

[15] Z. Q. Zhou and L. Sun. 2019. Metamorphic testing of driverless cars. *Commun. ACM* 62, 3 (March 2019), 61–67.

[16] H. Zhu. 2015. JFuzz: A tool for automated Java unit testing based on data mutation and metamorphic testing methods. In *Proc. of The 2nd Int'l Conf. on Trustworthy Systems and Their Applications (TSA 2015).* 8–15.

[17] H. Zhu, I. Bayley, D. Liu, and X. Zheng. 2019. *Morphy: A Datamorphic Software Test Automation Tool.* Technical Report OBU-ECM-AFM-2019-01. School of Engineering, Computing and Mathematics, Oxford Brookes University, Oxford, UK. http://arxiv.org/abs/1912.09881

[18] H. Zhu, I. Bayley, D. Liu, and X. Zheng. 2020. Automation of Datamorphic Testing. In *Proc. of 2nd IEEE International Conference on Artificial Intelligence Testing (AITest 2020).* In Press.

[19] H. Zhu, P. Hall, and J. May. 1997. Software unit test coverage and adequacy. *ACM Computing Survey* 29, 4 (Dec. 1997), 366–427.

[20] H. Zhu, D. Liu, I. Bayley, R. Harrison, and F. Cuzzolin. 2018. *Datamorphic Testing: A Methodology for Testing AI Applications.* Technical Report OBU-ECM-AFM-2018-02. School of Engineering, Computing and Mathematics, Oxford Brookes University, Oxford OX33 1HX, UK. http://arxiv.org/abs/1912.04900

[21] H. Zhu, D. Liu, I. Bayley, R. Harrison, and F. Cuzzolin. 2019. Datamorphic Testing: A Method for Testing Intelligent Applications. In *Proc. of The First IEEE International Conference on Artificial Intelligence Testing (AITest 2019).* IEEE Computer Society, Los Alamitos, CA, USA, 149–156. https://doi.org/10.1109/AITest.2019.00018