Tutorial at IEEE AITest 2023

Athens, Greece, 17th July 2023

The Datamorphic Testing Methodology

-- Principles, Tools and Applications to ML

Prof. Hong Zhu
School of Engineering, Computing and Mathematics
Oxford Brookes University
Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk

Acknowledgement and References

- Hong Zhu, et al., [2018], ***Datamorphic Testing: A Methodology for Testing AI Applications***, Technical Report OBU-ECM-AFM-2018-02, Oxford Brookes University, Oxford OX33 1HX, UK, Dec. 21, 2018. (Arxiv at http://arxiv.org/abs/1912.04900)
- Hong Zhu, et al. [2019], ***Morphy: A Datamorphic Software Test Automation Tool***, Technical Report OBU-ECM-AFM-2019-01, Oxford Brookes University, Oxford, UK. 9 Dec. 2019. (Arxiv at http://arxiv.org/abs/1912.09881)
- Hong Zhu, et al. [2019], ***Datamorphic Testing: A Method for Testing Intelligent Applications***, in Proc. of IEEE AITest 2019, San Francisco, California, USA, April, 4 - 9, 2019.
- Hong Zhu and Ian Bayley [2020], ***Exploratory Datamorphic Testing of Classification Applications***, The 1st IEEE/ACM International Conference on Automation of Software Test (AST 2020), Seoul, South Korea, May 25-26, 2020, pp51-60.
- Hong Zhu, Ian Bayley, Dongmei Liu and Xiaoyu Zheng [2020], ***Automation of Datamorphic Testing***, The Second IEEE International Conference on Artificial Intelligence Testing (AITest 2020), Oxford, UK, April 13 - 16, 2020, pp64-72.
- Hong Zhu, Ian Bayley, and Mark Green [2022]. ***Metrics for Measuring Error Extents of Machine Learning Classifiers***. In Proc. of the 2022 IEEE International Conference On Artificial Intelligence Testing (AITest 2022), 15-18 Aug. 2022, pp. 48-55.
- Hong Zhu, and Ian Bayley [2022], ***Discovering boundary values of feature-based machine learning classifiers through exploratory datamorphic testing***, Journal of Systems and Software, Vol. 187, Article 111231, May 2022.
- Hong Zhu, et al. [2023], ***A Scenario-Based Functional Testing Approach to Improving DNN Performance*** (*Invited Paper*), in Proc. of The 17th IEEE International Conference on Service-Oriented System Engineering (SOSE 2023), 17th-20th July, 2023, Athens, Greece. (*In press*)

Outline

1. **Background**
2. **Principles and Basic Concepts**
3. **Morphy: An Automated Datamorphic Testing Environment**
4. **Examples of Application to Machine Learning**

Background

- Machine learning (ML) is increasingly used in computer applications
  - Personalisation (e.g. targeted advertisement)
  - Security (e.g. authentication through face recognition, fingerprint, etc.)
  - Driverless vehicles
  - Big Data, IoT, Edge and Fog computing, Cloud computing (e.g. IT Operations)
  - Smart cities, smart homes, healthcare, etc.
  - Robotics (e.g. chatbots, rescue devices, etc.)
- Inadequately tested AI applications impose a threat to the safety, security and reliability of computer systems
  - Fatal accidents of driverless cars
  - Unfairness in recruitment and job applications
  - Etc.
- Testing ML applications are expensive and difficult
  - Large volume of test dataset is required
  - Difficult and expensive to label data for testing
  - Traditional testing techniques, methods and tools are not simply applicable

The Challenges

- **Fundamental differences between traditional programs and ML models**
  - A machine learning model cannot be ***debugged.***
    - We cannot change a ML model at microscale manually to fix “bugs”.
    - To improve a ML model, it has to be re-trained!
  - A machine learning model cannot be ***verified*** *or* ***validated*** for its ***correctness*.**
    - ***Impossible***: In lack of complete verifiable and testable specification of requirements
    - ***Undesirable***: Verification or validation of a ML model’s correctness is ***undesirable***, if not impossible.
    - ***Need to be statistically Assessed***: The quality of a ML model must be ***assessed statistically***, because the PAC ML is regarded as the theoretical foundation for ML applications
- **Implications on software testing in practice**
  - How to provide ***feedbacks*** to developers to improve the quality of the ML model
    - A list of incorrect instances alone (traditional bug reports) may not be useful.
  - How to ***assess*** the quality of ML models
    - Static testing, such as formal review and Fagan inspection, may not be applicable
    - Quality attributes specific for ML applications: robustness, fairness, etc.
  - How to ***manage*** testing process and resources
    - Large volume of data and frequent change of the ML model require test automation

Part 1

Principles and Basic Concepts

The Philosophy of Datamorphic Testing

Datamorphic testing takes a ***systems engineering*** approach to software testing.

- It regards software testing as an ***engineering process***.
- It emphasises on the ***system*** that embodies testing activities and assets.

Software testing is an engineering process in which a ***test system*** is developed, maintained, evolved and operated to achieve the purposes of testing and to manage testing resources effectively and efficiently.

What is a system?

- consisting of components that interact with each other
- demonstrating functions, properties and behaviours that beyond what each individual component alone can

Test System

- What is a test system
  - A test system is a system for supporting testing activities and manage testing resources
- Why do we explicitly define a test system
  - Specified formally or informally
  - Operated to achieve testing purposes
  - Tested and formally proved for correctness
  - Maintained, reused, and evolved like all software assets
  - Implemented as software assets
  - Used to achieve test automation
- How should a test system be defined and structured
  - A test system should not be just an aggregate of unrelated assets.
  - A test system should be well structured to enable test automation, especially
    - Effective and efficient performance of testing activities, manage test recourse
    - Easy to evolve when the system under test evolves
    - Reusable for different testing purposes and different systems to be tested

Artefacts of Software Testing

- **Entities**:
  - Objects and data used and/or generated in testing
  - *Examples*:
    - *test cases*, *test suites,* the *program under test*, *test design documents, test reports*, etc.
- **Morphisms**:
  - Mappings from and/or to test entities
  - Generating and transforming test entities to achieve testing objectives
  - Invoked to perform test activities
  - Composed to form test processes
  - Implemented as test code or test scripts for test automation
  - *Examples*:
    - test case generators, test oracles, test adequacy metrics, test result analysers, bug report generators, etc.

Test Systems in Datamorphic Testing Methodology

Definition:

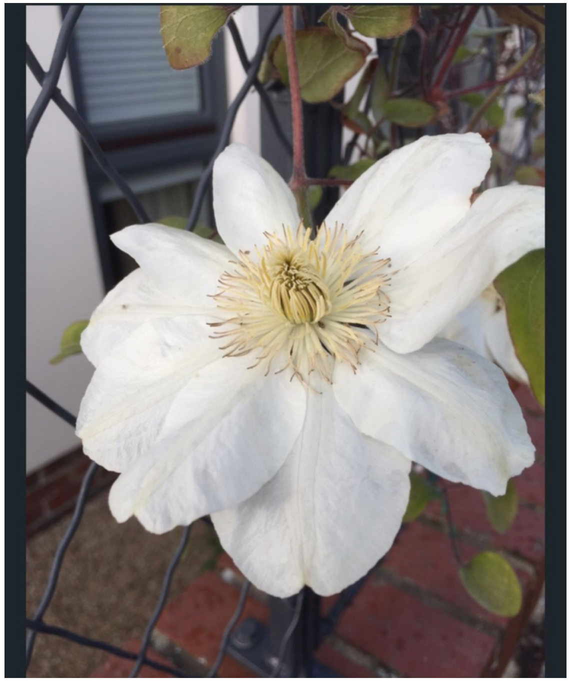
A test system ***T*** **= <*E*, *M*>** consists of a set ***E*** of **test entities** and a set ***M*** of **test morphisms**, where each test morphism in ***M*** is a mapping defined on the test entities in ***E***.

Examples of Test Morphisms

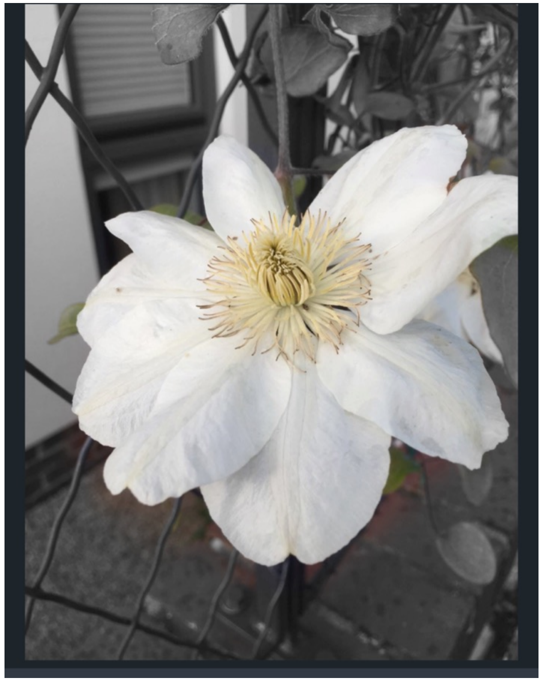
- Seed makers:
  - Generates a set of test data from other types of test entities. Such test cases are called ***seed test cases***.
  - Examples:
    - Generate from the program under test
    - Selected from an existing profile of recorded real data
    - Convert csv files into image, etc.
- Datamorphisms:
  - Transforms existing test data to new test data. Such test cases are called ***mutants*** of the original test data
  - Also called *test data mutation operators* in [Shan & Zhu 2006, Zhu 2015]
- Metamorphisms:
  - Predicates on test cases to check if the program is correct or not on the test case
    - Checking the relationship between the original and mutant test data, and their expected outputs from the program
  - Metamorphic relations (*compare*):
    - A k-ary relation (k>1) on test cases
    - A special form of axioms in algebraic formal specifications

Example: Identification of Flowers

- ***Datamorphism***: Change background colour to black-and-white

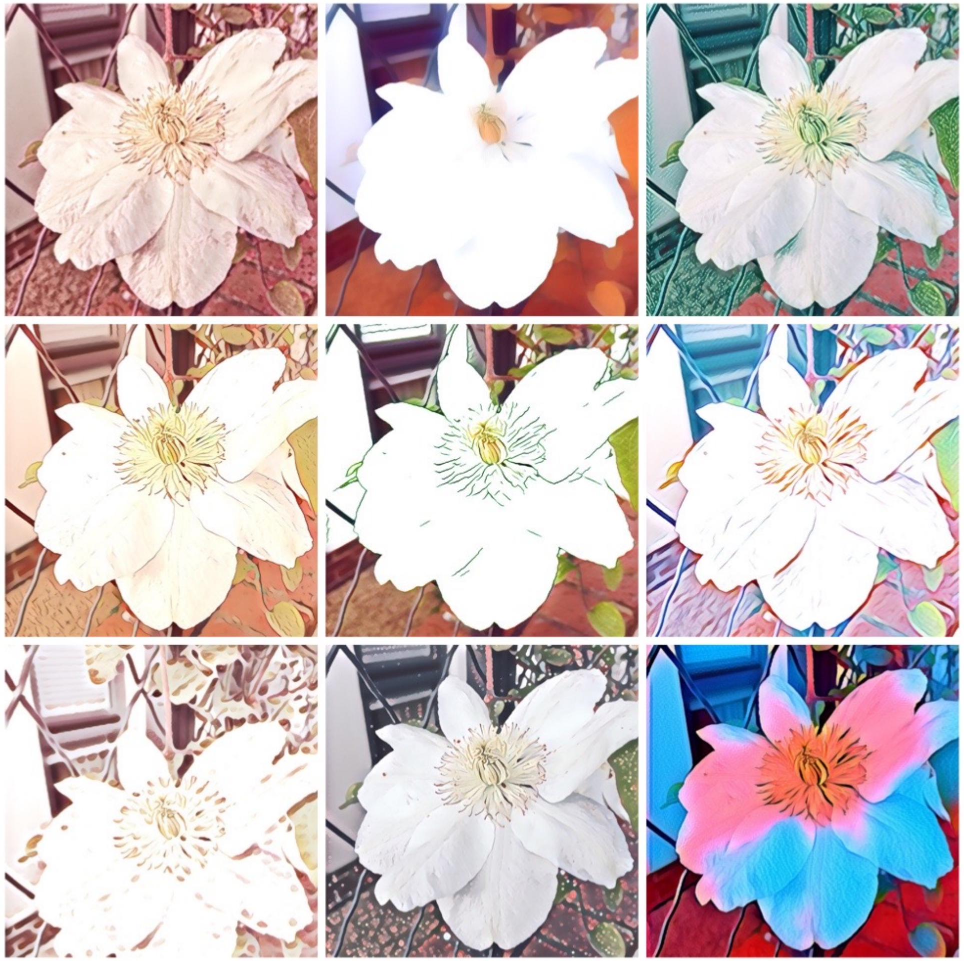


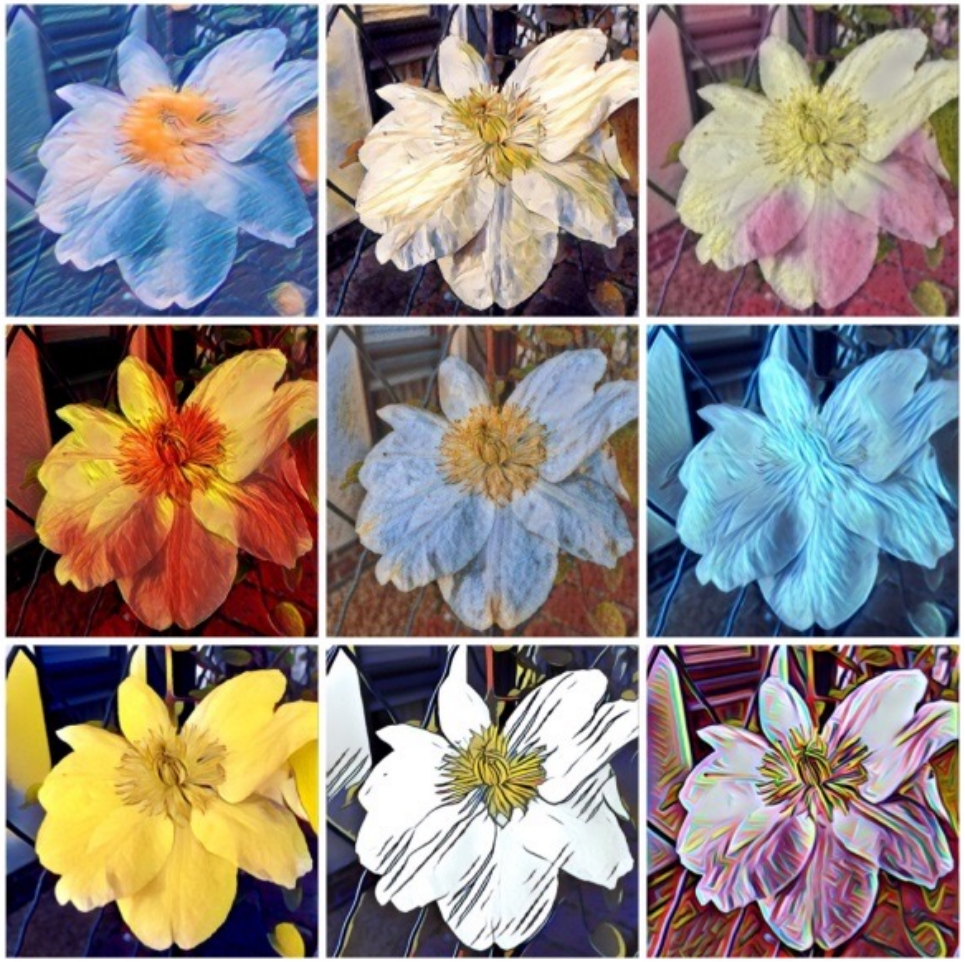
Seed test case: original test case



Mutant test case: test case generated by applying the *datamorphism*

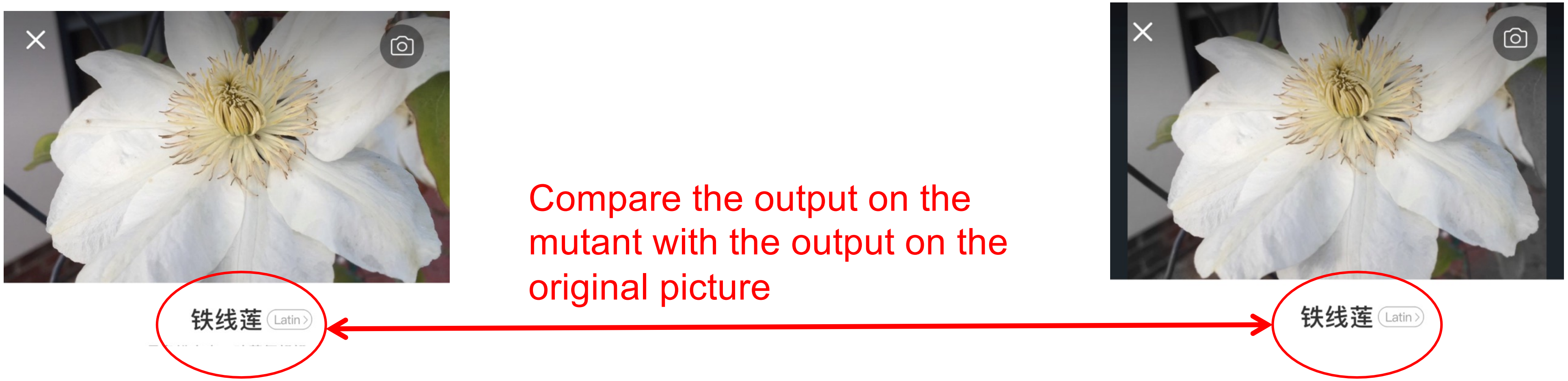
Example: More Datamorphisms





Example: A Metamorphism

By changing the background into black-and-white, the flower should be identified as the same kind as in the original picture.

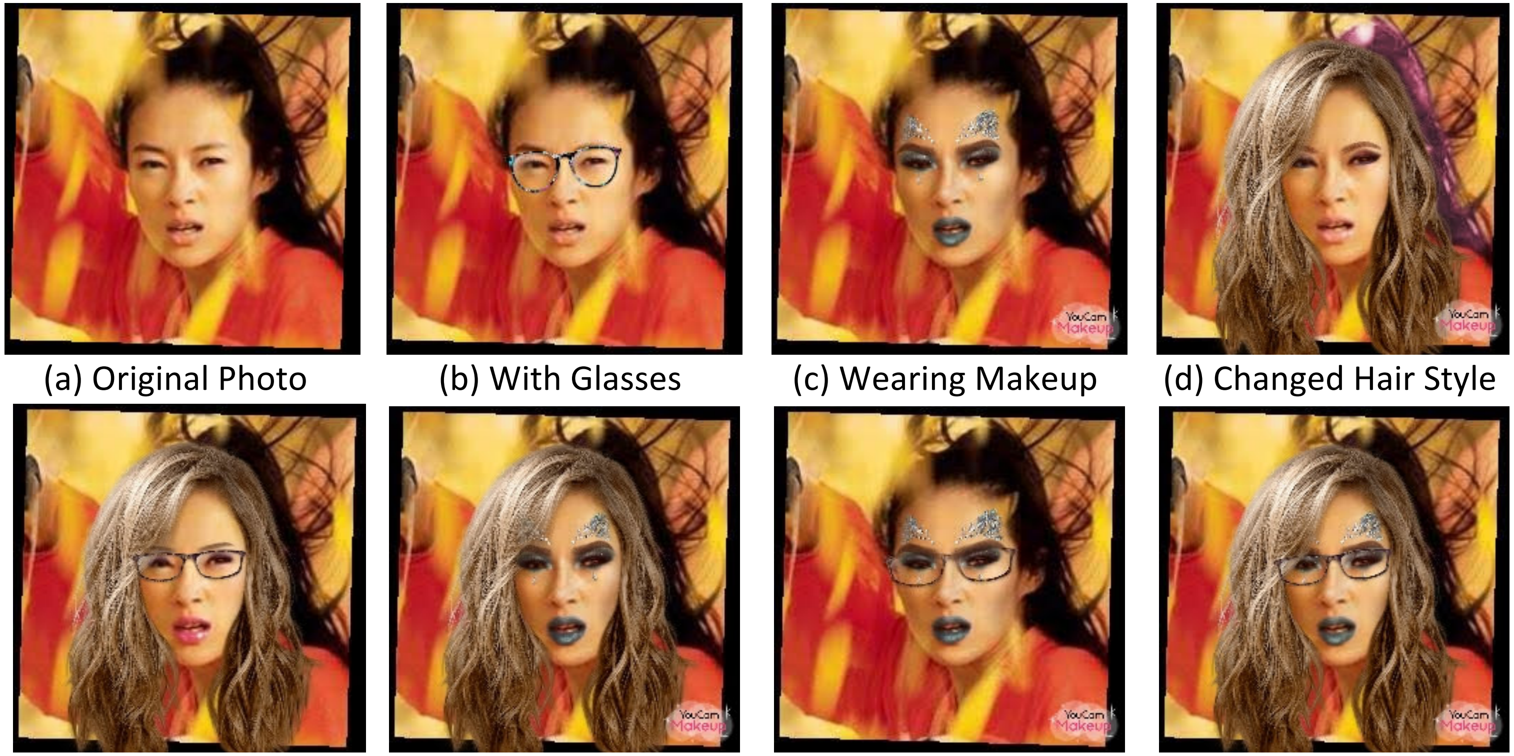


Compare the output on the mutant with the output on the original picture

Metamorphic relation:

$$FlowerRec(x) = FlowerRec(ChangeBackground(x))$$

Example: Datamorphisms



(a) Original Photo

(b) With Glasses

(c) Wearing Makeup

(d) Changed Hair Style

(e) b + d

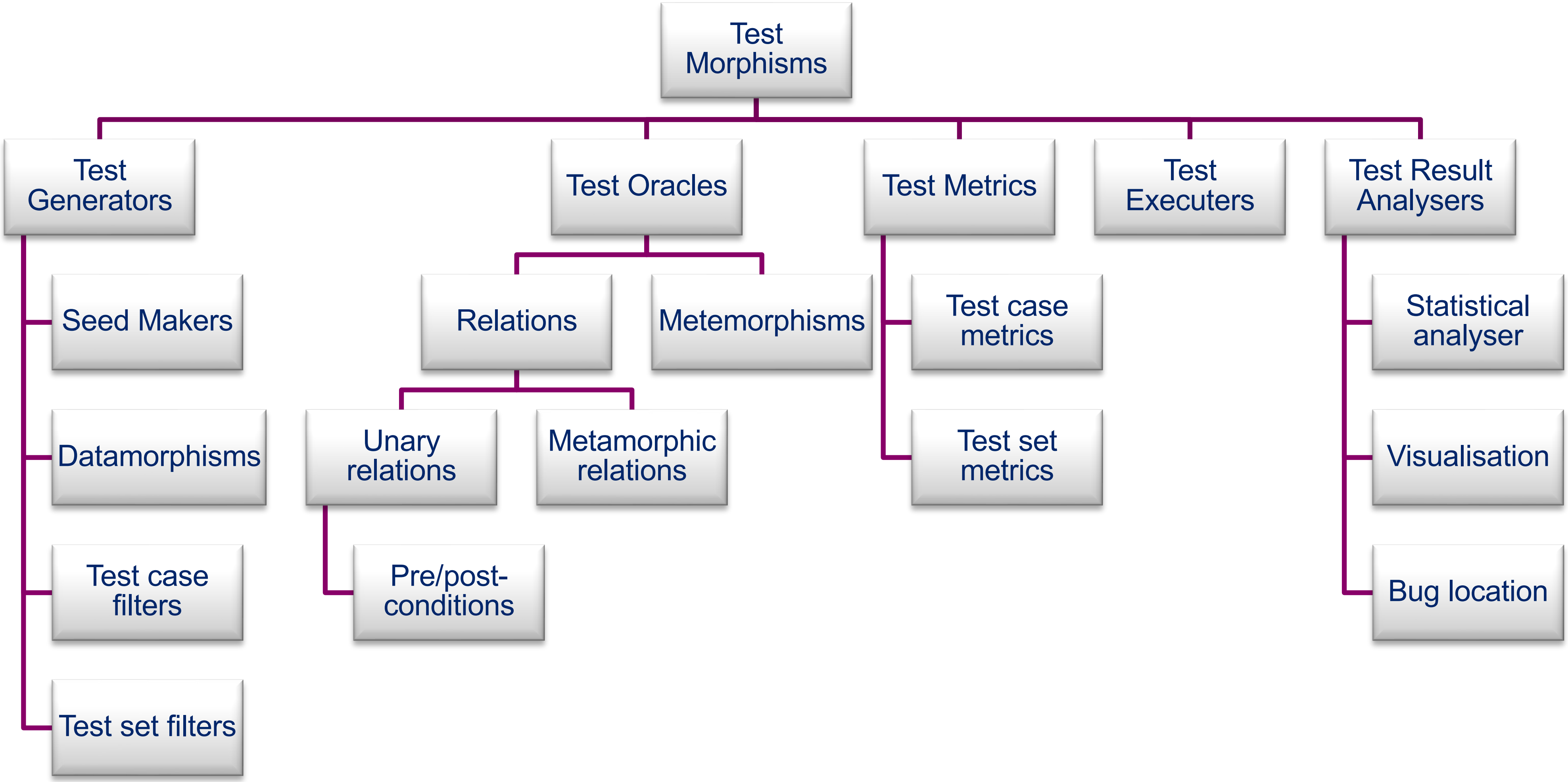
(f) c + d

(g) b + c

(h) b + c + d

Higher order mutants can be obtained by applying datamorphisms multiple times. They represent combinations of operation conditions/scenarios.

A Category of Test Morphisms



Part 2

Morphy: An Automated Tool for Datamorphic Testing

Main Functions

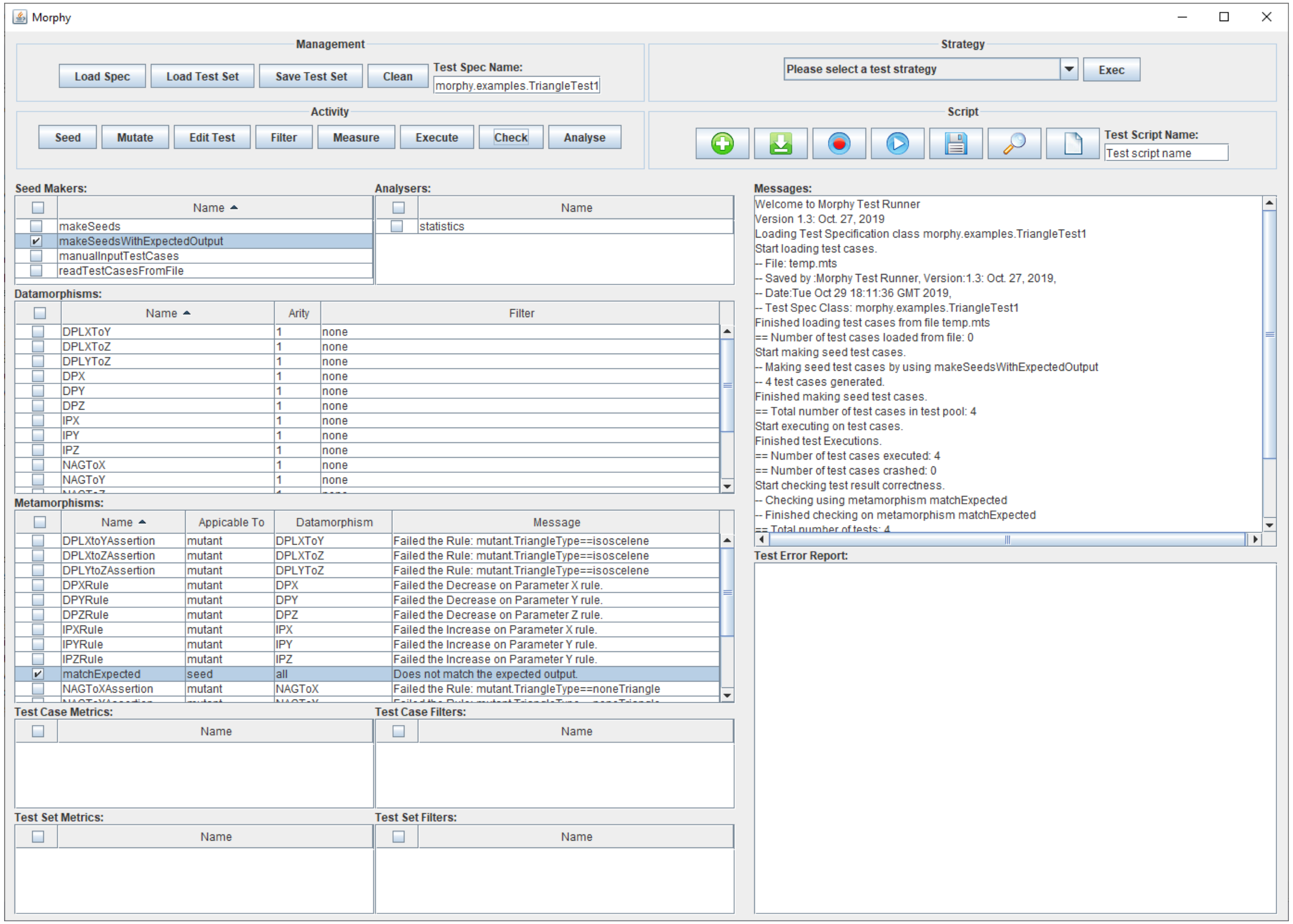
- Management of test systems
  - Management of test entities
  - Management of test specifications

A test system is defined/implemented as a Java class, where

- *Test entities* are stored in attributes
- *Test morphisms* are implemented as methods

- Test Automation at 3 levels:
  - Activity level:
    - Perform testing activities automatically through invocations of test morphisms
  - Strategy level:
    - Apply test strategies implemented by the tools with user’s selection of parameters
  - Process level:
    - Interactive uses of the tool can be recorded, and replayed
    - Test scripts can be edited and executed

Graphic User Interface



Test Spec Name: morphy.examples.TriangleTest1

Strategy: Please select a test strategy

Test Script Name: Test script name

Seed Makers:

□	Name ▲
□	makeSeeds
☑	makeSeedsWithExpectedOutput
□	manualInputTestCases
□	readTestCasesFromFile

Analysers:

□	Name
□	statistics

Datamorphisms:

□	Name ▲	Arity	Filter
□	DPLXToY	1	none
□	DPLXToZ	1	none
□	DPLYToZ	1	none
□	DPX	1	none
□	DPY	1	none
□	DPZ	1	none
□	IPX	1	none
□	IPY	1	none
□	IPZ	1	none
□	NAGToX	1	none
□	NAGToY	1	none

Metamorphisms:

□	Name ▲	Appicable To	Datamorphism	Message
□	DPLXtoYAssertion	mutant	DPLXToY	Failed the Rule: mutant.TriangleType==isoscelene
□	DPLXtoZAssertion	mutant	DPLXToZ	Failed the Rule: mutant.TriangleType==isoscelene
□	DPLYtoZAssertion	mutant	DPLYToZ	Failed the Rule: mutant.TriangleType==isoscelene
□	DPXRule	mutant	DPX	Failed the Decrease on Parameter X rule.
□	DPYRule	mutant	DPY	Failed the Decrease on Parameter Y rule.
□	DPZRule	mutant	DPZ	Failed the Decrease on Parameter Z rule.
□	IPXRule	mutant	IPX	Failed the Increase on Parameter X rule.
□	IPYRule	mutant	IPY	Failed the Increase on Parameter Y rule.
□	IPZRule	mutant	IPZ	Failed the Increase on Parameter Y rule.
☑	matchExpected	seed	all	Does not match the expected output.
□	NAGToXAssertion	mutant	NAGToX	Failed the Rule: mutant.TriangleType==noneTriangle

Test Case Metrics:

□	Name

Test Case Filters:

□	Name

Test Set Metrics:

□	Name

Test Set Filters:

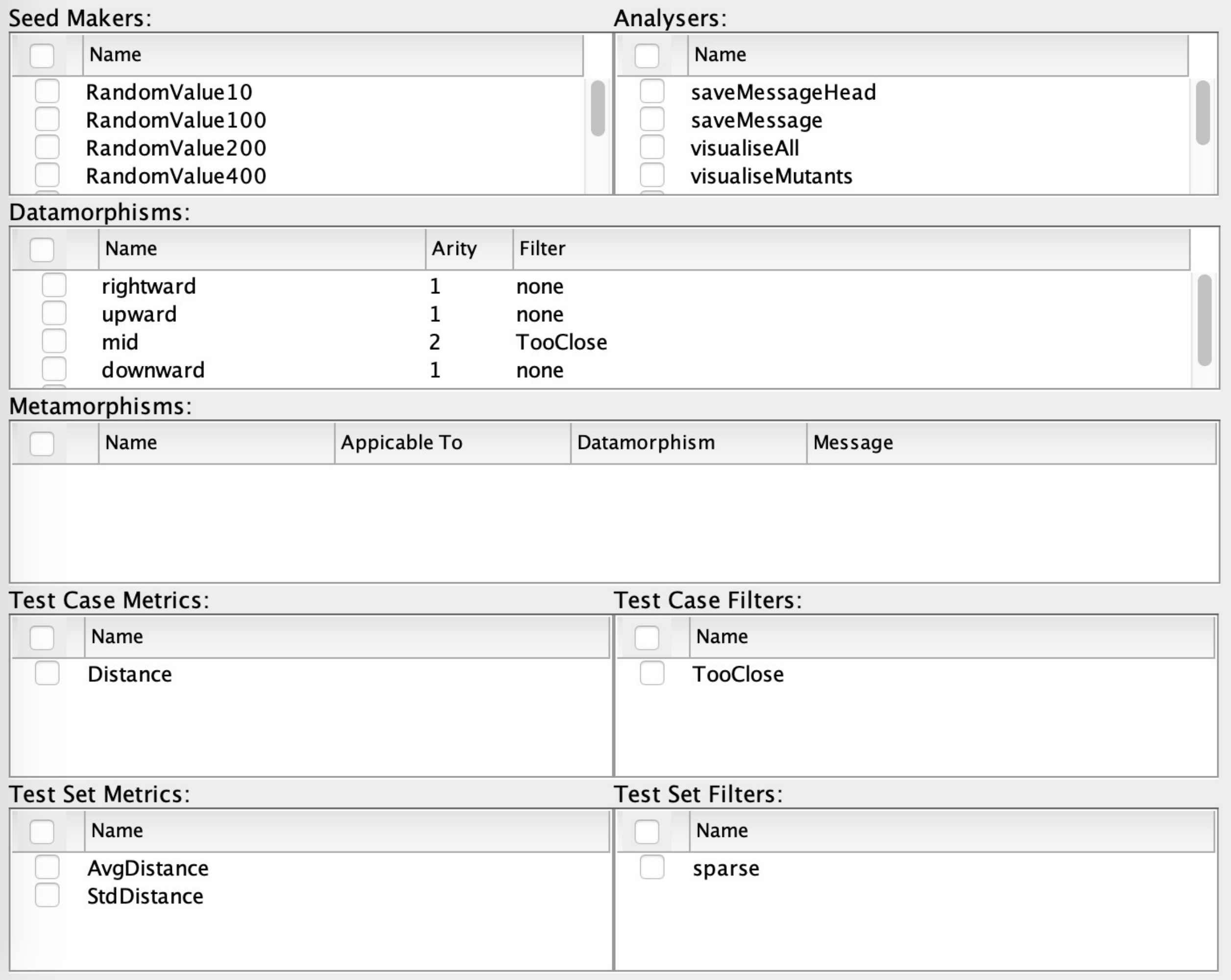
□	Name

Messages:

```
Welcome to Morphy Test Runner
Version 1.3: Oct. 27, 2019
Loading Test Specification class morphy.examples.TriangleTest1
Start loading test cases.
-- File: temp.mts
-- Saved by :Morphy Test Runner, Version:1.3: Oct. 27, 2019,
-- Date:Tue Oct 29 18:11:36 GMT 2019,
-- Test Spec Class: morphy.examples.TriangleTest1
Finished loading test cases from file temp.mts
== Number of test cases loaded from file: 0
Start making seed test cases.
-- Making seed test cases by using makeSeedsWithExpectedOutput
-- 4 test cases generated.
Finished making seed test cases.
== Total number of test cases in test pool: 4
Start executing on test cases.
Finished test Executions.
== Number of test cases executed: 4
== Number of test cases crashed: 0
Start checking test result correctness.
-- Checking using metamorphism matchExpected
-- Finished checking on metamorphism matchExpected
== Total number of tests: 4
```

Test Error Report:

Test Morphism Panel



Seed Makers:

□	Name
□	RandomValue10
□	RandomValue100
□	RandomValue200
□	RandomValue400

Analysers:

□	Name
□	saveMessageHead
□	saveMessage
□	visualiseAll
□	visualiseMutants

Datamorphisms:

□	Name	Arity	Filter
□	rightward	1	none
□	upward	1	none
□	mid	2	TooClose
□	downward	1	none

Metamorphisms:

□	Name	Appicable To	Datamorphism	Message

Test Case Metrics:

□	Name
□	Distance

Test Case Filters:

□	Name
□	TooClose

Test Set Metrics:

□	Name
□	AvgDistance
□	StdDistance

Test Set Filters:

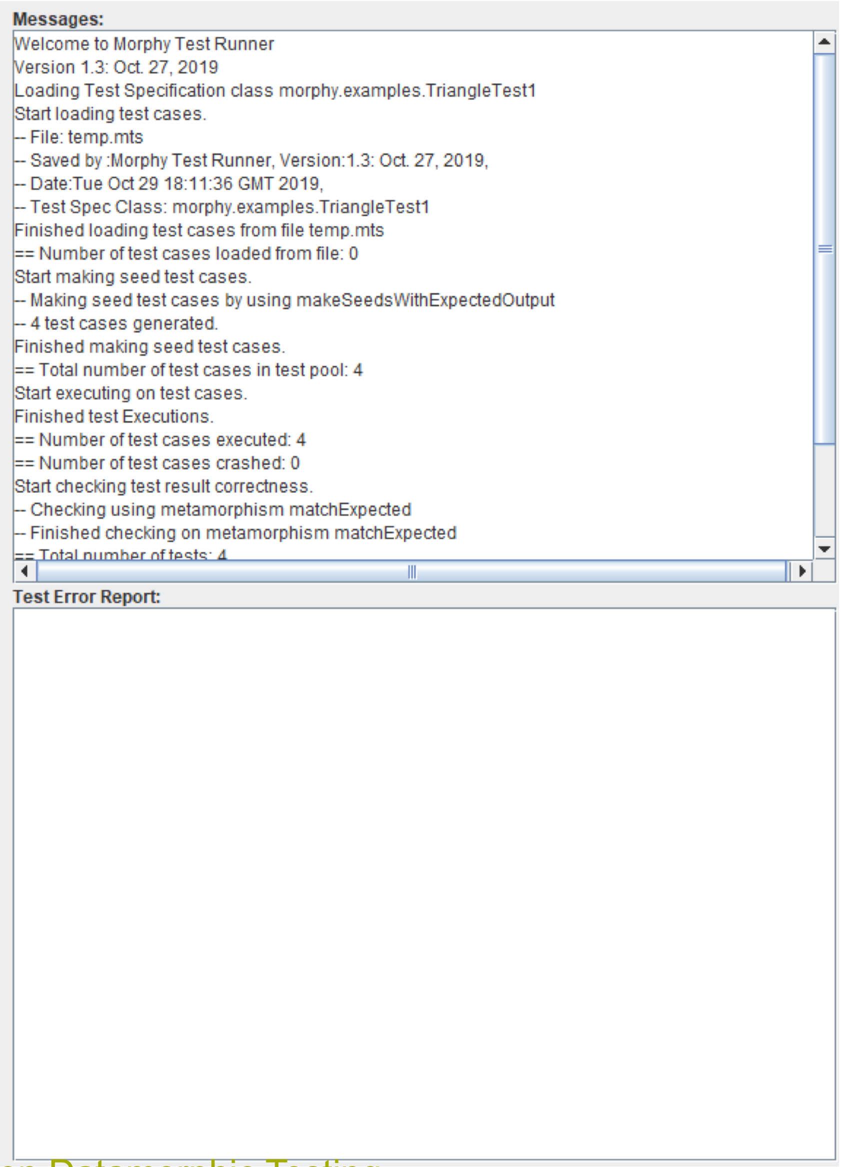
□	Name
□	sparse

Various types of test morphisms are listed in this panel

Users can select the test morphisms to apply interactively or as parameters of strategies

Message Panel

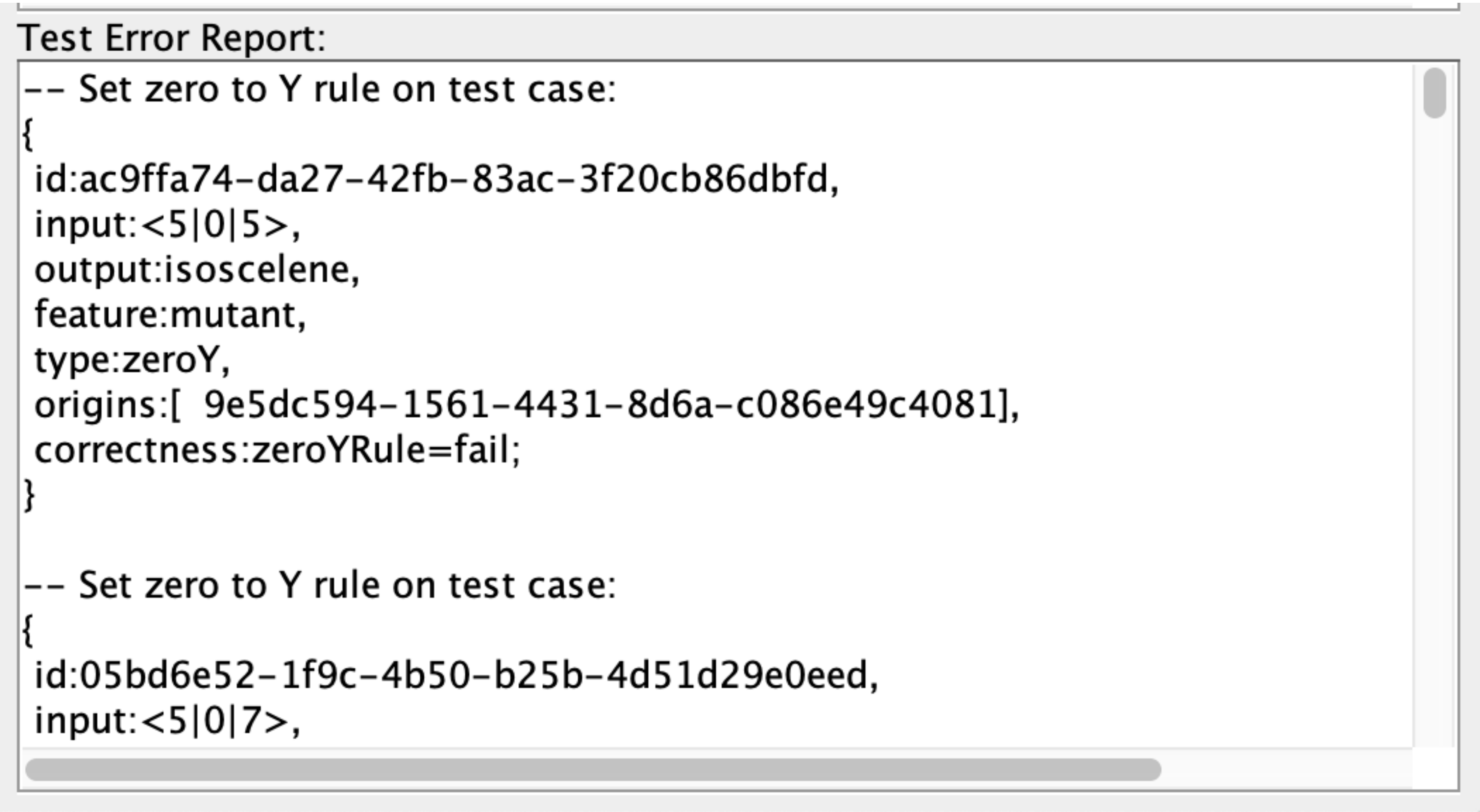
The Message panel shows the activities performed during the test process



```
Welcome to Morphy Test Runner
Version 1.3: Oct. 27, 2019
Loading Test Specification class morphy.examples.TriangleTest1
Start loading test cases.
-- File: temp.mts
-- Saved by :Morphy Test Runner, Version:1.3: Oct. 27, 2019,
-- Date:Tue Oct 29 18:11:36 GMT 2019,
-- Test Spec Class: morphy.examples.TriangleTest1
Finished loading test cases from file temp.mts
== Number of test cases loaded from file: 0
Start making seed test cases.
-- Making seed test cases by using makeSeedsWithExpectedOutput
-- 4 test cases generated.
Finished making seed test cases.
== Total number of test cases in test pool: 4
Start executing on test cases.
Finished test Executions.
== Number of test cases executed: 4
== Number of test cases crashed: 0
Start checking test result correctness.
-- Checking using metamorphism matchExpected
-- Finished checking on metamorphism matchExpected
== Total number of tests: 4
```

Error Report Panel

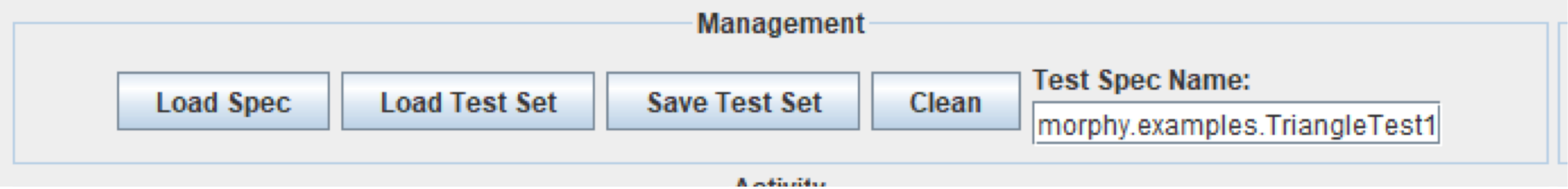
The Error Report panel shows errors detected during the test process



Test Error Report:

```
-- Set zero to Y rule on test case:
{
 id:ac9ffa74-da27-42fb-83ac-3f20cb86dbfd,
 input:<5|0|5>,
 output:isoscelene,
 feature:mutant,
 type:zeroY,
 origins:[  9e5dc594-1561-4431-8d6a-c086e49c4081],
 correctness:zeroYRule=fail;
}

-- Set zero to Y rule on test case:
{
 id:05bd6e52-1f9c-4b50-b25b-4d51d29e0eed,
 input:<5|0|7>,
```

Management of Test System

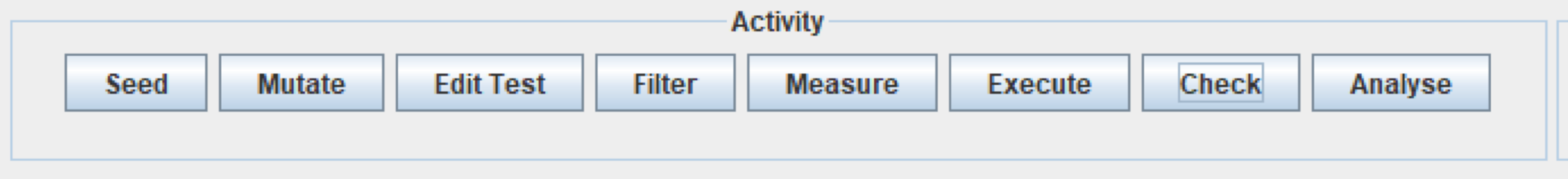


Test Spec Name: morphy.examples.TriangleTest1

- **Load Spec**: Load a test specification that is a Java class file.
- **Load Test Set**: Load a test set file previously save on the computer and add them to the current set of test cases
- **Save Test Set**: Save the current test set to a file
- **Clean**: Re-initialise the system’s state
- **Test Spec Name**: Give the test specification name that is currently used

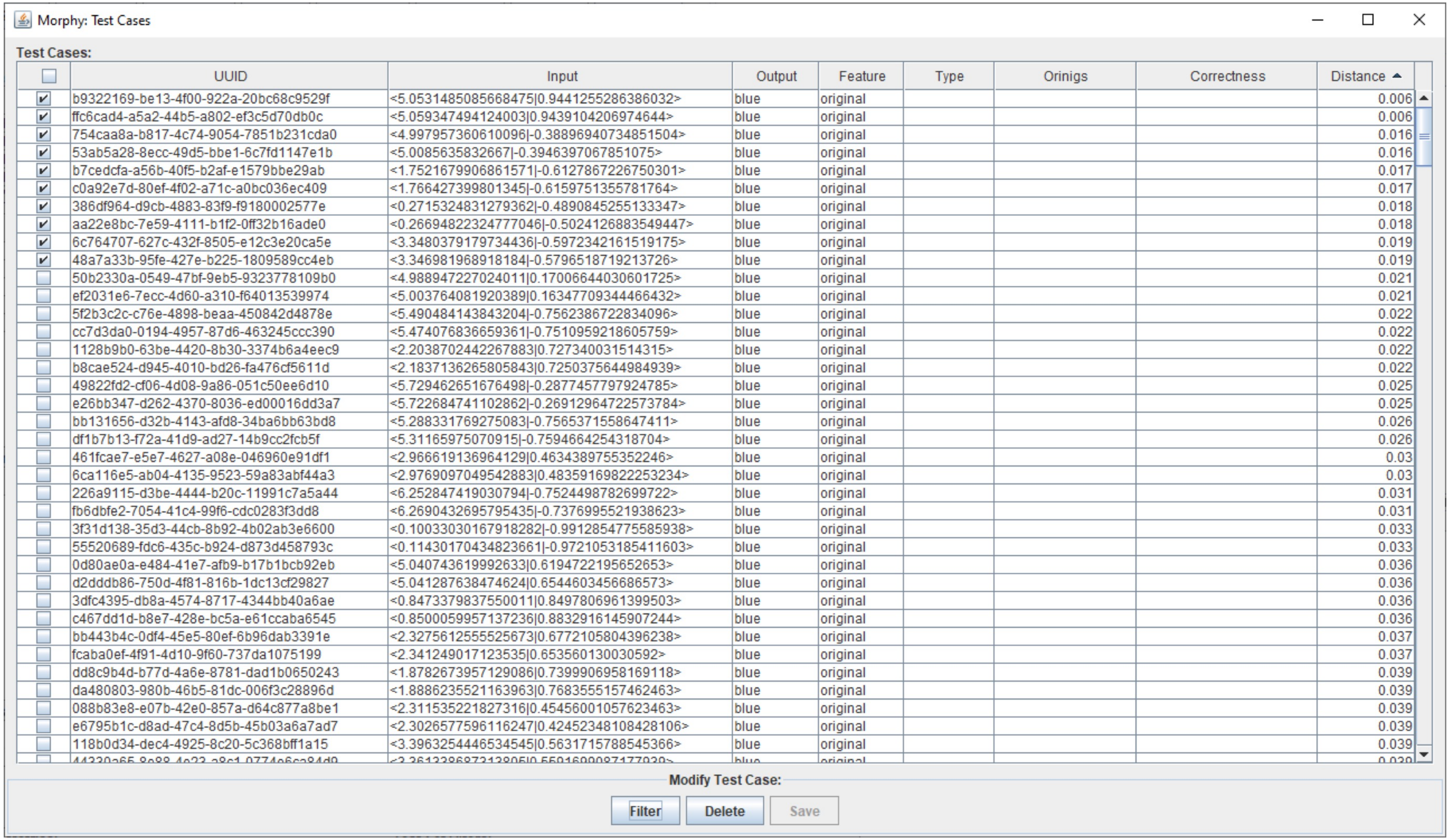
- When the system is started, it will restore the state of the last time it is used.
- When a new test specification is loaded, the system will initialise its state, so remove all the test cases in the current test set.

Automation at Activity Level



- **Seed**: Invoke selected seed maker morphism(s) to generate a set of seed test cases
- **Mutate**: Apply selected datamorphisms to the current test set to generate mutant test cases and add to the current test set
- **Edit Test**: View and edit the current test set
- **Filter**: Apply selected test set filter(s) to modify the current test set
- **Measure**: Apply selected test set metrics to measure the test quality
- **Execute**: Run the selected executer to run the program under test on the current set of test cases
- **Check**: Check the correctness of the test results against the selected metamorphisms
- **Analyse**: Invoke the selected analysers to generate test report

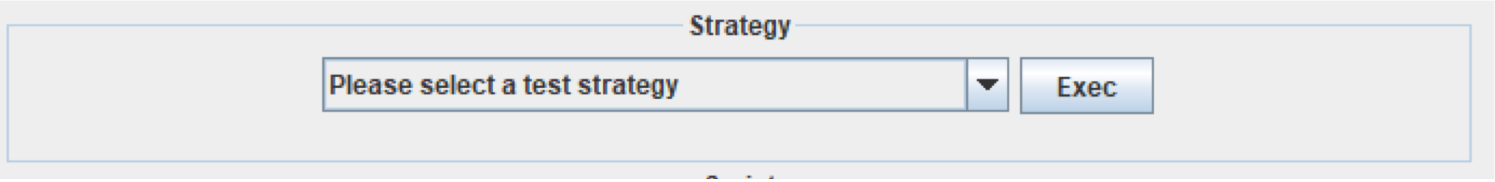
View and Edit Test Set



Test Cases:

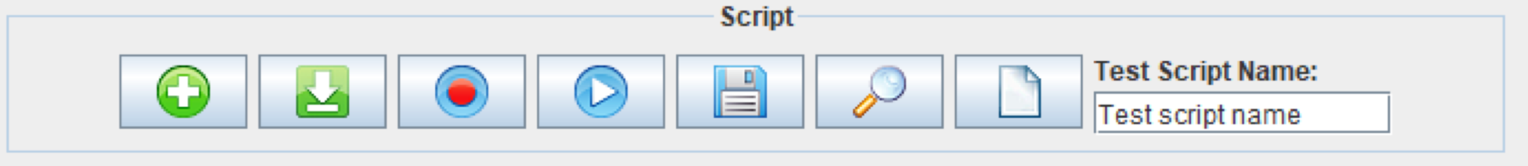
☐	UUID	Input	Output	Feature	Type	Orinigs	Correctness	Distance ▲
☑	b9322169-be13-4f00-922a-20bc68c9529f	<5.0531485085668475|0.9441255286386032>	blue	original				0.006
☑	ffc6cad4-a5a2-44b5-a802-ef3c5d70db0c	<5.059347494124003|0.9439104206974644>	blue	original				0.006
☑	754caa8a-b817-4c74-9054-7851b231cda0	<4.997957360610096|-0.38896940734851504>	blue	original				0.016
☑	53ab5a28-8ecc-49d5-bbe1-6c7fd1147e1b	<5.0085635832667|-0.3946397067851075>	blue	original				0.016
☑	b7cedcfa-a56b-40f5-b2af-e1579bbe29ab	<1.7521679906861571|-0.6127867226750301>	blue	original				0.017
☑	c0a92e7d-80ef-4f02-a71c-a0bc036ec409	<1.766427399801345|-0.6159751355781764>	blue	original				0.017
☑	386df964-d9cb-4883-83f9-f9180002577e	<0.2715324831279362|-0.4890845255133347>	blue	original				0.018
☑	aa22e8bc-7e59-4111-b1f2-0ff32b16ade0	<0.26694822324777046|-0.5024126883549447>	blue	original				0.018
☑	6c764707-627c-432f-8505-e12c3e20ca5e	<3.3480379179734436|-0.5972342161519175>	blue	original				0.019
☑	48a7a33b-95fe-427e-b225-1809589cc4eb	<3.346981968918184|-0.5796518719213726>	blue	original				0.019
☐	50b2330a-0549-47bf-9eb5-9323778109b0	<4.988947227024011|0.17006644030601725>	blue	original				0.021
☐	ef2031e6-7ecc-4d60-a310-f64013539974	<5.003764081920389|0.16347709344466432>	blue	original				0.021
☐	5f2b3c2c-c76e-4898-beaa-450842d4878e	<5.490484143843204|-0.7562386722834096>	blue	original				0.022
☐	cc7d3da0-0194-4957-87d6-463245ccc390	<5.474076836659361|-0.7510959218605759>	blue	original				0.022
☐	1128b9b0-63be-4420-8b30-3374b6a4eec9	<2.2038702442267883|0.727340031514315>	blue	original				0.022
☐	b8cae524-d945-4010-bd26-fa476cf5611d	<2.1837136265805843|0.7250375644984939>	blue	original				0.022
☐	49822fd2-cf06-4d08-9a86-051c50ee6d10	<5.729462651676498|-0.2877457797924785>	blue	original				0.025
☐	e26bb347-d262-4370-8036-ed00016dd3a7	<5.722684741102862|-0.26912964722573784>	blue	original				0.025
☐	bb131656-d32b-4143-afd8-34ba6bb63bd8	<5.288331769275083|-0.7565371558647411>	blue	original				0.026
☐	df1b7b13-f72a-41d9-ad27-14b9cc2fcb5f	<5.31165975070915|-0.7594664254318704>	blue	original				0.026
☐	461fcae7-e5e7-4627-a08e-046960e91df1	<2.966619136964129|0.4634389755352246>	blue	original				0.03
☐	6ca116e5-ab04-4135-9523-59a83abf44a3	<2.9769097049542883|0.48359169822253234>	blue	original				0.03
☐	226a9115-d3be-4444-b20c-11991c7a5a44	<6.252847419030794|-0.7524498782699722>	blue	original				0.031
☐	fb6dbfe2-7054-41c4-99f6-cdc0283f3dd8	<6.2690432695795435|-0.7376995521938623>	blue	original				0.031
☐	3f31d138-35d3-44cb-8b92-4b02ab3e6600	<0.10033030167918282|-0.9912854775585938>	blue	original				0.033
☐	55520689-fdc6-435c-b924-d873d458793c	<0.11430170434823661|-0.9721053185411603>	blue	original				0.033
☐	0d80ae0a-e484-41e7-afb9-b17b1bcb92eb	<5.040743619992633|0.6194722195652653>	blue	original				0.036
☐	d2dddb86-750d-4f81-816b-1dc13cf29827	<5.041287638474624|0.6544603456686573>	blue	original				0.036
☐	3dfc4395-db8a-4574-8717-4344bb40a6ae	<0.8473379837550011|0.8497806961399503>	blue	original				0.036
☐	c467dd1d-b8e7-428e-bc5a-e61ccaba6545	<0.8500059957137236|0.8832916145907244>	blue	original				0.036
☐	bb443b4c-0df4-45e5-80ef-6b96dab3391e	<2.3275612555525673|0.6772105804396238>	blue	original				0.037
☐	fcaba0ef-4f91-4d10-9f60-737da1075199	<2.341249017123535|0.653560130030592>	blue	original				0.037
☐	dd8c9b4d-b77d-4a6e-8781-dad1b0650243	<1.8782673957129086|0.7399906958169118>	blue	original				0.039
☐	da480803-980b-46b5-81dc-006f3c28896d	<1.8886235521163963|0.7683555157462463>	blue	original				0.039
☐	088b83e8-e07b-42e0-857a-d64c877a8be1	<2.311535221827316|0.45456001057623463>	blue	original				0.039
☐	e6795b1c-d8ad-47c4-8d5b-45b03a6a7ad7	<2.3026577596116247|0.42452348108428106>	blue	original				0.039
☐	118b0d34-dec4-4925-8c20-5c368bff1a15	<3.3963254446534545|0.5631715788545366>	blue	original				0.039

Automation at Strategy Level



- Three sets of test strategies have been implemented:
  - Mutant combination strategies
  - Exploratory strategies
  - Test optimisation strategies using genetic algorithms
- The user selects a strategy from the drop down menu, select the parameters as instructed, then press the execute button to run the selected test strategy.
- The execution process will be reported in the message panel.

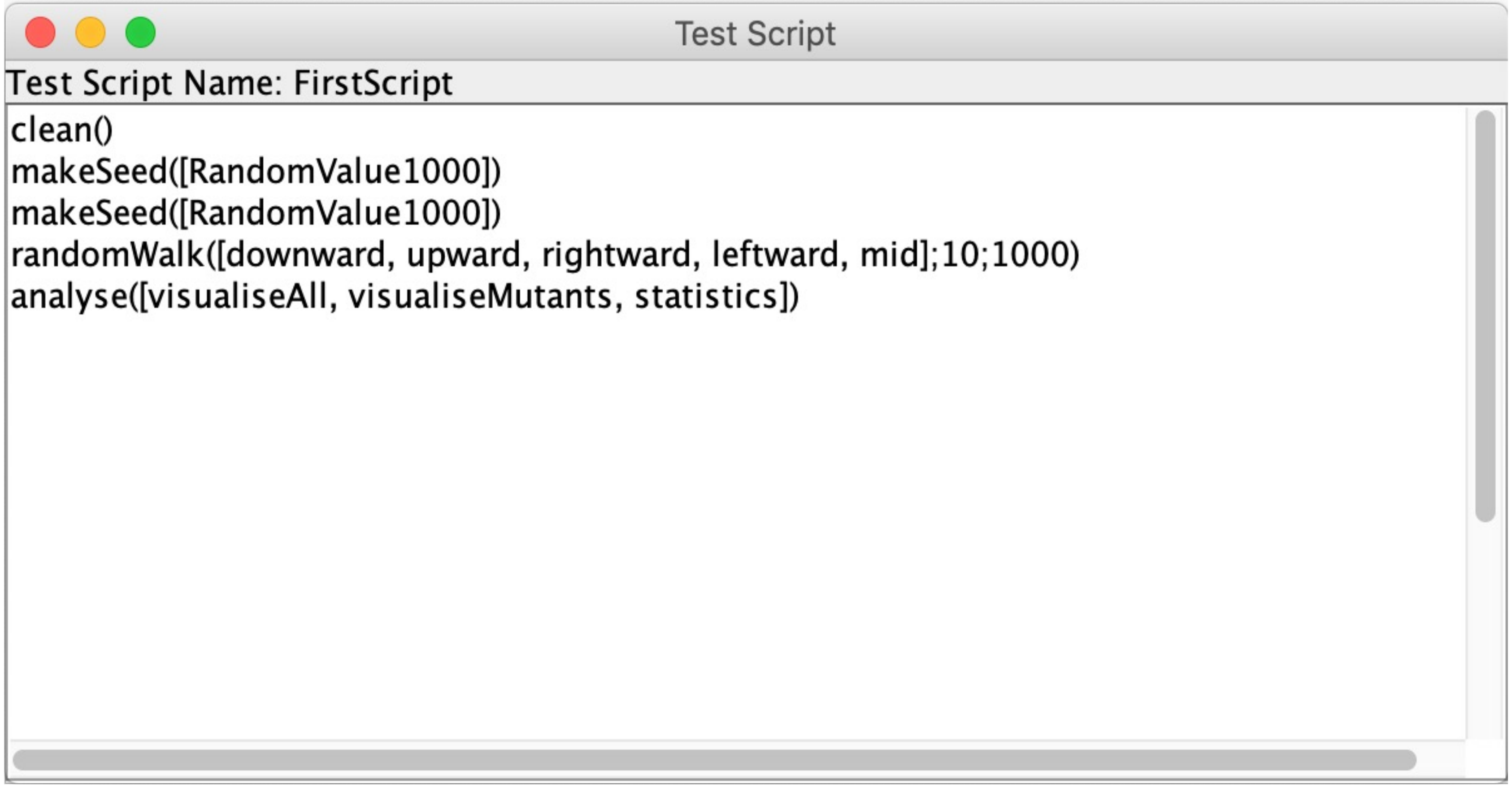
Automation at Process Level



Test Script Name: Test script name

- Create a new test script
- Load a previously saved test script
- Start recording interactive test activities
- Play the current test script
- Save the test script to a file
- View test script
- Clean the test script

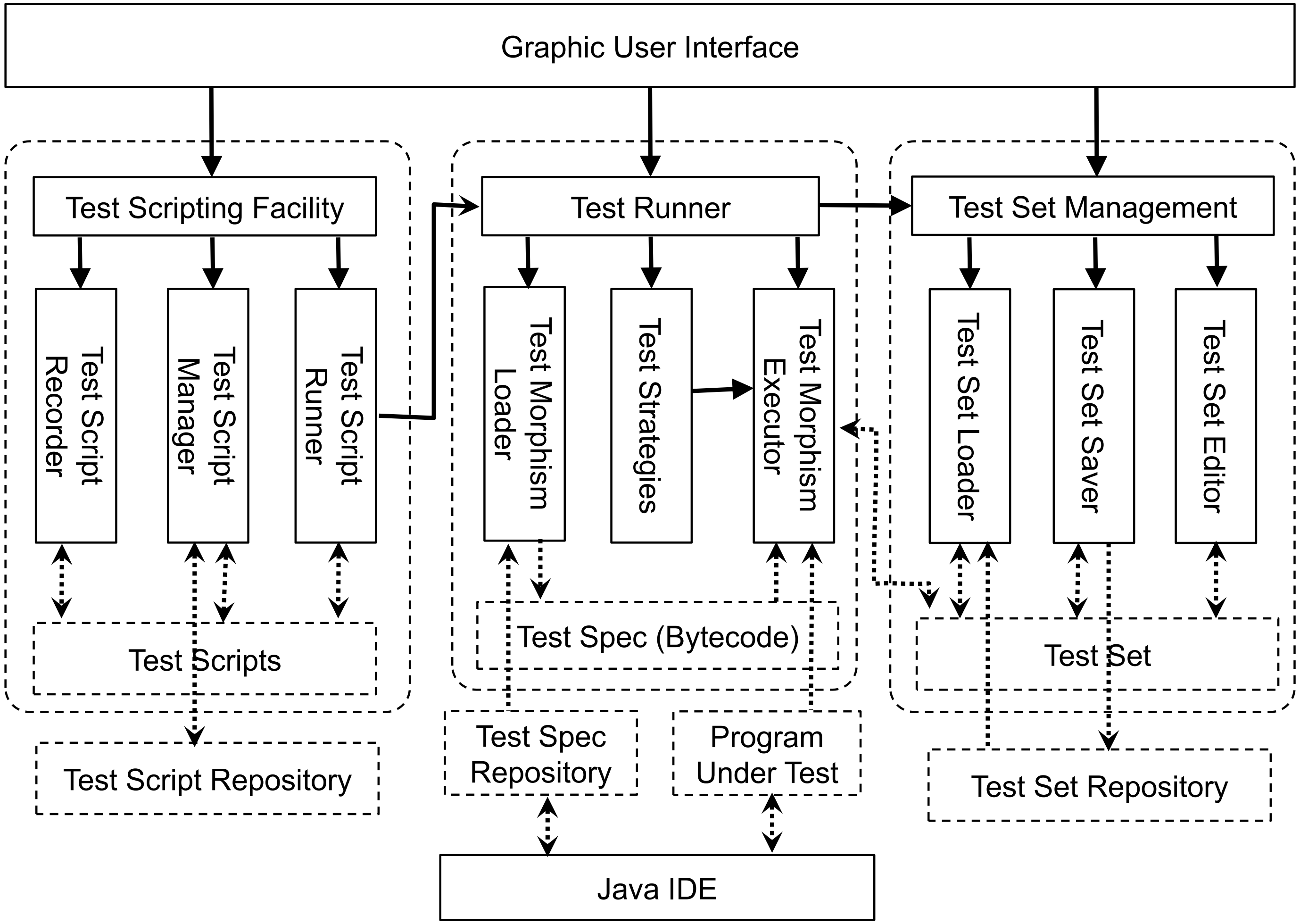
Example of Test Script



Test Script Name: FirstScript

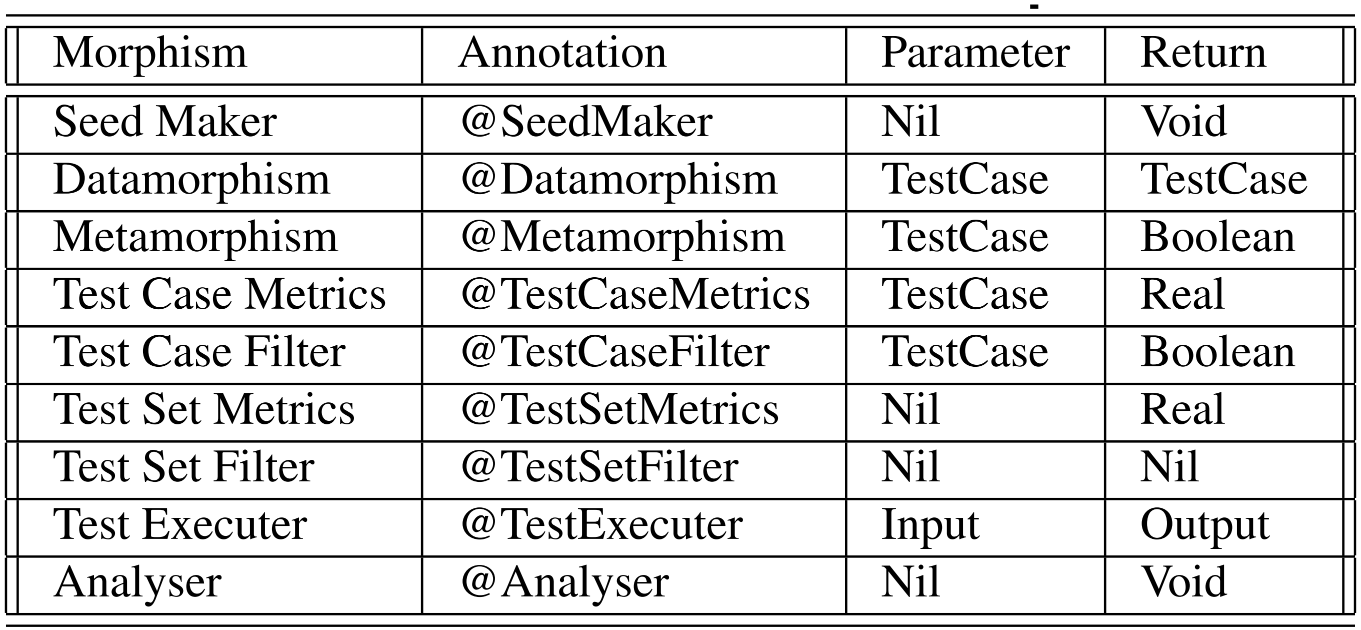
```
clean()
makeSeed([RandomValue1000])
makeSeed([RandomValue1000])
randomWalk([downward, upward, rightward, leftward, mid];10;1000)
analyse([visualiseAll, visualiseMutants, statistics])
```

Morphy’s Architecture



Morphy’s Format to Specify Test Systems

- Test Entities:
  - Java generic class *TestCase* for representing test cases
  - Java generic class *TestPool* for representing test suites/set
- Test Morphisms:
  - Java methods annotated with metadata



Morphism	Annotation	Parameter	Return
Seed Maker	@SeedMaker	Nil	Void
Datamorphism	@Datamorphism	TestCase	TestCase
Metamorphism	@Metamorphism	TestCase	Boolean
Test Case Metrics	@TestCaseMetrics	TestCase	Real
Test Case Filter	@TestCaseFilter	TestCase	Boolean
Test Set Metrics	@TestSetMetrics	Nil	Real
Test Set Filter	@TestSetFilter	Nil	Nil
Test Executer	@TestExecuter	Input	Output
Analyser	@Analyser	Nil	Void

Example 1. Triangle Classification

- The Program under Test:

  "reads three integer values from an input dialog. The three values represent the lengths of the sides of a triangle. The program displays a message that states whether the triangle is scalene, isosceles, or equilateral."

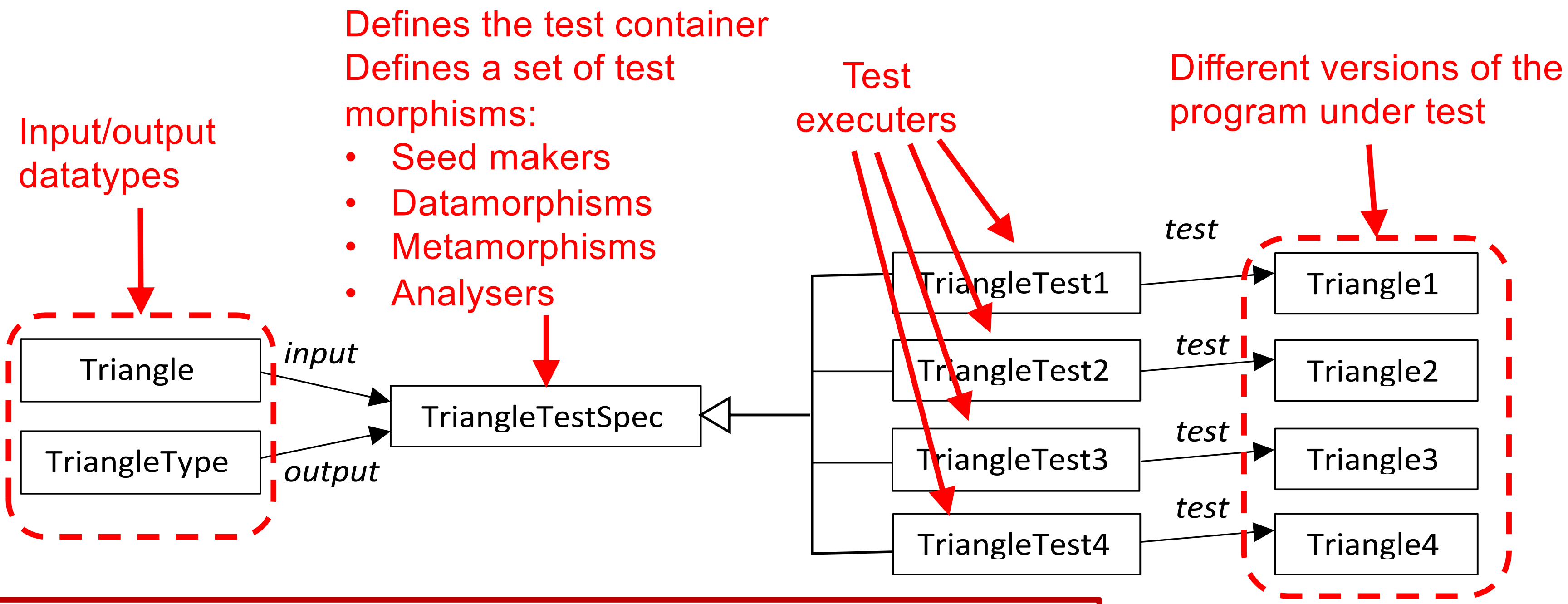
  [G. J. Myers. *The Art of Software Testing*. John Wiley & Sons, Inc., 1st edt, 1979, 2nd edt, 2004.]

- The Problem of Testing:

  Myer listed 14 questions for testers to assess how well he/she tests the program for such a seemly simple program and reported that *"highly qualified professional programmers score, on the average, only 7.8 out of a possible 14"*.

- The Research Questions:
  - Can datamorphic testing achieve a good score?
  - Can Morphy automate this testing process?

Structure of the Test System



Input/output datatypes

Defines the test container
Defines a set of test morphisms:

- Seed makers
- Datamorphisms
- Metamorphisms
- Analysers

Test executers

Different versions of the program under test

```
@TestSetContainer(
   inputTypeName = "Triangle",
   outputTypeName = "TriangleType"
)
public TestPool<Triangle, TriangleType> testSuite
   = new TestPool<Triangle, TriangleType>();
```

The Input and Output Datatypes

```java
package morphy.examples;
public class Triangle {
   public int x =0;
   public int y =0;
   public int z =0;

   public Triangle() {
      x=0; y=0; z=0;
   }

   public Triangle(int a, int b, int c) {
      x=a; y=b; z=c;
   }

   public String toString() {
      String str = "<"+x+"|"+y+"|"+z+">";
      return str;
   }

   public void valueOf(String str) { … }
}
```

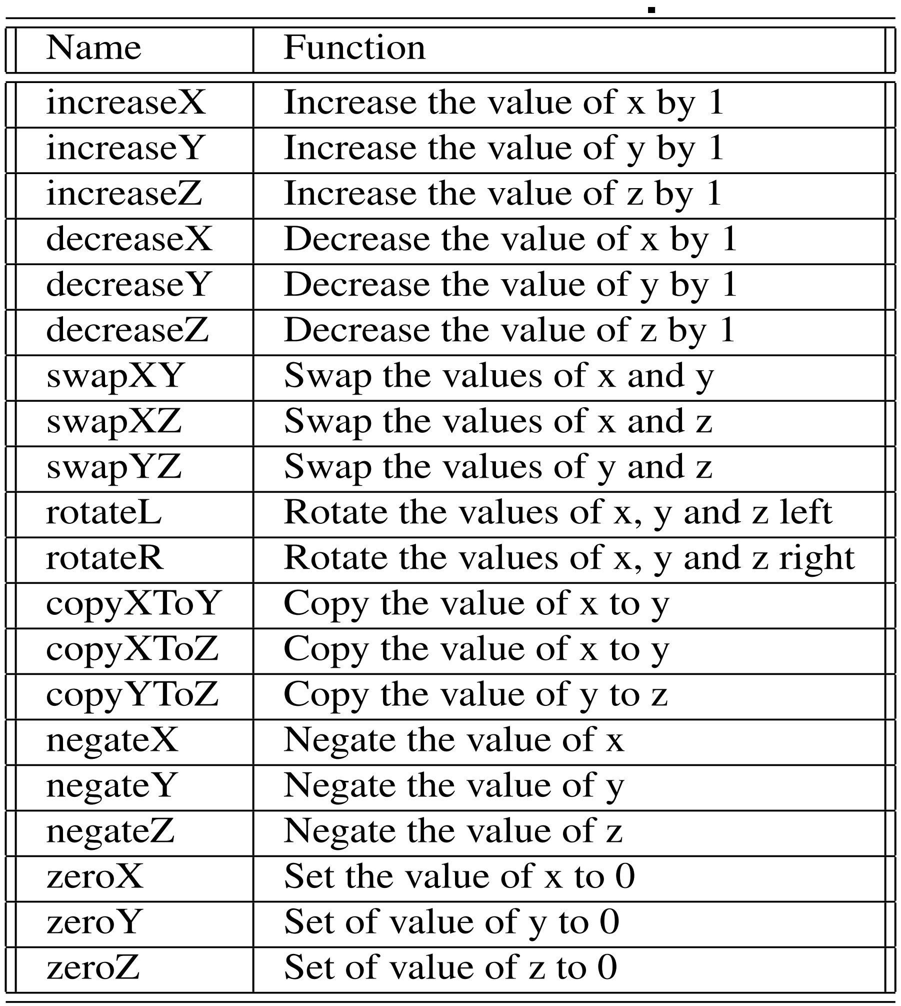
```java
package morphy.examples;
public enum TriangleType {
   equilaterial,
   isoscelene,
   scalene,
   noneTriangle
};
```

Seed Makers

- Four methods were coded to generate seed test cases
  - Literal constants without expected output
  - Literal constants with expected output
  - Manual input
  - Read test cases from a file
- Example:

```
@MakeSeed
public void makeSeeds(){
   testSuite.addInput(new Triangle(5,5,5));
   testSuite.addInput(new Triangle(5,5,7));
   testSuite.addInput(new Triangle(5,7,9));
   testSuite.addInput(new Triangle(3,5,9));
}
```

Datamorphisms



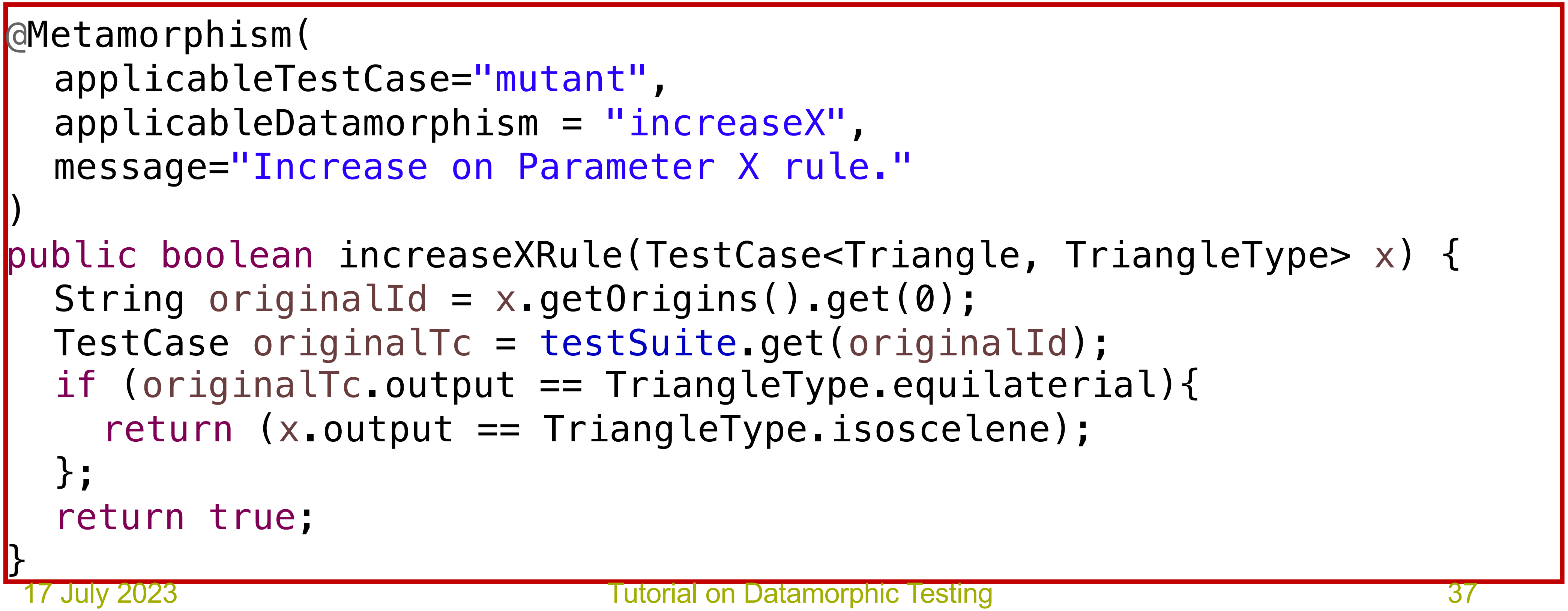
Name	Function
increaseX	Increase the value of x by 1
increaseY	Increase the value of y by 1
increaseZ	Increase the value of z by 1
decreaseX	Decrease the value of x by 1
decreaseY	Decrease the value of y by 1
decreaseZ	Decrease the value of z by 1
swapXY	Swap the values of x and y
swapXZ	Swap the values of x and z
swapYZ	Swap the values of y and z
rotateL	Rotate the values of x, y and z left
rotateR	Rotate the values of x, y and z right
copyXToY	Copy the value of x to y
copyXToZ	Copy the value of x to y
copyYToZ	Copy the value of y to z
negateX	Negate the value of x
negateY	Negate the value of y
negateZ	Negate the value of z
zeroX	Set the value of x to 0
zeroY	Set of value of y to 0
zeroZ	Set of value of z to 0

Example of Datamorphism

```java
@Datamorphism
public TestCase<Triangle, TriangleType>
  increaseX(TestCase<Triangle, TriangleType> seed){
    TestCase<Triangle, TriangleType> mutant = new
      TestCase<Triangle, TriangleType>();
    Triangle m = new Triangle(1,1,1);
    m.x=seed.input.x+1;
    m.y=seed.input.y;
    m.z=seed.input.z;
    mutant.input = m;
    return mutant;
}
```

Metamorphisms

- There is a metamorphism for test cases generated by the literal constant with expected output to compare the execution results again the expected output
- For each datamorphism, there is a corresponding metamorphism to check correctness of the test output on the mutant test cases



```
@Metamorphism(
  applicableTestCase="mutant",
  applicableDatamorphism = "increaseX",
  message="Increase on Parameter X rule."
)
public boolean increaseXRule(TestCase<Triangle, TriangleType> x) {
  String originalId = x.getOrigins().get(0);
  TestCase originalTc = testSuite.get(originalId);
  if (originalTc.output == TriangleType.equilaterial){
    return (x.output == TriangleType.isoscelene);
  };
  return true;
}
```

Test Executer

```java
package morphy.examples;
public class Triangle1 {
   public int x, y, z;
   public Triangle1(int a, int b, int c) {
      x = a; y = b; z = c;
   }
   public TriangleType Classify() { … }
}
```

The code under test

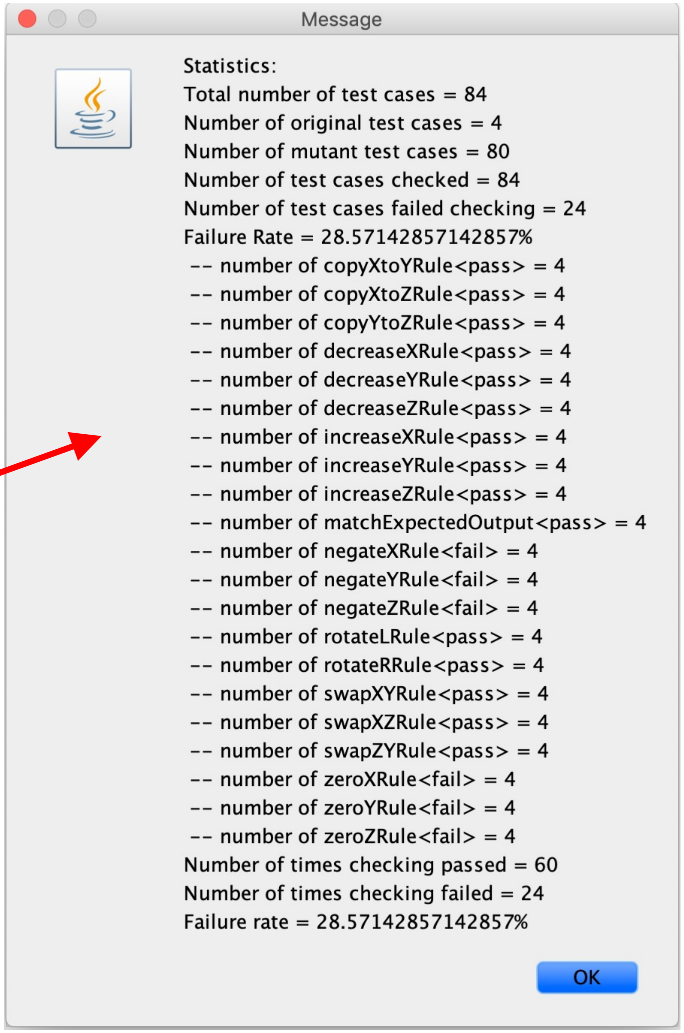
```java
package morphy.examples;
import morphy.annotations.*;

public class TriangleTest1 extends TriangleTestSpec {
   @TestExecuter
   public TriangleType TriangleClassifier1(Triangle tc) {
      int x = tc.x;
      Triangle1 tx = new Triangle1(x, tc.y, tc.z);
      return tx.Classify();
   }
}
```

Test Result Analyser

An analyser was written to analyse the test results on a test set statistically.

```
@Analyser
public void
statisticsOfCorrectness() { … }
```

An output of the analyser



```
Statistics:
Total number of test cases = 84
Number of original test cases = 4
Number of mutant test cases = 80
Number of test cases checked = 84
Number of test cases failed checking = 24
Failure Rate = 28.571428571428573%
 -- number of copyXtoYRule<pass> = 4
 -- number of copyXtoZRule<pass> = 4
 -- number of copyYtoZRule<pass> = 4
 -- number of decreaseXRule<pass> = 4
 -- number of decreaseYRule<pass> = 4
 -- number of decreaseZRule<pass> = 4
 -- number of increaseXRule<pass> = 4
 -- number of increaseYRule<pass> = 4
 -- number of increaseZRule<pass> = 4
 -- number of matchExpectedOutput<pass> = 4
 -- number of negateXRule<fail> = 4
 -- number of negateYRule<fail> = 4
 -- number of negateZRule<fail> = 4
 -- number of rotateLRule<pass> = 4
 -- number of rotateRRule<pass> = 4
 -- number of swapXYRule<pass> = 4
 -- number of swapXZRule<pass> = 4
 -- number of swapZYRule<pass> = 4
 -- number of zeroXRule<fail> = 4
 -- number of zeroYRule<fail> = 4
 -- number of zeroZRule<fail> = 4
Number of times checking passed = 60
Number of times checking failed = 24
Failure rate = 28.571428571428573%
```

Example 2: Trigonometric Functions

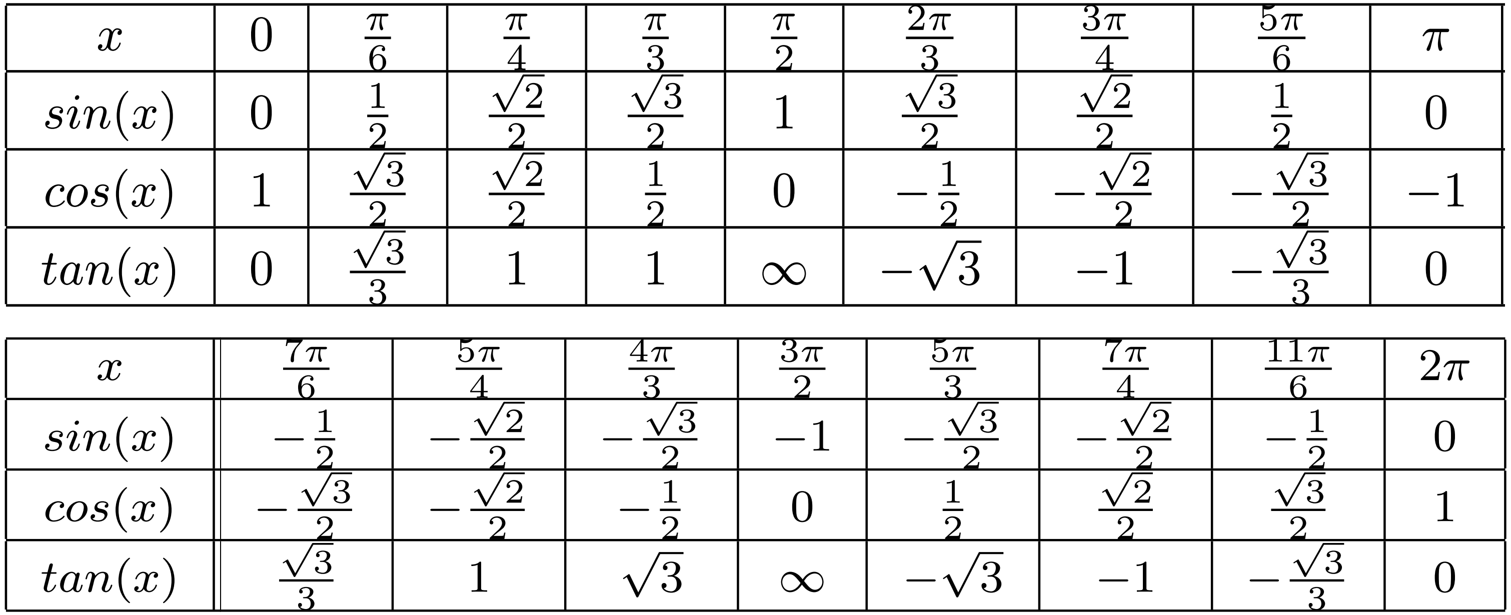
- Programs under Test:
  - Three trigonometric functions $Sin(x)$, $Cos(x)$ and $Tan(x)$ provided by Java math library
- Problem of Test:
  - Can such functions be tested for their accuracy and correctness?
- Solution in the damamorphic testing approach:
  - Test on specific input values that the output value is known
  - Test on random input test values to check if algebraic laws of these functional are held
    - Laws involve multiple invocation of the same function
    - Laws invoke multiple functions on the same input values
    - Laws invoke multiple functions on different input values

Seed Makers

Two seed makers are written:

1. Generate a number of random real numbers (without expected outputs)
2. Generate a set of special input values and the corresponding expected output

Special Input Values and Expected Outputs



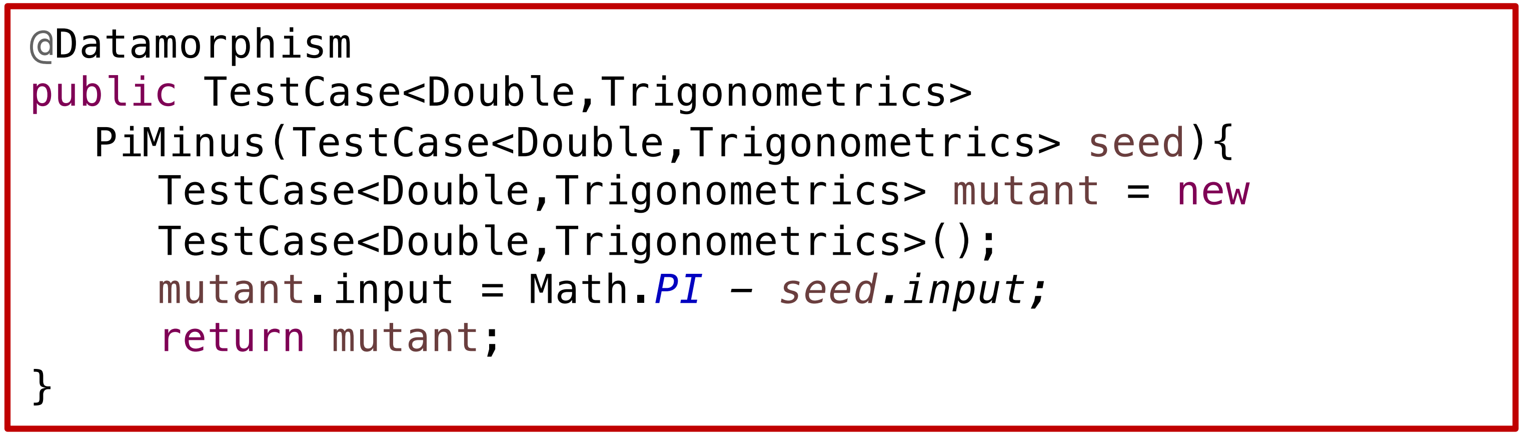
x	0	$\frac{\pi}{6}$	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$	$\frac{2\pi}{3}$	$\frac{3\pi}{4}$	$\frac{5\pi}{6}$	π
$sin(x)$	0	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	1	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$	0
$cos(x)$	1	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$	$-\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{3}}{2}$	-1
$tan(x)$	0	$\frac{\sqrt{3}}{3}$	1	1	∞	$-\sqrt{3}$	-1	$-\frac{\sqrt{3}}{3}$	0

x	$\frac{7\pi}{6}$	$\frac{5\pi}{4}$	$\frac{4\pi}{3}$	$\frac{3\pi}{2}$	$\frac{5\pi}{3}$	$\frac{7\pi}{4}$	$\frac{11\pi}{6}$	2π
$sin(x)$	$-\frac{1}{2}$	$-\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{3}}{2}$	-1	$-\frac{\sqrt{3}}{2}$	$-\frac{\sqrt{2}}{2}$	$-\frac{1}{2}$	0
$cos(x)$	$-\frac{\sqrt{3}}{2}$	$-\frac{\sqrt{2}}{2}$	$-\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	1
$tan(x)$	$\frac{\sqrt{3}}{3}$	1	$\sqrt{3}$	∞	$-\sqrt{3}$	-1	$-\frac{\sqrt{3}}{3}$	0

Datamorphisms

Name	Function	Name	Function
halfPiPlus	$x \rightarrow \pi/2 + x$	halfPiMinus	$x \rightarrow \pi/2 - x$
piPlus	$x \rightarrow \pi + x$	piMinus	$x \rightarrow \pi - x$
twoPiPlus	$x \rightarrow 2\pi + x$	twoPiMinus	$x \rightarrow 2\pi - x$
sum	$(x, y) \rightarrow x + y$	diff	$(x, y) \rightarrow x - y$
negate	$x \rightarrow -x$		

Example: Implementation of datamorphisms



```
@Datamorphism
public TestCase<Double,Trigonometrics>
   PiMinus(TestCase<Double,Trigonometrics> seed){
      TestCase<Double,Trigonometrics> mutant = new
      TestCase<Double,Trigonometrics>();
      mutant.input = Math.PI - seed.input;
      return mutant;
}
```

Metamorphisms

Algebraic laws of Trigonometric Functions

$$sin(x+y) = sin(x)cos(y) + cos(x)sin(y) \qquad cos(x+y) = cos(x)cos(y) - sin(x)sin(y)$$

$$sin(x-y) = sin(x)cos(y) - cos(x)sin(y) \qquad cos(x-y) = cos(x)cos(y) + sin(x)sin(y)$$

$$tan(x+y) = \frac{tan(x)+tan(y)}{1-tan(x)tan(y)} \qquad tan(x-y) = \frac{tan(x)-tan(y)}{1+tan(x)tan(y)}$$

$$sin(\pi+x) = -sin(x) \qquad cos(\pi+x) = -cos(x) \qquad tan(\pi+x) = tan(x)$$

$$sin(\pi-x) = sin(x) \qquad cos(\pi-x) = -cos(x) \qquad tan(\pi-x) = -tan(x)$$

$$sin(\pi/2+x) = cos(x) \qquad cos(\pi/2+x) = -sin(x) \qquad tan(\pi/2+x) = -1/tan(x)$$

$$sin(\pi/2-x) = cos(x) \qquad cos(\pi/2-x) = sin(x) \qquad tan(\pi/2-x) = 1/tan(x)$$

$$sin(2\pi-x) = -sin(x) \qquad cos(2\pi-x) = cos(x) \qquad tan(2\pi-x) = -tan(x)$$

$$sin(2\pi+x) = sin(x) \qquad cos(2\pi+x) = cos(x) \qquad tan(2\pi+x) = tan(x)$$

$$sin(-x) = -sin(x) \qquad cos(-x) = cos(x) \qquad tan(-x) = -tan(x)$$

Examples: Implementations of Metamorphisms

OXFORD BROOKES UNIVERSITY

```
@Metamorphism(
   applicableTestCase="seed",
   message="Special Sin(x) value")
public boolean specialSinValueAssertion(TestCase<Double,
Trigonometrics> tc) {
   if (expected.get(tc.id).output == null) { return true; };
   return (Math.abs(tc.output.sin - expected.get(tc.id).output.sin)
     < error) ;
}
```

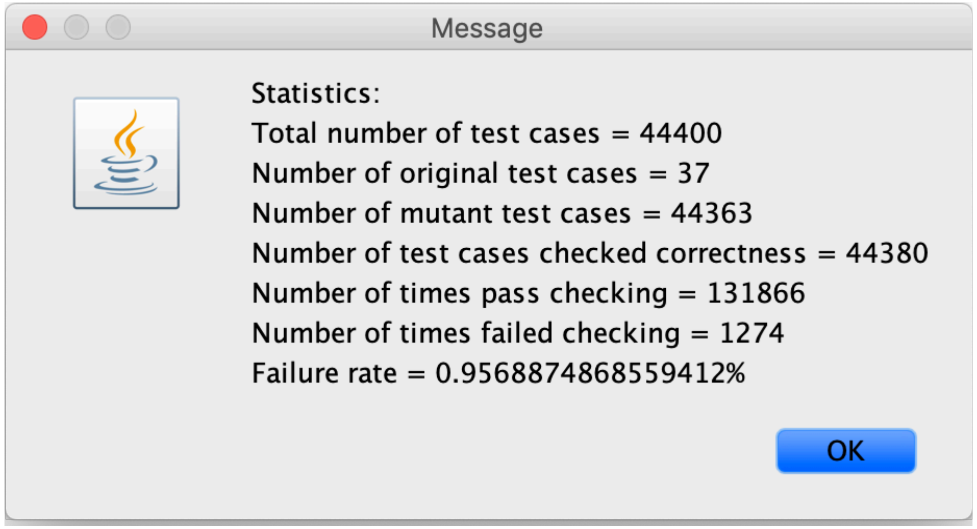
Compares program output against expected output on special values

```
@Metamorphism(
   applicableTestCase="mutant",
   applicableDatamorphism="HalfPiPlus",
   message="The rule: Sin(pi/2+x) = Cos(x)"
)
public boolean HalfPiPlusSinAssertion(TestCase<Double,
Trigonometrics> tc) {
   TestCase<Double, Trigonometrics> originalTc =
      testSuite.get(tc.getOrigins().get(0));
   return (Math.abs(tc.output.sin - originalTc.output.cos)
      <= error);
}
```

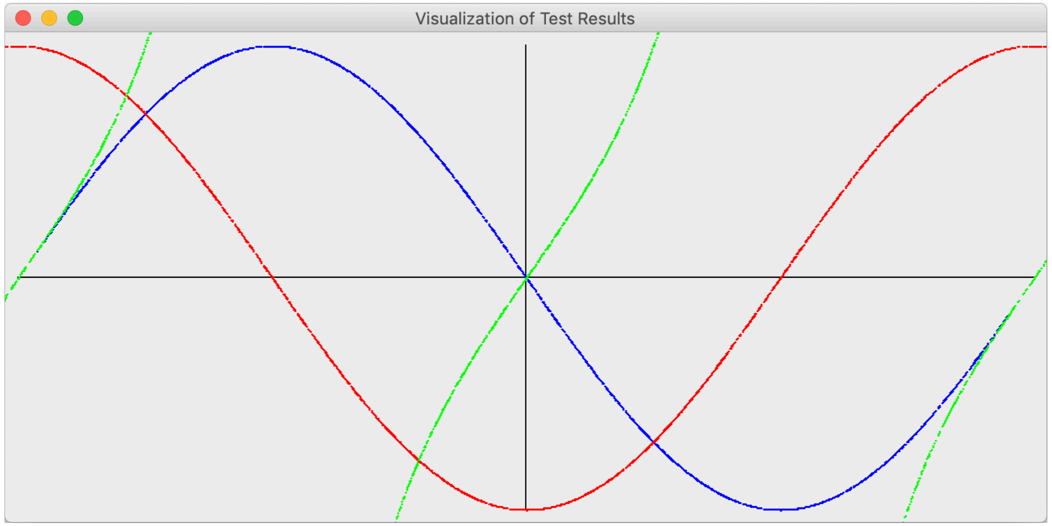
Checks if an algebraic law is held.

Test Result Analysers

- Two test result analysers:
  - Statistical analysis of test result:
    - Reused (simplified) a part of the analyser developed for Triangle Classification case study
  - Visual display of the functions



```
Statistics:
Total number of test cases = 44400
Number of original test cases = 37
Number of mutant test cases = 44363
Number of test cases checked correctness = 44380
Number of times pass checking = 131866
Number of times failed checking = 1274
Failure rate = 0.9568874868559412%
```



Part 3

Test Strategies for Machine Learning Applications

1. Scenario-Based Confirmatory Testing
2. Exploratory Testing of ML Classifiers
3. Scenario-based Exploratory Functional Testing

Scenario-Based Confirmatory Testing

1) Development process of test systems
2) Strategies to combine scenarios and adequacy criteria
3) Algorithms to generate adequate test sets

Test System Development Process

OXFORD BROOKES UNIVERSITY

Stage 1: Analysis

Analysis of the testing problem to design a test system

- Identify the seed test cases
- Identify the datamorphisms
- Identify the metamorphisms

Stage 2: Realisation

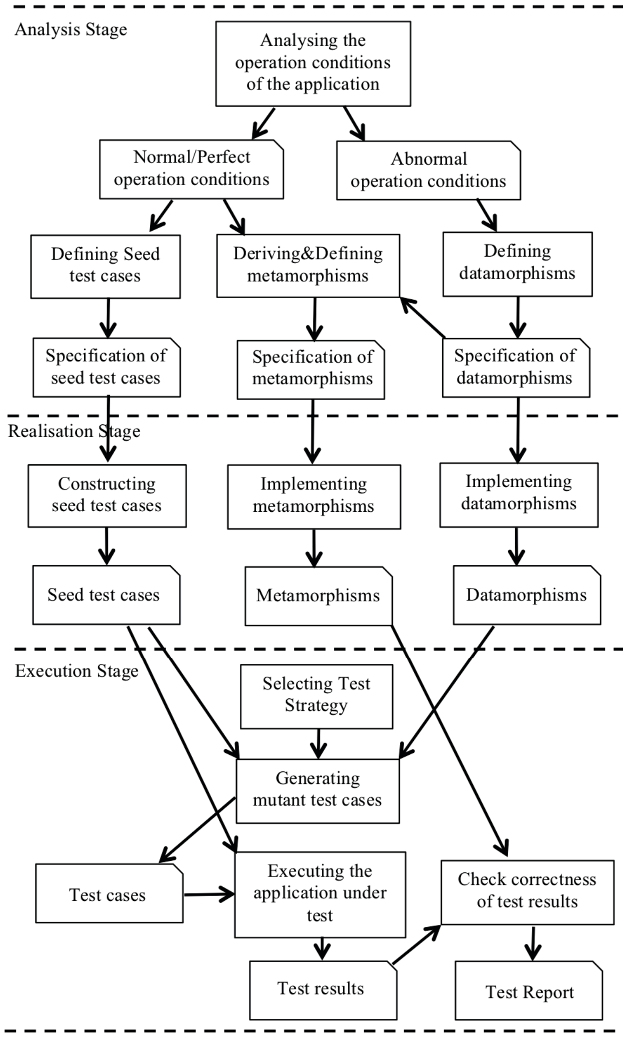
Realisation of the elements in the test system

- Collecting seed test data
- Implementation of datamorphisms
- Implementation of metamorphisms

Stage 3: Execution

Execution of testing using the test system

- Selection of test adequacy criteria
- Generating test cases
- Execution of test



Analysis Stage

Analysing the operation conditions of the application

Normal/Perfect operation conditions

Abnormal operation conditions

Defining Seed test cases

Deriving&Defining metamorphisms

Defining datamorphisms

Specification of seed test cases

Specification of metamorphisms

Specification of datamorphisms

Realisation Stage

Constructing seed test cases

Implementing metamorphisms

Implementing datamorphisms

Seed test cases

Metamorphisms

Datamorphisms

Execution Stage

Selecting Test Strategy

Generating mutant test cases

Test cases

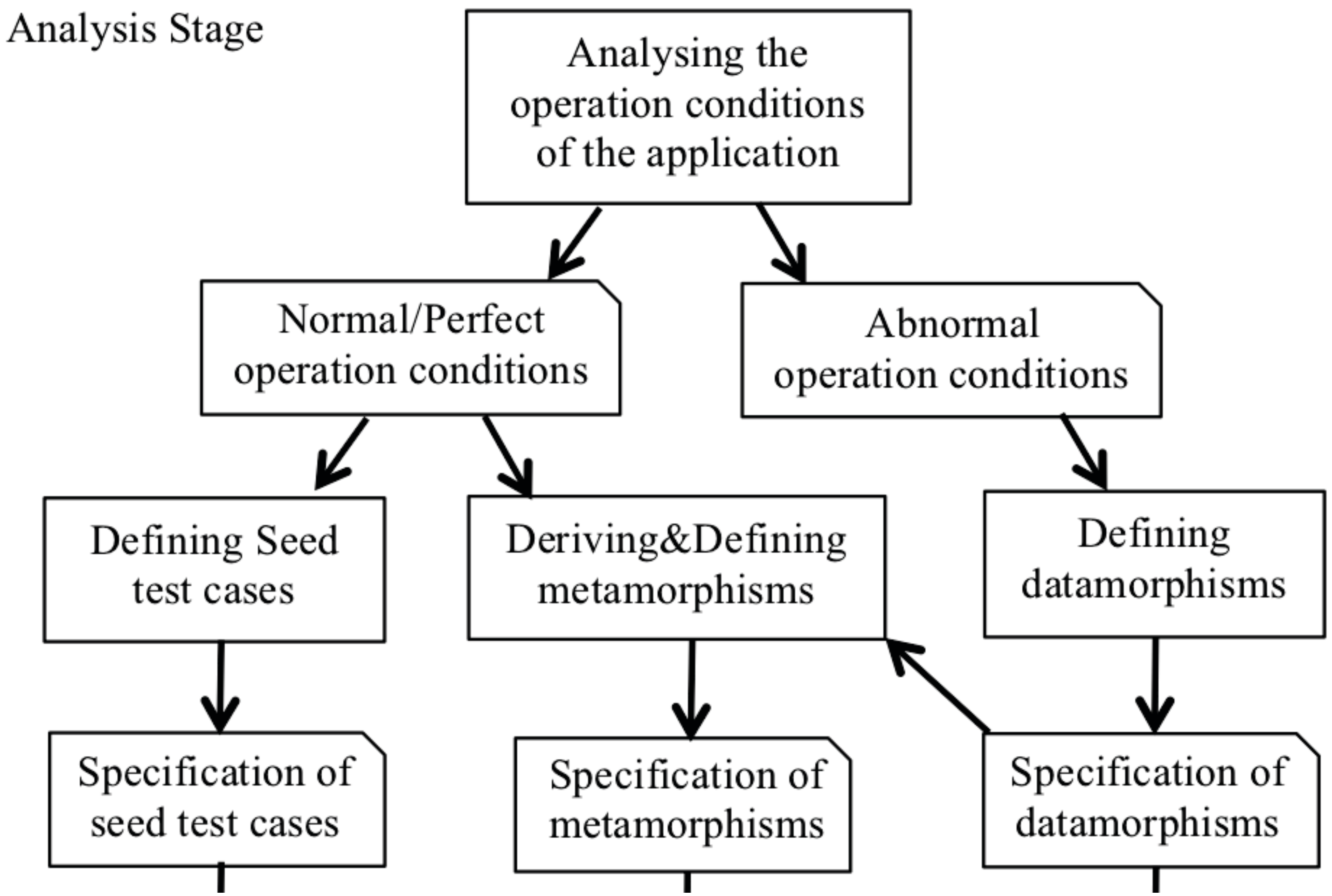
Executing the application under test

Check correctness of test results

Test results

Test Report

Stage 1: Analysis


Analysis Stage
Analysing the operation conditions of the application
Normal/Perfect operation conditions
Abnormal operation conditions
Defining Seed test cases
Deriving&Defining metamorphisms
Defining datamorphisms
Specification of seed test cases
Specification of metamorphisms
Specification of datamorphisms

Example: Face Recognition (1)

- Usage 1: Automated passport control at airport
- Usage 2: Detect criminal suspects using images from surveillance cameras

Operation Conditions	Usage 1	Usage 2
Front face images in database	✔	✔
Side face images in database		✔
Face image of older age	✔	✔
Face image sun tanned	✔	✔
Face image in different hair style/colour	✔	✔
Face image wearing makeup	✔	✔
Face image with sunglasses		✔
Face image with beard	✔	✔
Variable lighting and background		✔
Face image from a side angle		✔
Face image from an upper angle		✔
Image from artist drawings		✔

Example: Face Recognition (2)

- Seed test cases:
  - A set of photos of human faces in different races, ages, genders, etc.
- Datamorphisms:
  - Add a pair of glasses;
  - Add makeup;
  - Change the background;
  - Change the illumination;
  - Change hair style;
  - Change hair colour;
  - Swap: replace a part of the image with another person’s image.

- *φ(x)* is any of the datamorphisms given in the Table
- *FaceSimile* is any of the face recognition application under test

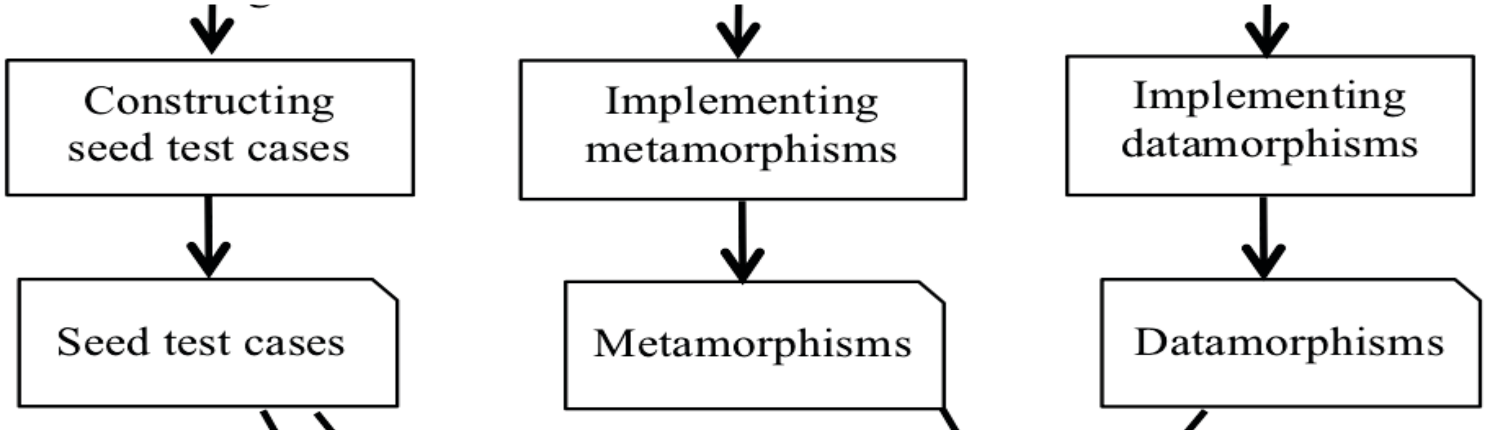
- Metamorphisms:

$$FaceSimile(x, \varphi(x)) \geq 80\%$$

$$FaceOf(Swap(x,y))=FaceOf(x) \ \ \textbf{and} \ FaceOf(Swap(x,y))=FaceOf(y)$$

Depends on the application

Stage 2: Realisation

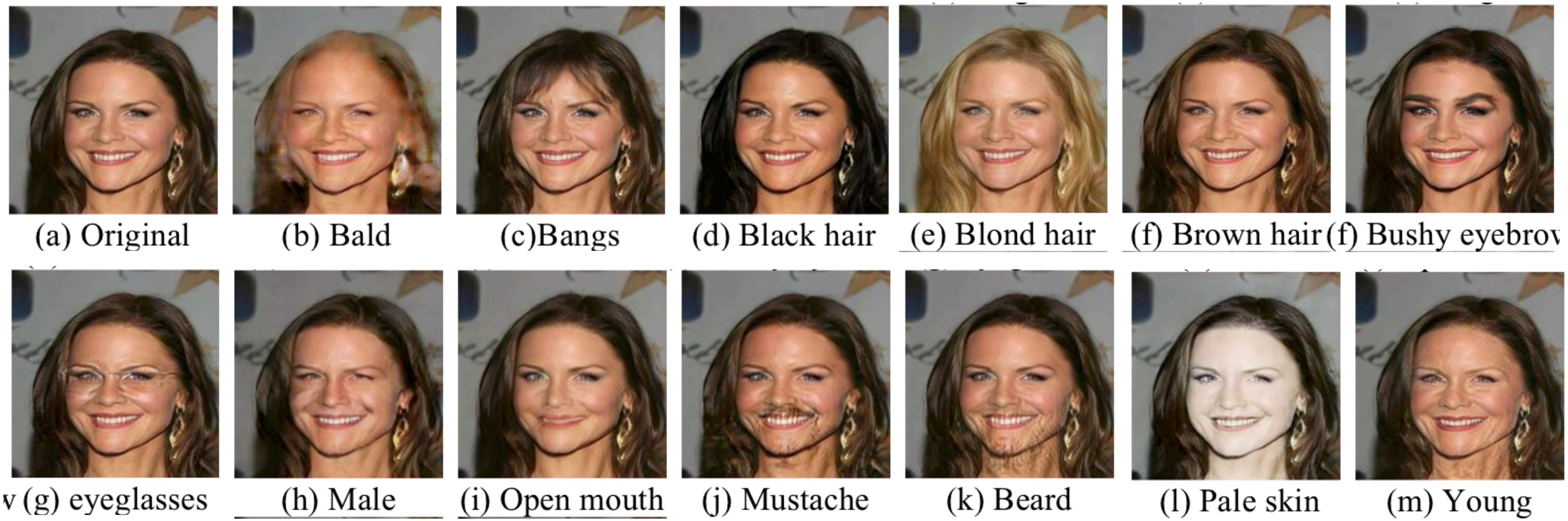

Constructing seed test cases
Implementing metamorphisms
Implementing datamorphisms
Seed test cases
Metamorphisms
Datamorphisms

- Seeds:
  - Often available from other development activities, such as training data, benchmarks
  - Can be collected from the real world, though costly
  - Could be manual effort
- Datamorphisms:
  - Often can be implemented as small program code fairly easily
  - Many application domains have open source, library, etc. available
- Metamorphisms:
  - Often easy to implement as small program code fairly easily

Example: Datamorphims of Images

- ***Seed test case (a):***

  A photo in the Public dataset *Labeled Faces in the Wild at* URL: http://vis-www.cs.umass.edu/lfw/



(a) Original (b) Bald (c)Bangs (d) Black hair (e) Blond hair (f) Brown hair (f) Bushy eyebrov

v (g) eyeglasses (h) Male (i) Open mouth (j) Mustache (k) Beard (l) Pale skin (m) Young

- ***Mutants: (b – j)***

  A subset of the photos obtained from the seed by manipulations of the seed photo.

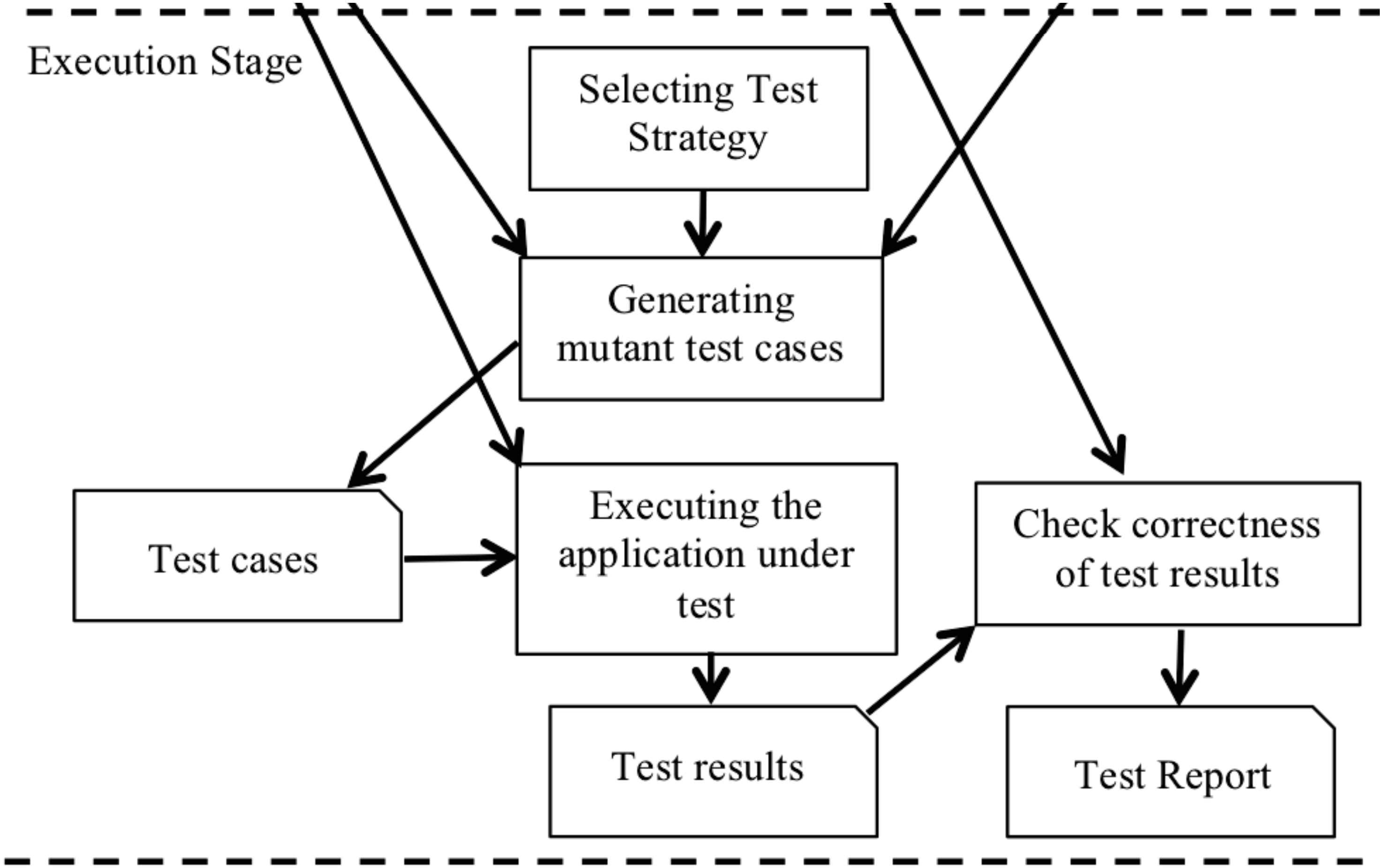
Implementation of The Datamorphisms

OXFORD BROOKES UNIVERSITY

AttGAN's Face Attribute Editing Operators

Operation	***Meaning***
Bald	Change the facial image into bald
Bangs	Add bangs to the facial image
Black Hair	Change the hair colour into black
Blond Hair	Change the hair colour into blond
Brown Hair	Change the hair colour into brown
Bushy Eyebrows	Change the eyebrows to be bushy
Eyeglasses	Add eyeglasses to the image
Male	Change the image from female to male
Mouth Open	Change the mouth to be slightly open
Mustache	Add or remove mustache to the facial image
Beard	Add or remove beard
Pale Skin	Make the skin tone to be pale
Young	Change the image to look younger

Stage 3: Execution


Execution Stage
Selecting Test Strategy
Generating mutant test cases
Test cases
Executing the application under test
Check correctness of test results
Test results
Test Report

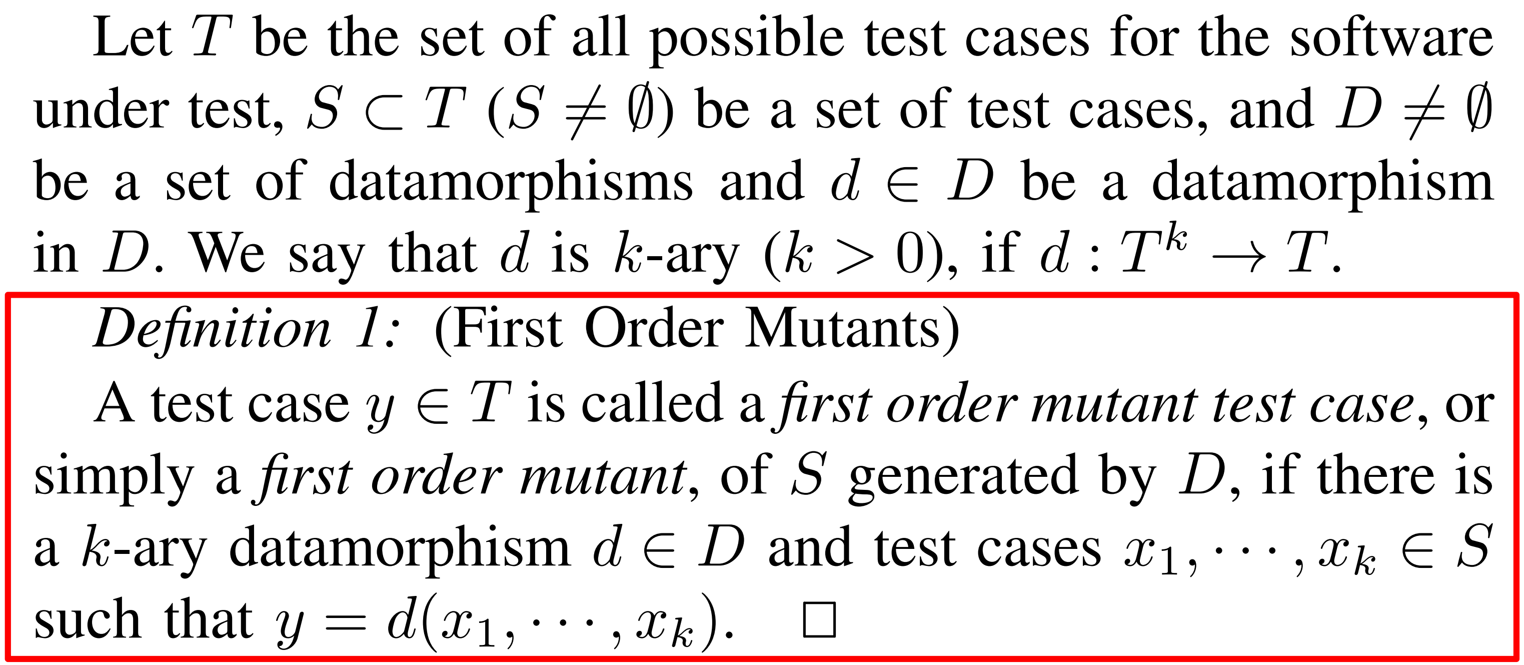
Mutant Combination Strategies

- **Basic Ideas**
  - Uses seed test cases to test the normal operation condition of the AI system under test,
  - Uses datamorphisms to transform a test case that represents other operation conditions that can be derived from the normal operation conditions.
  - Combining datamorphisms means combinations of different operation conditions.
- **Examples**
  - For testing face recognition applications: datamorphisms are used to transform the images of human faces by editing the facial attributes, such as adding makeup, wearing glasses, changing skin tunes, change hair styles and colour, etc.
  - For testing driverless vehicles in [Tian et al. 2018], datamorphisms are developed to alter the weather condition of a recorded driving process to be in fog, to transform the lighting condition from daytime to night time with street lights, etc.

First Order Mutant Coverage

➢ The Notion of First Order Mutants

- First order mutants are mutant test cases generated from seed test cases.
- Each first order mutant represents one operation condition of the system.

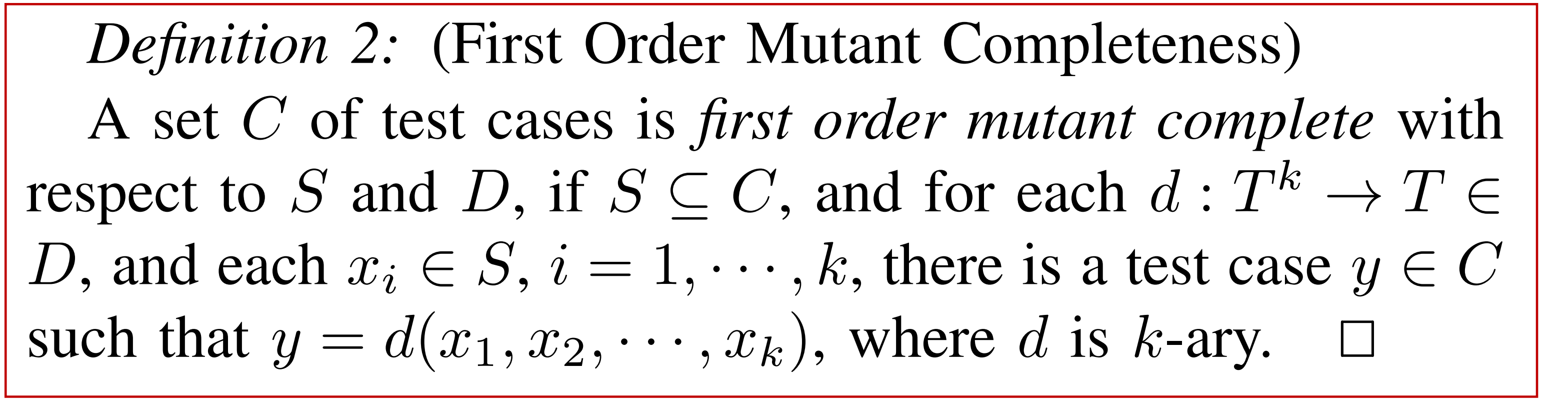


Let T be the set of all possible test cases for the software under test, $S \subset T$ ($S \neq \emptyset$) be a set of test cases, and $D \neq \emptyset$ be a set of datamorphisms and $d \in D$ be a datamorphism in D. We say that d is k-ary ($k > 0$), if $d : T^k \to T$.

Definition 1: (First Order Mutants)

A test case $y \in T$ is called a *first order mutant test case*, or simply a *first order mutant*, of S generated by D, if there is a k-ary datamorphism $d \in D$ and test cases $x_1, \cdots, x_k \in S$ such that $y = d(x_1, \cdots, x_k)$. □

Test Adequacy Criterion: First Order Mutant Completeness



Definition 2: (First Order Mutant Completeness)

A set C of test cases is *first order mutant complete* with respect to S and D, if $S \subseteq C$, and for each $d : T^k \to T \in D$, and each $x_i \in S$, $i = 1, \cdots, k$, there is a test case $y \in C$ such that $y = d(x_1, x_2, \cdots, x_k)$, where d is k-ary. □

- A test set is first order mutant complete means it contains all seed test cases and all first order mutants of the seed test cases
- Testing on a test set that is first order mutant complete means the testing covered all operation conditions, but not their combinations.

Algorithm 1: Generate 1st Order Complete Test Set

```
Input: S = the set of seed test cases;
          D = the set of datamorphisms;
Output: C = a set of test cases;
Variables: tempT = temporal set of test cases;
Begin
1:    C = EmptySet;
2:    for (each datamorphism d in D){
2.1:     tempT = EmptySet;
2.2:     Assume that d is a k-ary datamorphism;
2.3:     forall k-tuples (x1,... ,xk) of S {
           add d(x1,... ,xk) to tempT;
         };
2.4:     C = C + tempT;
      };
3:   return C + S;
End
```

Correctness of The Algorithm

Theorem 1 *The test set generated from S using D by Algorithm 1 is the minimal set of test cases that is first order mutant complete with respect to S and D.*

Proof.

(a) *Completeness*: Assume that the output test set C from Algorithm 1 is not complete. This means there is either a seed test case y not in C or there is a first order mutant y generated from seeds $x_1, \cdots, x_k \in S$ by using a k-ary datamorphism $d \in D$ is not in C. In the former case, it is in conflict with Step 3. In the latter case, it is in conflict with Step 2.3. Therefore, the assumption is incorrect.

(b) *Minimalness*: It is obvious to see that the output only contains seeds and first order mutants. □

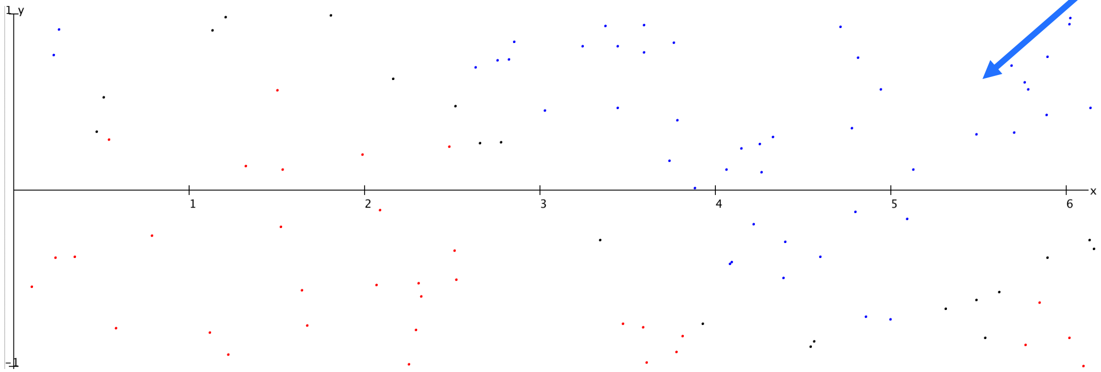
Example:

The application under test:

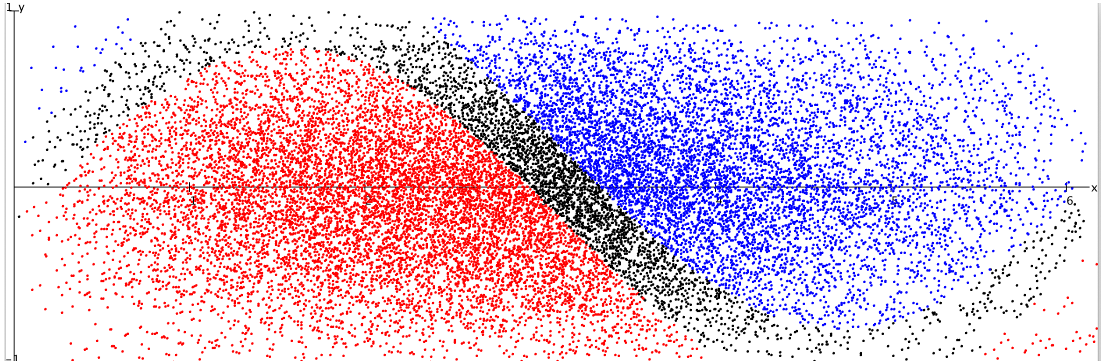
- classify points in a two dimensional space into tree types: red, blue and black.

The seed test set:

- 100 random points



(a) Original Test Set



(b) 1st Order Mutant Complete Test Set

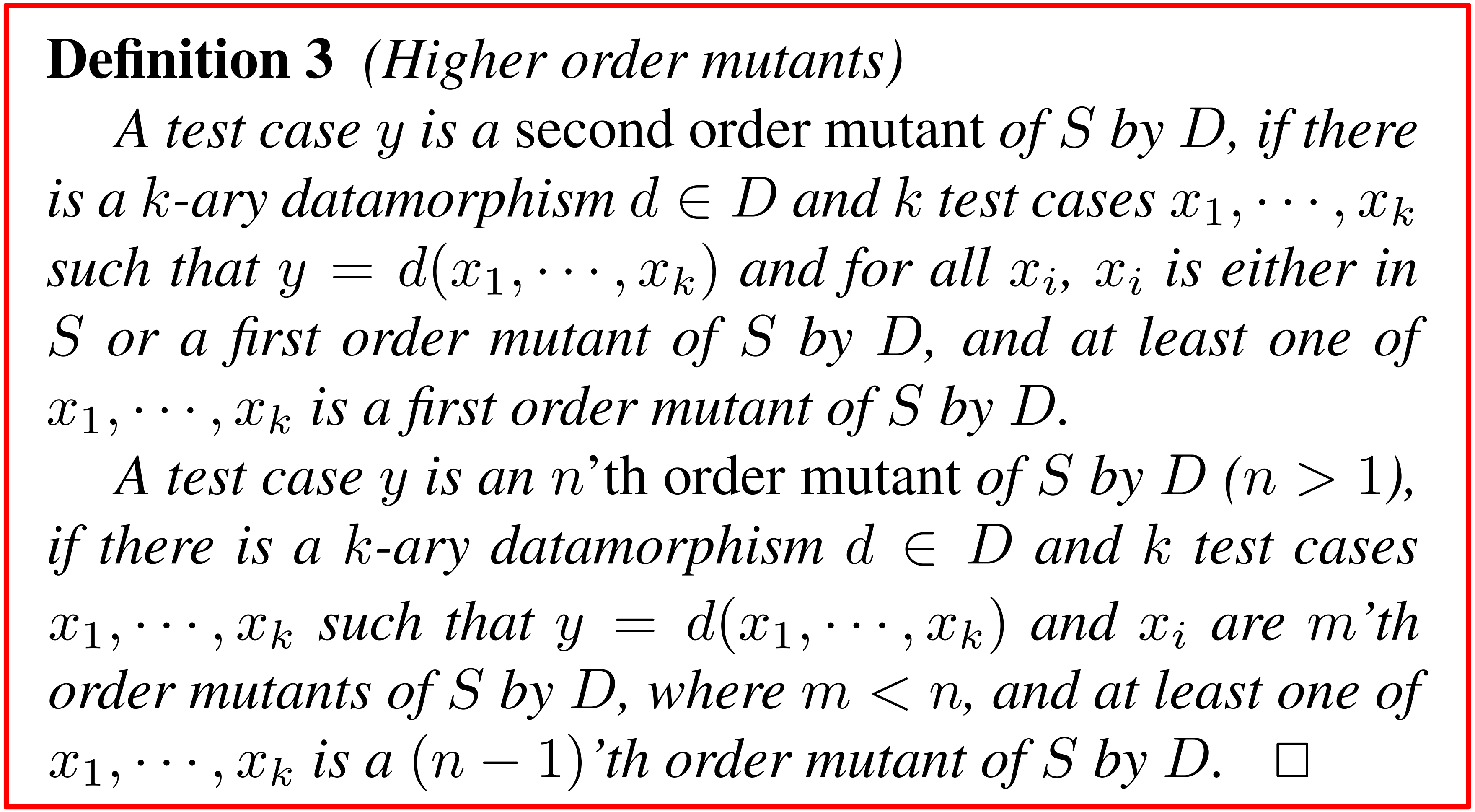
Datamorphism:

- Generates the midpoint of two test cases.

The test set generated:

- 10000 points as the 1st order mutant test cases
- 100 original test cases

High Order Mutants



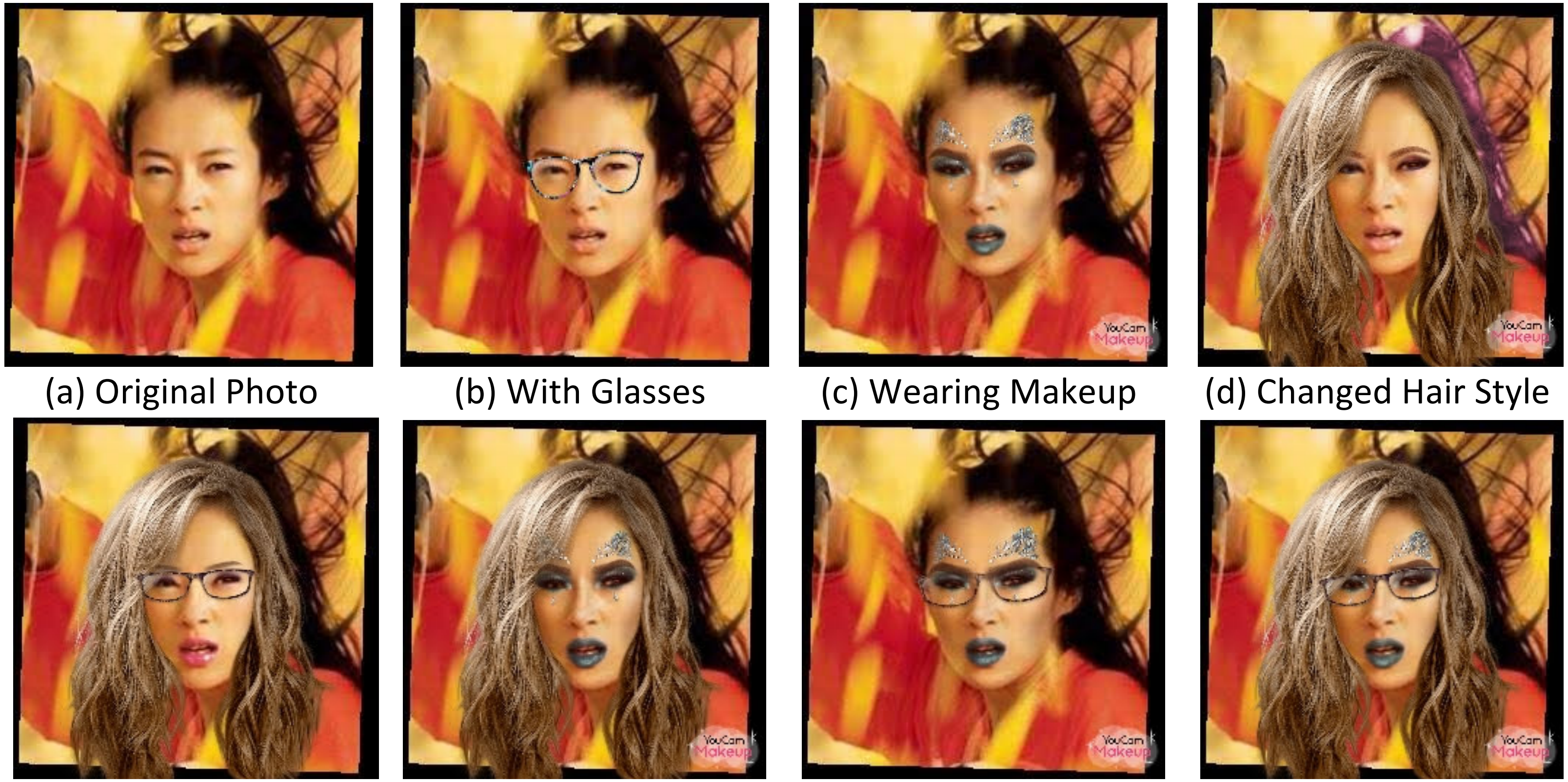
Definition 3 *(Higher order mutants)*

A test case y is a second order mutant *of S by D, if there is a k-ary datamorphism $d \in D$ and k test cases $x_1, \cdots, x_k$ such that $y = d(x_1, \cdots, x_k)$ and for all x_i, x_i is either in S or a first order mutant of S by D, and at least one of $x_1, \cdots, x_k$ is a first order mutant of S by D.*

A test case y is an n’th order mutant *of S by D ($n > 1$), if there is a k-ary datamorphism $d \in D$ and k test cases $x_1, \cdots, x_k$ such that $y = d(x_1, \cdots, x_k)$ and x_i are m’th order mutants of S by D, where $m < n$, and at least one of $x_1, \cdots, x_k$ is a $(n-1)$’th order mutant of S by D.* □

Examples of 1st Order and Higher Order Mutants

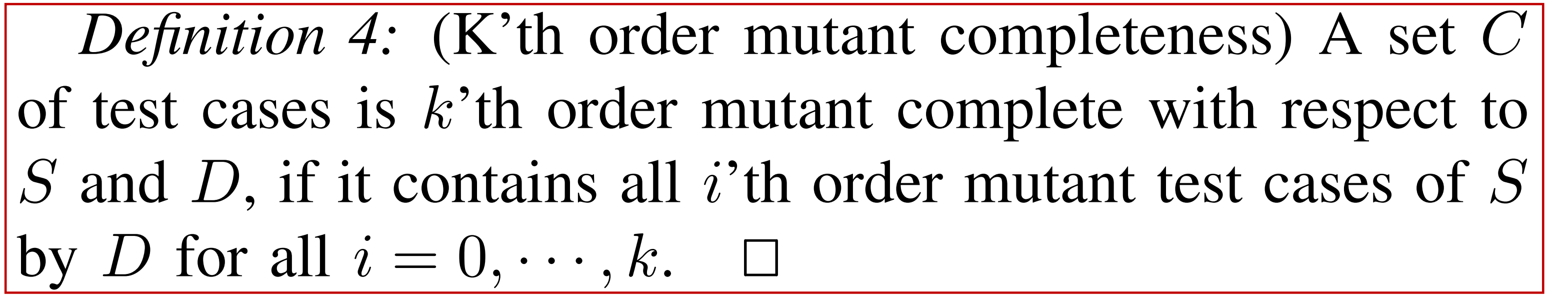
Higher order mutants are obtained by applying datamorphisms multiple times. They represent combinations of operation conditions.



(a) Original Photo (b) With Glasses (c) Wearing Makeup (d) Changed Hair Style

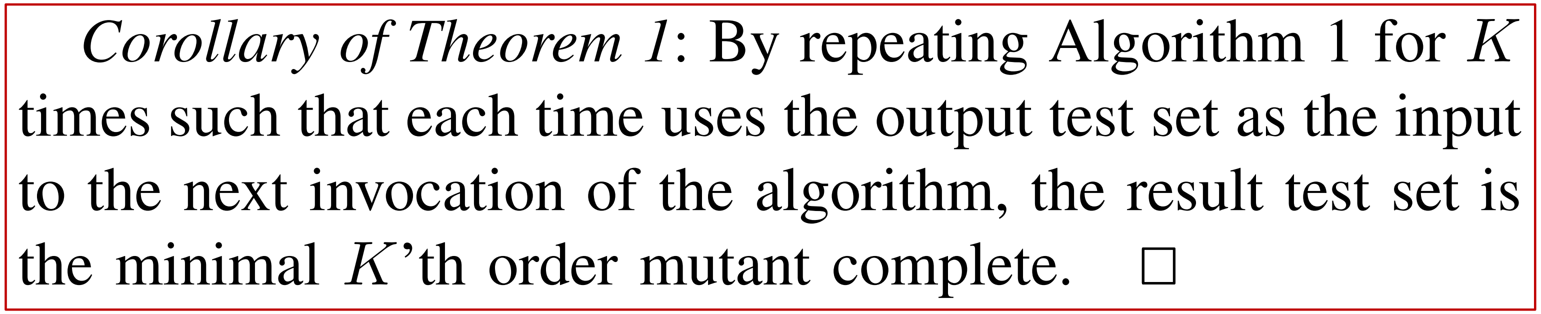
(e) b + d (f) c + d (g) b + c (h) b + c + d

Test Adequacy Criterion: K’th Order Mutant Completeness



Definition 4: (K’th order mutant completeness) A set C of test cases is k’th order mutant complete with respect to S and D, if it contains all i’th order mutant test cases of S by D for all $i = 0, \cdots, k$. □

- **Algorithm**: Generate K’th Order Mutant Complete Test Sets:
  - Call Algorithm 1 repeatedly for K times with the previous output as the input of the next call
- **Correctness of the Algorithm**:



Corollary of Theorem 1: By repeating Algorithm 1 for K times such that each time uses the output test set as the input to the next invocation of the algorithm, the result test set is the minimal K’th order mutant complete. □

Permutation Completeness and Exhaustive Test

Assume that the set *D* of datamorphisms contains *N* methods.

- Permutation complete test set:

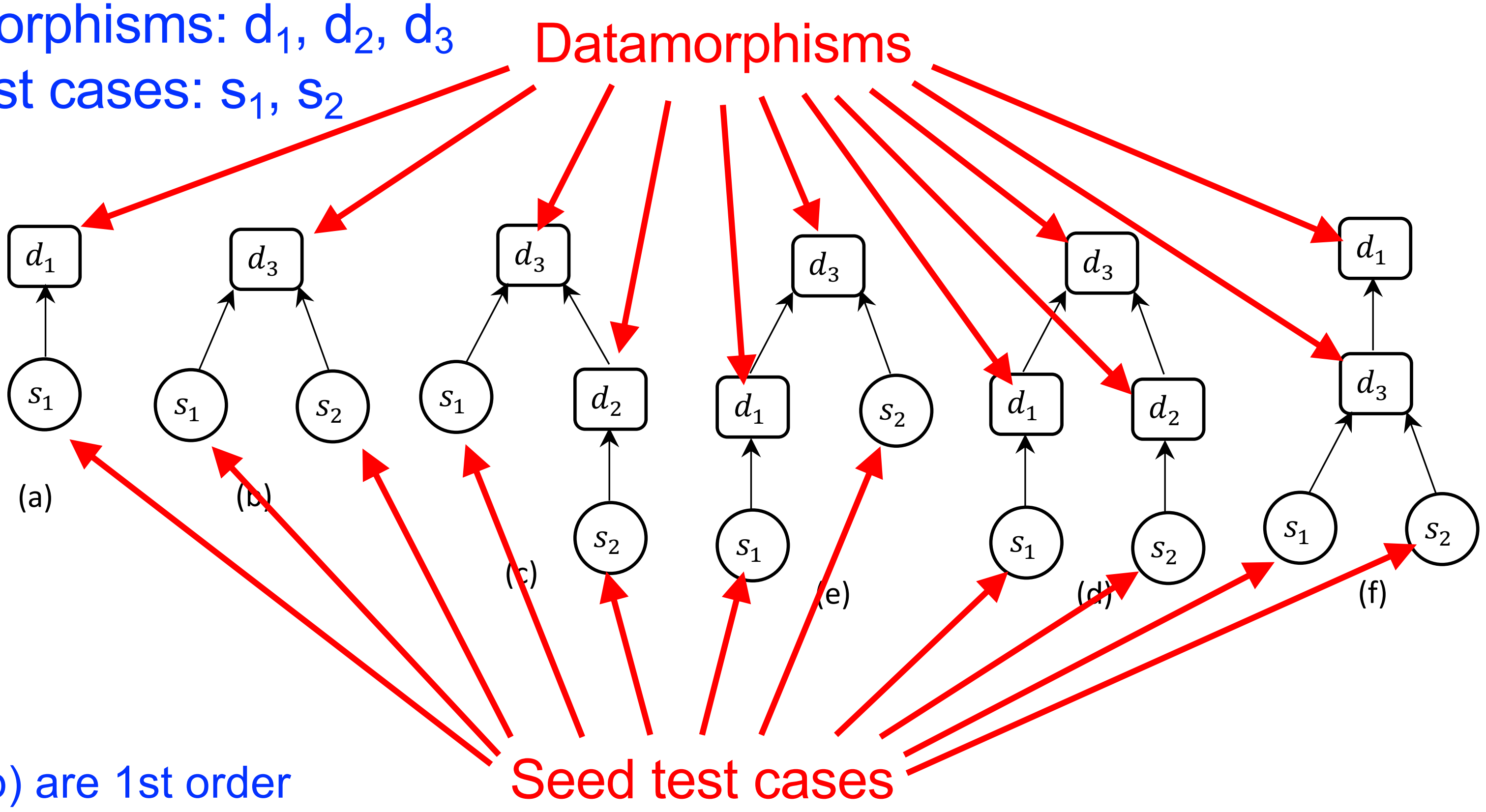
  If a test set is *N*'th order mutant complete with respect to *S* and *D*, it will contains all permutations of the datamorphisms applied to all test cases.

- Exhaustive test set:

  If the datamorphisms are *associative*, *commutative*, *distributive* and *idempotent*, a permutation complete test set contains all possible test cases that can be derived from a given set of test cases using the set of datamorphisms. The test set is therefore *exhaustive* with regard to the set of seeds and the datamorphisms.

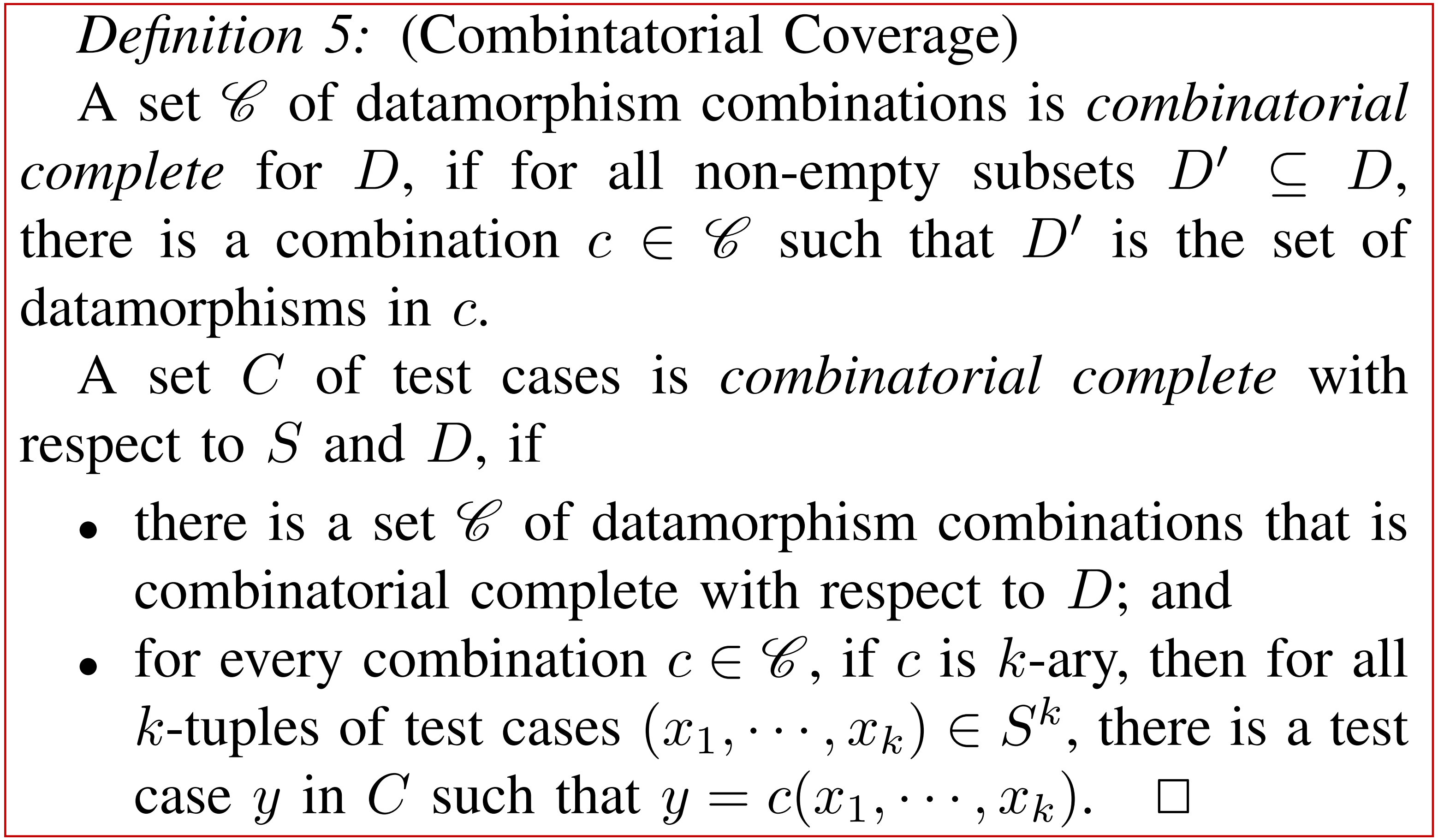
Combinations of Datamorphisms

- 3 datamorphisms: d_1, d_2, d_3
- 2 see test cases: s_1, s_2



- (a) and (b) are 1st order
- (c) - (f) are 2nd order
- There are more possible combinations

Test Adequacy Criterion: Combinatorial Coverage



Definition 5: (Combintatorial Coverage)

A set $\mathscr{C}$ of datamorphism combinations is *combinatorial complete* for D, if for all non-empty subsets $D' \subseteq D$, there is a combination $c \in \mathscr{C}$ such that D' is the set of datamorphisms in c.

A set C of test cases is *combinatorial complete* with respect to S and D, if

- there is a set $\mathscr{C}$ of datamorphism combinations that is combinatorial complete with respect to D; and
- for every combination $c \in \mathscr{C}$, if c is k-ary, then for all k-tuples of test cases $(x_1, \cdots, x_k) \in S^k$, there is a test case y in C such that $y = c(x_1, \cdots, x_k)$. □

Algorithm 2: Generate Combinatorial Complete Test Set

```
Input: S = the set of seed test cases;
          D = the set of datamorphisms;
Output: C = a set of test cases;
Variables: tempT = temporal set of test cases;
Begin
1:      for (each datamorphism d in D) {
1.1:      tempT = empty_set;
1.2:      Assume d is a k-ary, where k>0;
1.3:      for (all k-tuples (x1,...,xk) of S){
                add d(x1,...,xk) to tempT;
            };
1.4:      S = S + tempT;
         };
2:      return C + S;
End
```

Correctness of the Algorithm

Theorem 2 *The test set generated by Algorithm 2 is combinatorial complete with respect to S and D.*

Note:

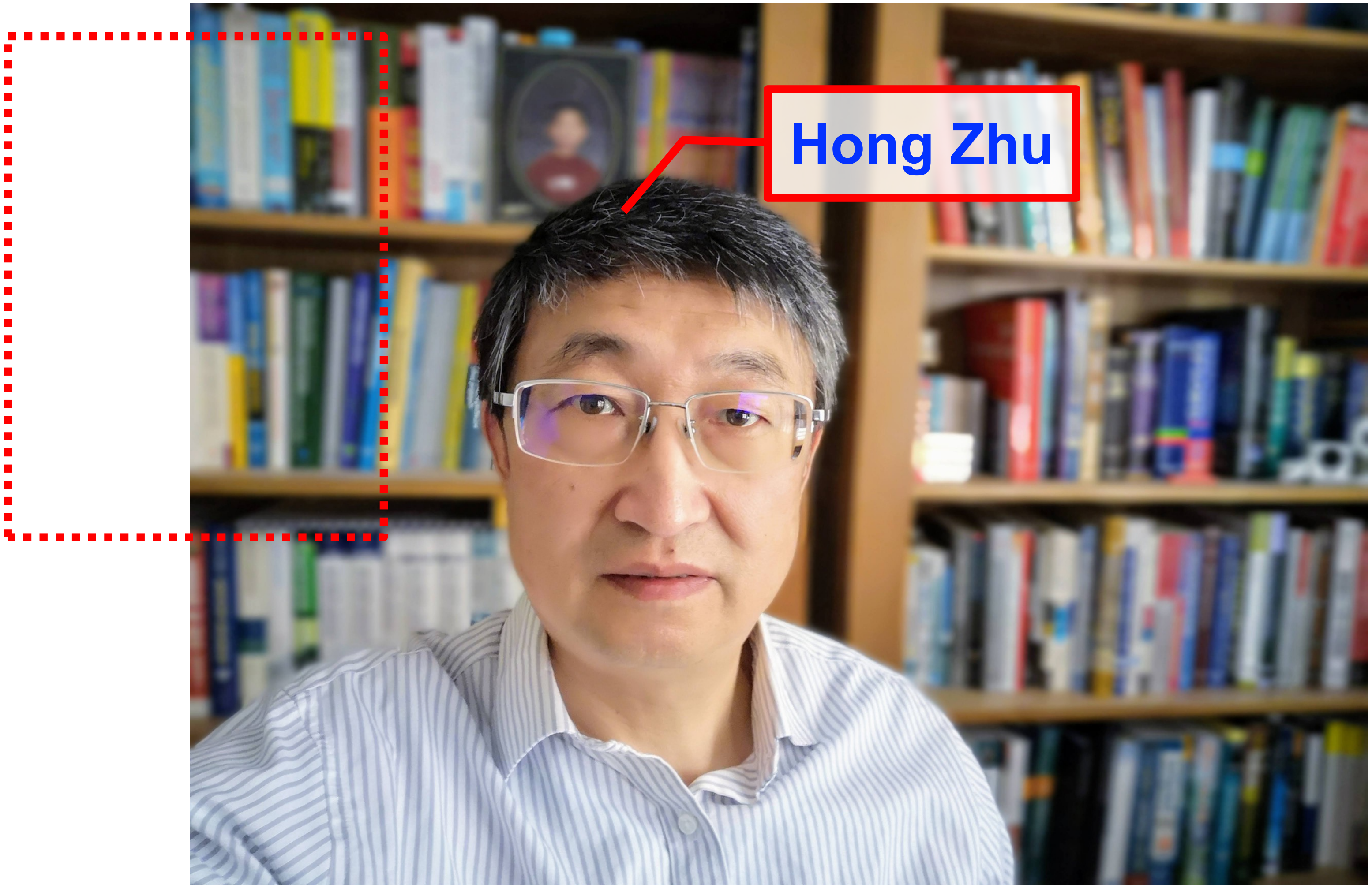
1. A combinatorial complete test set covers all combinations of the operation conditions represented by the datamorphisms.
2. The test set generated by the algorithm may be not minimal.
3. A proof of the theorem can be found in the following paper:

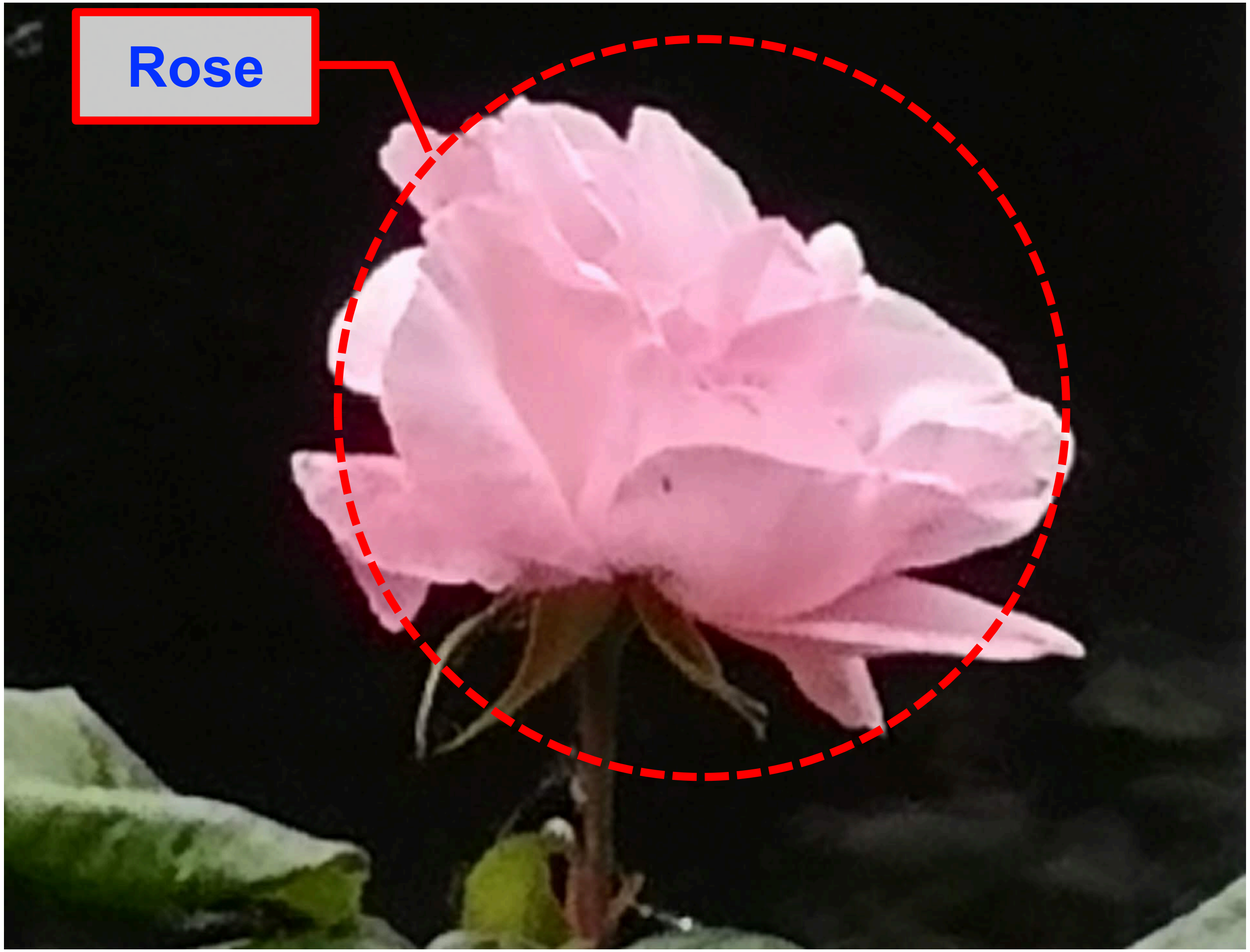
Hong Zhu, Ian Bayley, Dongmei Liu and Xiaoyu Zheng, ***Automation of Datamorphic Testing***, Proceedings of The Second IEEE International Conference on Artificial Intelligence Testing (AITest 2020), Aug. 3 - 6, 2020.

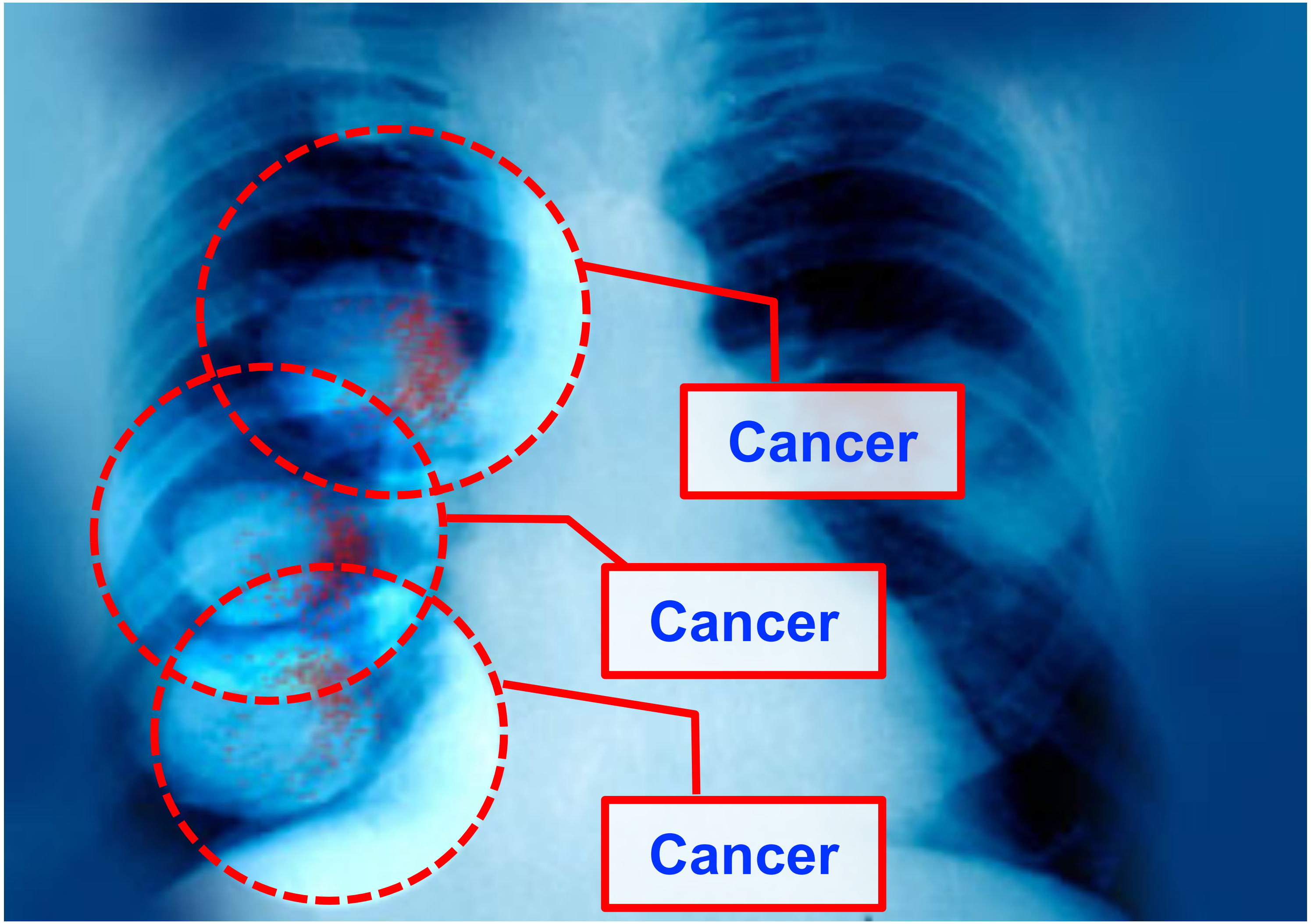
Exploratory Testing ML Classification Models

1. ML Classification Models
2. Exploratory testing (ET) methodology
3. Datamorphic approach to automate ET
   a. Test system and completeness
   b. Test strategies
4. Application to testing feature-based ML classifiers
5. The uses of the information discovered by ET

Typical Classification Applications




Rose


Cancer
Cancer
Cancer

AI Techniques to Develop Classifiers

- **Clustering**: (unsupervised learning)

  To find a way of partitioning data points into groups according to a similarity or a distance function

- **Classification**: (supervised learning)

  To find a function from a set of labelled data to classify the data into groups such that data of the same label are in the same class

Classifiers

A ***classifier*** (or a ***classification program***) is a mapping $P\colon D \rightarrow G$ from the data space D into a non-empty set of groups $G = \{l_1, \ldots, l_n\}$ *(also called classes)* such that $D = \bigcup_{l \in G} D_l$, where $D_l = \{x \in D \mid P(x) = l\}$, and $\forall x, y \in G. \left(x \neq y \Longrightarrow D_x \cap D_y = \emptyset\right)$.

We assume that there is a distance function $\|.,.\|\colon D^2 \rightarrow R^+$ *, such that*

$$\forall x \in D.(\|x, x\| = 0)$$

$$\forall x,y \in D.(\|x, y\| \geq 0)$$

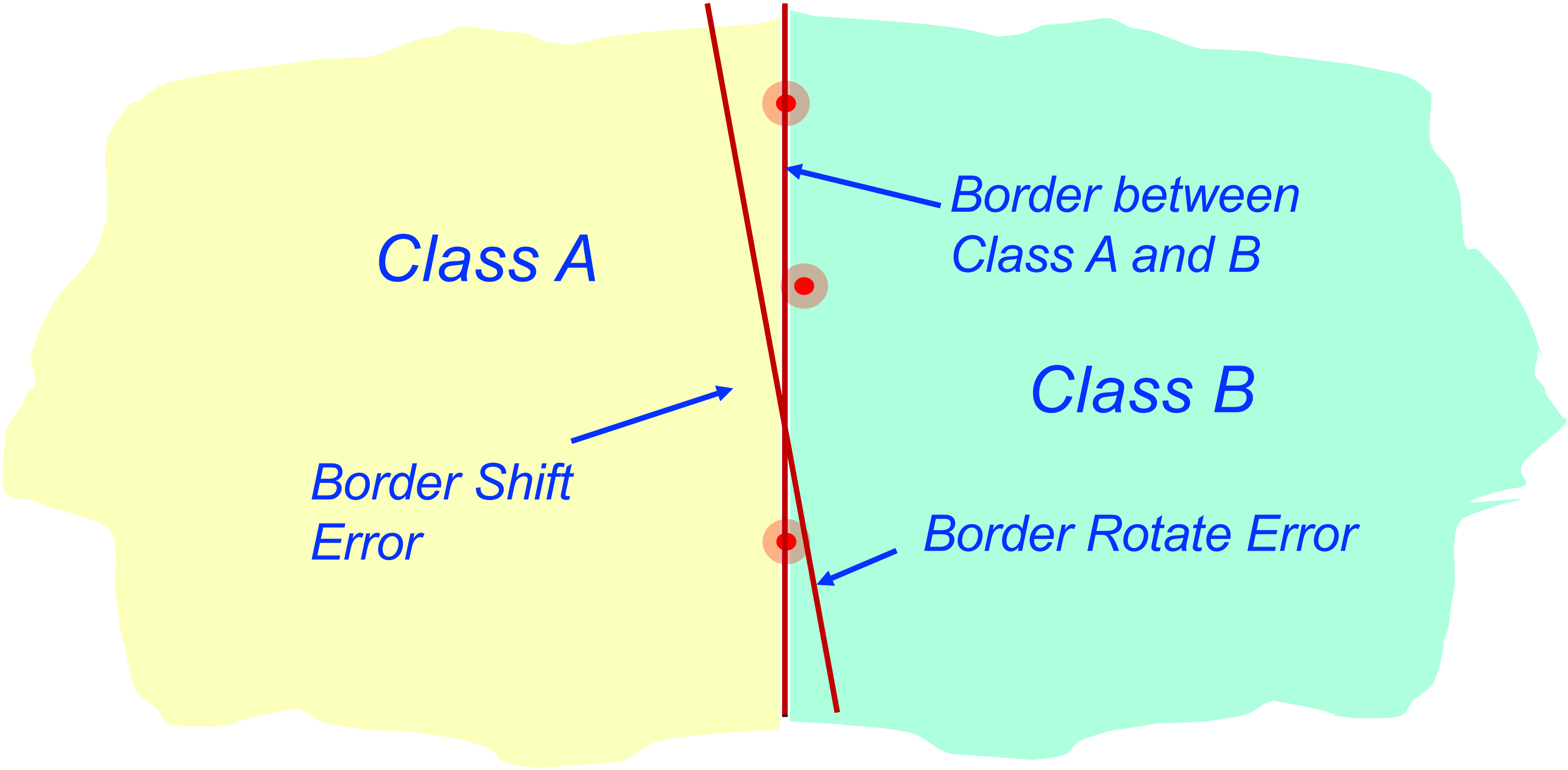
$$\forall x,y \in D.(\|x, y\| = \|y, x\|)$$

$$\forall x,y,z \in D.(\|x, y\| + \|y, z\| \geq \|x, z\|)$$

Testing Classification Systems

- Traditional Approach: Category Partitioning Testing (also known as *domain analysis*)
  - **Focusing on the borders** between different classes,
    - Defined by the specification, or
    - As implemented by the code, or
    - A combination of the above
  - **Technique**:
    - For each class: selecting test cases on the borders and near-by to the borders
    - The number of test cases on or nearby to a border depends on the dimension of the data space
  - **Theory** (e.g. in the perturbation testing theory):
    - Test cases on the border and near-by to the borders can ensure no linear transformations of the border (e.g. border shift errors and rotate errors) under certain conditions on the border and data space.

Category Partitioning Test (Domain Analysis)



Class A

Border between Class A and B

Class B

Border Shift Error

Border Rotate Error

Can we borrow the Ideas of category partitioning test to ML?

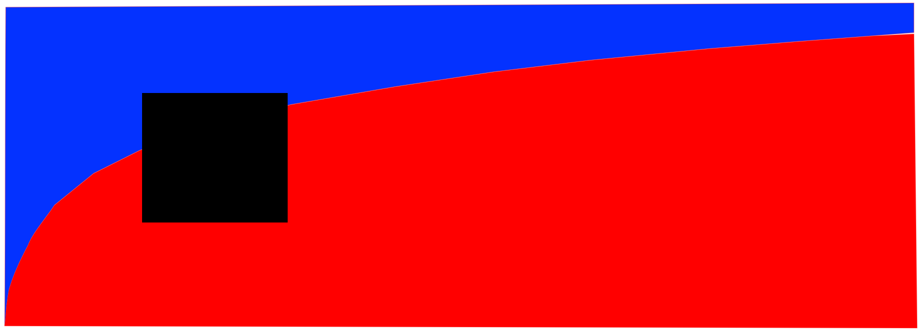
Problems to Apply Partitioning Test to AI Applications

- The borders between classes are often unknown
  - No definition of the required border in the specification
  - Not easy to get the border as implemented by the ML model
- The data space and the borders are highly complicated
  - High dimensional
  - Non-numerical data
- The theory of domain analysis does not apply
  - The common errors in the application of AI technology may be not linear transformations (not border shift or rotate errors)

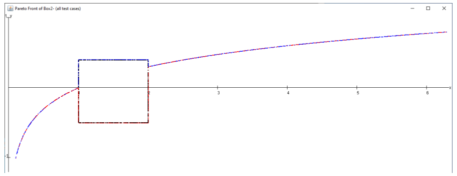
Examples of Errors in Machine Learning Models

The classifier:

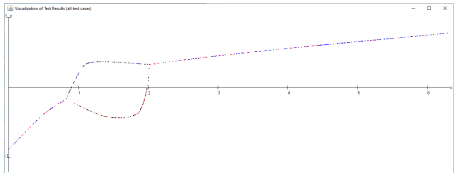
$$[0,2\pi]\times[-1,1] \rightarrow \{red, blue, black\}$$



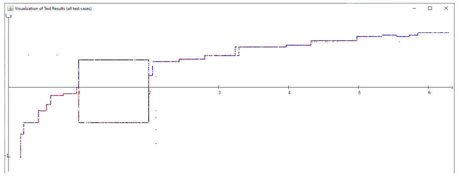
- Take 5000 random samples of the original classifier
- Apply various ML techniques to train ML models
- The result models are shown on the right:



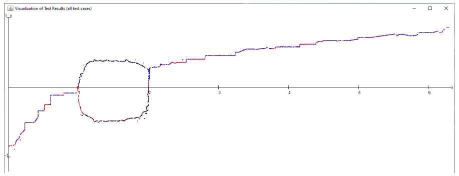
Original Coded Classifier



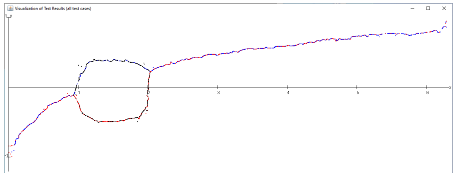
Deep Neural Network (DNN)



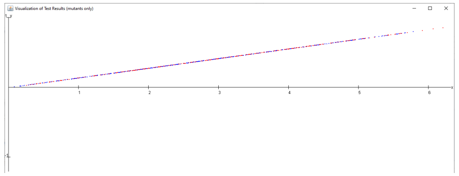
Decision Tree (DT)



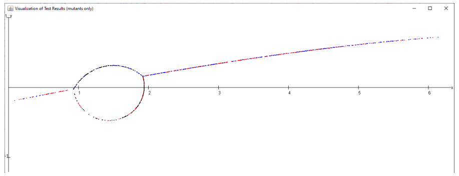
Hard Voting of LR, KNN and DT (HV)



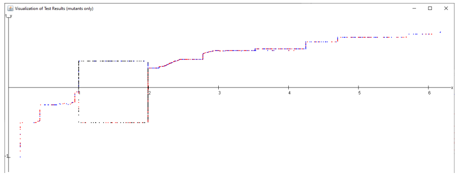
K-Nearest Neighbour (KNN)



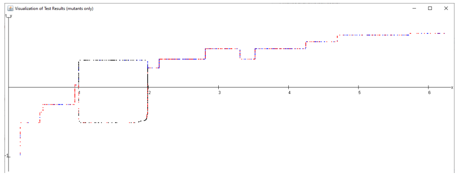
Logistic Regression (LR)



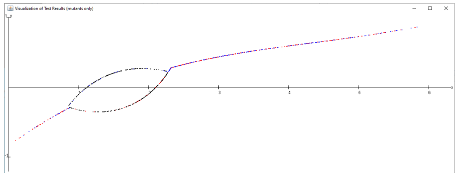
Naïve Bayes (NB)



Stacking KNN over LR, DT and HV (Stack)



Soft Voting of LR, KNN and DT (SV)



Supporting Vector Machine (SVM)

Exploratory Testing

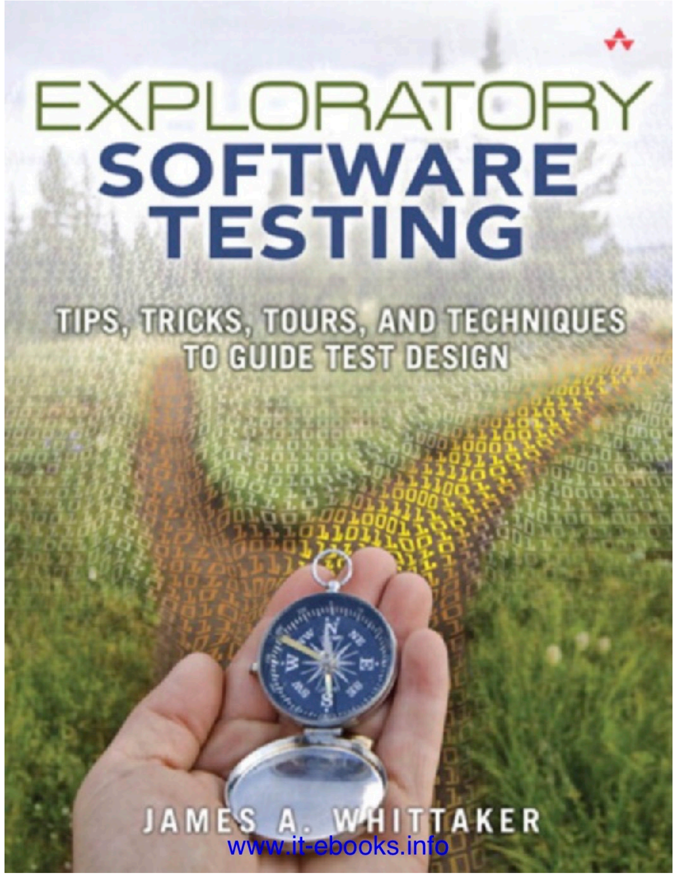
OXFORD BROOKES UNIVERSITY

"In exploratory testing, the *tester* interacts with the application and uses the information that the application provides to change the course of testing in order to explore the application's functionality."

[Whittaker, 2009]

"Simultaneously designing and executing tests to learn about the system, using your insights from the last experiment to inform the next."

[Hendrickson, 2013]


EXPLORATORY SOFTWARE TESTING
TIPS, TRICKS, TOURS, AND TECHNIQUES TO GUIDE TEST DESIGN
JAMES A. WHITTAKER
www.it-ebooks.info

Exploratory vs Confirmatory Testing

Confirmatory Testing

- ***Goal of Test***:
  - Confirming or disproving the correctness with respect to a given specification
  - Testing for verification and validation w.r.t. known requirements and specification
- ***Software under test***:
  - As an entity with clear definition and specification
  - Knowledge of the SUT is essential to perform testing
- ***Test cases:***
  - Pre-scripted
  - Independent from each other
  - Quality criteria: to coverage all possibilities

Exploratory Testing

- ***Goal of Test***:
  - Discovering the functions and properties of the software
  - Testing as experiments on the software
  - To search for useful information
- ***Software under test***:
  - As an entity unknown
  - No knowledge of the SUT is assumed
- ***Test cases:***
  - Generated or selected on the fly: using the result of the previous tests to guide the choice of the next
  - *Quality criteria*: to maximise its effectiveness in the process of searching for useful information

Exploratory Testing: A Brief Review

- A primitive form in the practice of ***manual testing*** existed for a long time
- Most suitable for situations where ***specification is not available*** or not well defined
- Relatively recently identified by researchers to provide guidance to improve the effectiveness of manual testing of interactive software
  - Kane [1988] coined the term “exploratory test”
  - Whittaker [2009] recognised a defined (informally) strategies for GUI based testing
  - Many researchers conducted empirical studies of the factors that effect ET

- Kaner, C., 1988. **Testing Computer Software**. John Wiley and Sons.
- Whittaker, J. A. 2009. **Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design**. Pearson Education.

Exploratory Testing of Classifiers

- Goal:
  - ✓ To discover the borders between classes as defined by the ML model under test
    - Borders are critical to understand the behaviour of a ML model
    - Values on borders are critical test cases for a ML model
- Problems:
  - How to represent borders?
  - Can borders be discovered?
  - If yes, how to discover borders?
  - Is the discovery of borders cost efficient? Can it be automated?
  - How to use borders?

Pareto Front: Representation of Borders

Definition 1. (Pareto Front of Classification)

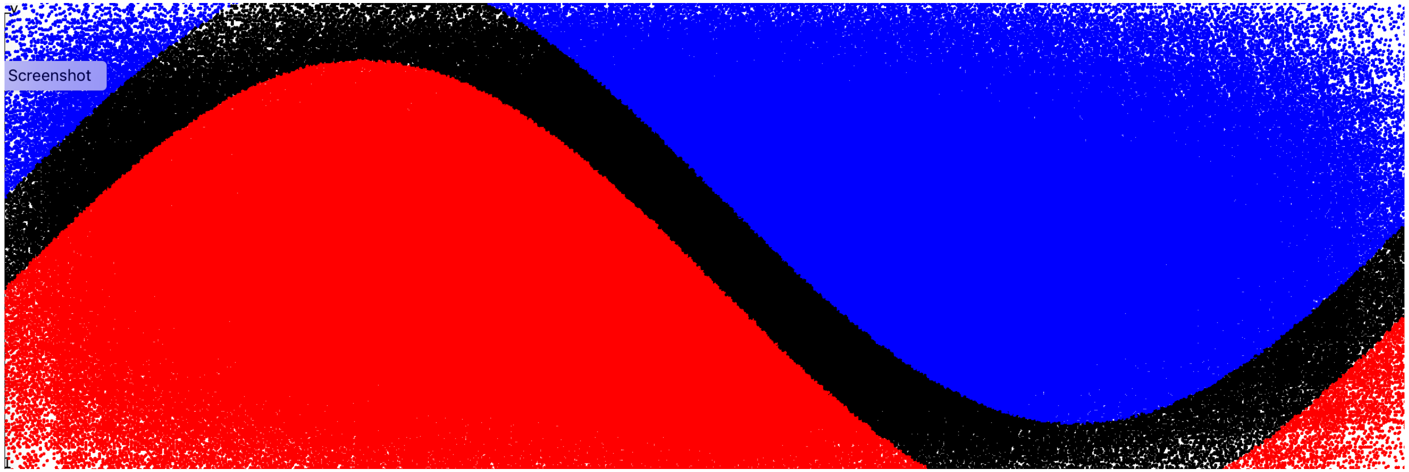
Let

- $P : D \to G = \{l_1, \dots, l_n\}$ $(n > 0)$ be a classifier,
- $\|\cdot,\cdot\| : D \times D \to \mathrm{R}^+$ be a distance metric on D, and
- $\delta > 0$ be any given real number.

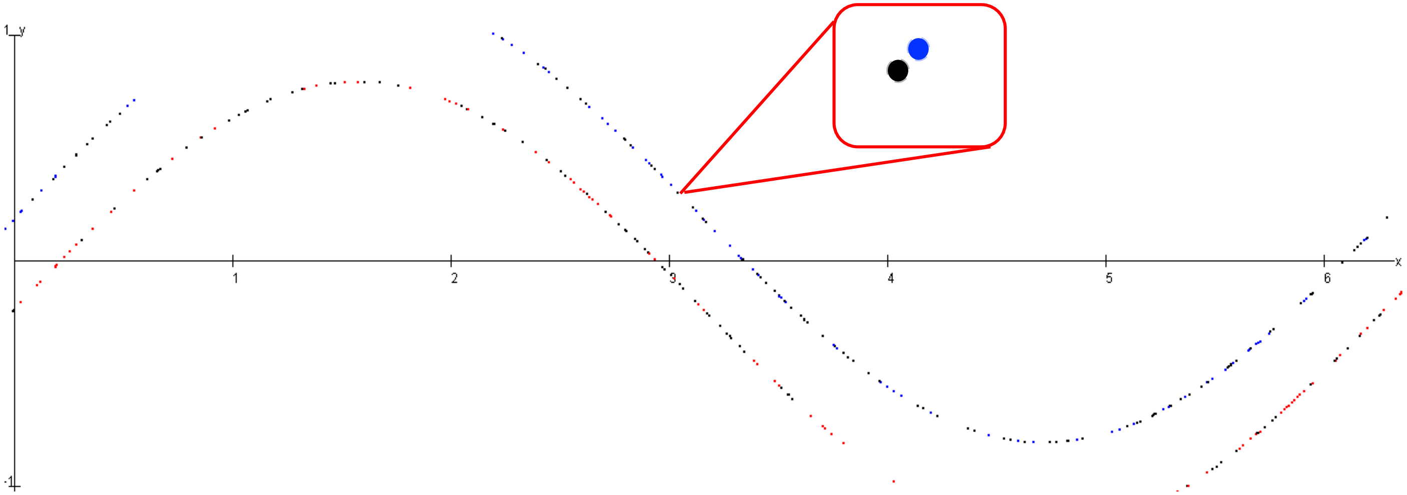
A set $\{\langle a_i, b_i\rangle | a_i, b_i \in D, i = 1, \dots, k\}$ $(k > 0)$ of data pairs is a ***Pareto front*** of the classes according to P with respect to $\|\cdot, \cdot\|$ and δ, if for all $i = 1,\cdots,k$, $P(a_i) \neq P(b_i)$ and $\|a_i, b_i\| \leq \delta$.

Example: Pareto front

Classifier:



A Pareto front :



Essential Elements of Exploratory Testing

OXFORD BROOKES UNIVERSITY

✓Designing:

It is concerned with identifying interesting things to vary and interesting ways in which to vary them so that the experiment can be better performed.

✓Executing:

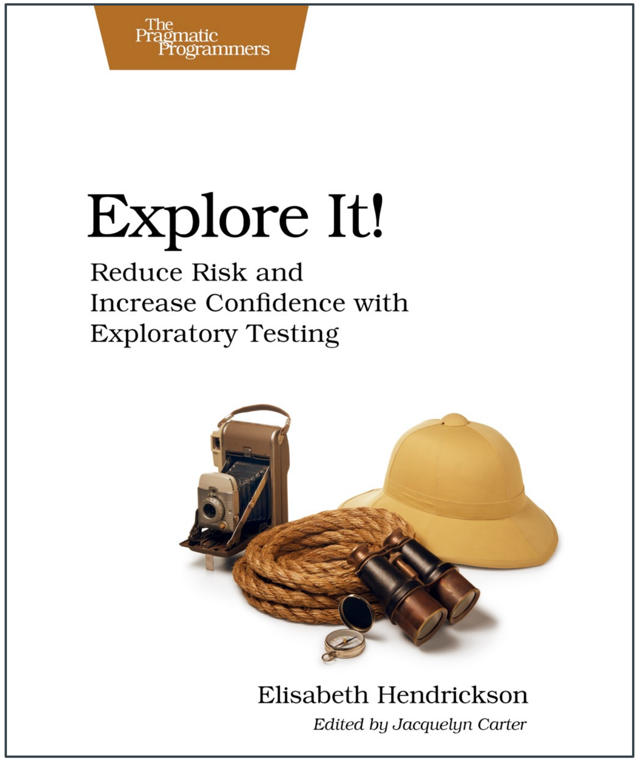
A test case is executed immediately when it is designed.

✓Learning:

The testers "*discover how the software operates*".

✓Steering:

Using the insights gained from the previous test execution(s) to inform the next.



The Pragmatic Programmers

Explore It!

Reduce Risk and Increase Confidence with Exploratory Testing

Elisabeth Hendrickson

Edited by Jacquelyn Carter

Datamorphic Approach to Exploratory Test

Essential Elements of ET	Datamorphic Approach to ET (**)
Design: Identifying interesting things to vary and interesting ways in which to vary them	Developing test morphisms to implement the ways in which to vary the test entities
Executing: Executing a test as soon as you think of a test	Invoking the test executor on test cases
Learning: Discovering how the software operates	Writing test code to analyse test results and present them in a format easy to digest by human beings
Steering: Using knowledge gain from testing to suggest the next test with focus on most important information to discover	Formalising steering strategies in the form of algorithms that utilise test entities and morphisms as parameters

(*) Elisabeth Hendrickson, 2013. ***Explore IT! Reduce Risk and Increase Confidence with Exploratory Testing***. The Pragmatic Bookshelf, Lighting Source UK Ltd.

(**) Hong Zhu and Ian Bayley, 2022. ***Discovering boundary values of feature-based machine learning classifiers through exploratory datamorphic testing***. Journal of Systems and Software, Vol. 187.

Exploratory Test System $\mathscr{T} = \langle \mathscr{E}, \mathscr{M} \rangle$

(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.*

Exploratory Test Systems

An ***exploratory test system*** is a test system $T = \langle E, M \rangle$ that *M* has contains the following test morphisms.

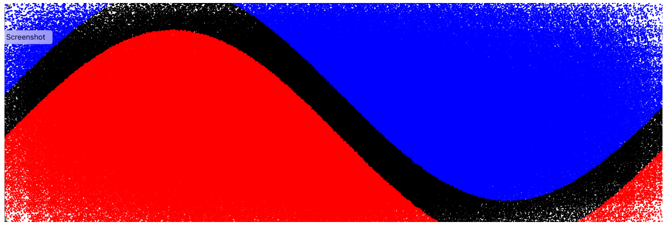
- A test executer $Exe_P(x)$: through the test morphism the program *P* under test are invoked on a test case *x* and receives the output of *P*. That is, $Exe_P(x) = P(x)$.
- A set of traversal methods: a set $W \subseteq M$ of *unary datamorphisms* defined on *D*.
  For each $w \in W$ and $x \in D,$ by repeatedly invoke the datamorphism $w, i.e.\ w(x), w^2(x), \cdots, w^n(x)$, we can generate a sequence of data points in *D*, where $w^1(x) = w(x), w^{n+1}(x) = w(w^n(x))$.
- A midpoint method: a *binary datamorphism* $m \in M$ such that

$$\forall x, y \in D. (\| x, y \| > \delta_m \Rightarrow \| x, z \| < \| x, y \| \wedge \| y, z \| < \| x, y \|)$$

where $z = m(x, y)$, $\delta_m = Min_{x \neq y \in D} \{\| x, y \|\}$.

Example: An Exploratory Test System

- The classifier under test:
  - **Input data space** D: $[0,2\pi] \times [-1,1]$
  - **Function**: classify into ***red***, ***blue*** and ***black***

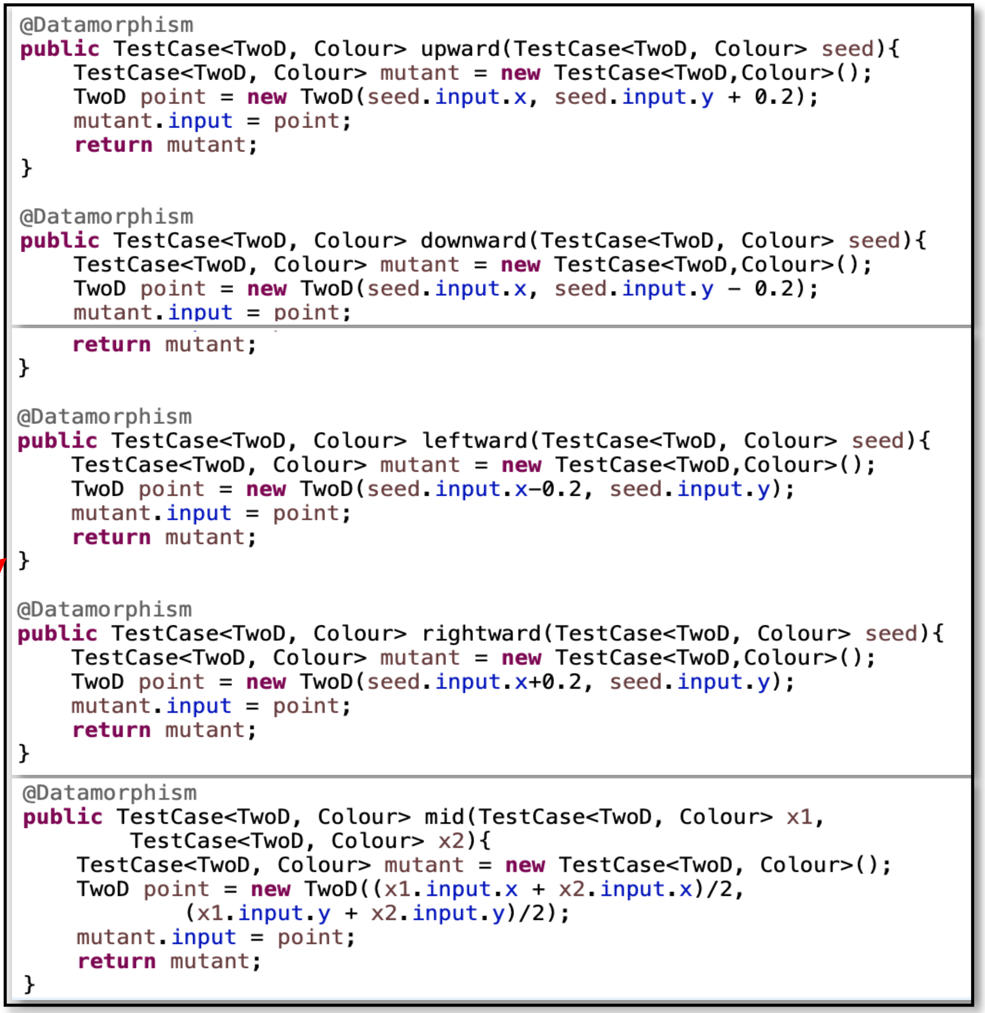


- The distance metrics:

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

- The datamorphisms:
  - upward(x);
  - downward(x);
  - leftward(x);
  - rightward(x);
  - mid(x, y);

Implementation of datamorphisms in Java



```java
@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;
}
```

Completeness of Exploratory Test System

Definition 5. (***Completeness***)

An exploratory test system $T = \langle E, M \rangle$ on data space D is ***complete***, if forall $a, b \in D$, there is a composition $\varphi(x)$ of datamorphisms in M such that $b = \varphi(a)$.

An exploratory test system T is ***approximately complete***, if for all $a, b \in D$ and every $\delta > \delta_m$, there is a composition $\varphi(x)$ of datamorphisms in M such that $\| b, \varphi(a) \| \leq \delta$.

The completeness of an exploratory test system ensures that there will be **no blind spot** in the data space that cannot be explored.

Exploratory Test Systems for Feature-Based Classifiers

Question:

Is there complete exploratory test system for ML classifiers?

Answer:

Yes, for feature-based classifiers, we can always construct a complete exploratory test system.

Feature-Based Classifiers

Definition 2. (***Feature Based Classifier***)

Let $P : D \rightarrow G$ be a classification program. We say that P is a ***feature-based classifier*** if there is a natural number $K \geq 1$ such that $D = D_1 \times \cdots \times D_K$, where for every $i = 1, \cdots, K$, D_i is the set of values of a feature f_i.

Types of features:

- A feature f_i is categorical, if D_i is a finite non-empty set.
- A feature f_i is discrete numerical, if D_i is the set of integer values or natural numbers.
- A feature f_i is continuous numerical, if D_i is the set of real numbers, or a non-empty interval of real numbers.

Datamorphisms for Continuous Numerical Features

- Two unary datamorphisms for each feature f_i as the traversal methods

$$up_i(\langle x_1, \cdots, x_K \rangle) = \langle x_1, \cdots, x_i + c_i, \cdots x_K \rangle$$

$$down_i(\langle x_1, \cdots, x_K \rangle) = \langle x_1, \cdots, x_i - c_i, \cdots x_K \rangle$$

where $c_i > 0$ is a given real value.

- A binary datamorphism $mid_E(x, y)$ as the midpoint method.

$$mid_E(\langle x_1, \cdots, x_K \rangle, \langle y_1, \cdots, y_k \rangle) = \left\langle \frac{x_1 + y_1}{2}, \cdots, \frac{x_K + y_K}{2} \right\rangle$$

- The Euclidean distance on multi-dimensional real numbers.

$$\|\langle x_1, \cdots, x_K \rangle, \langle y_1, \cdots, y_k \rangle\| = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$$

There are many other ways to define distance metrics on real numbers.

Datamorphisms for Discrete Numerical Features

- Two unary datamorphisms for each discrete numerical feature f_i as the traversal methods

$$up_i(\langle x_1 ,\cdots, x_K \rangle) = \langle x_1,\cdots, x'_i, \ldots , x_K \rangle, \text{ where } x'_i = x_i + 1.$$

$$downi(\langle x_1,\cdots, x_K\rangle) = \langle x_1,\cdots, x'_i, \ldots , x_K \rangle$$

where $x'_i = x_i - 1$, if D_i is the set of integers; otherwise

$$x'_i = \begin{cases} x_i - 1, & if\ x_i > 0 \\ 0, & otherwise \end{cases}$$

- The midpoint datamorphism $mid_N\ (x, y)$ is defined as follows.

$$mid_N(\langle x_1,\cdots, x_K\rangle, \langle y_1,\cdots, y_K\rangle) = \left\langle \left\lfloor \frac{|x_1 - y_y|}{2} \right\rfloor, \cdots, \left\lfloor \frac{|x_K - y_K|}{2} \right\rfloor \right\rangle$$

- The distance metric $\| \langle x_1,\cdots, x_K\rangle, \langle y_1,\cdots, y_K\rangle \|_N = \sum_{i=1}^{K} |y_i - x_i|$

Datamorphisms for Categorical Features

- Two unary datamorphisms as the traversal methods for each categorical feature f_i

$$up_i(\langle x_1, \cdots, x_K \rangle) = \langle x_1, \cdots, x'_i, \cdots, x_K \rangle, \text{ where } x'_i = \begin{cases} v_{i,j+1} & \text{if } x_i = v_{i,j} \text{ and } j < n_i \\ v_{i,n_i} & \text{if } x_i = v_{i,n_i} \end{cases}$$

$$down_i(\langle x_i, \cdots, x_K \rangle) = \langle x_i, \cdots, x'_i, \cdots, x_K \rangle, \text{where } x'_i = \begin{cases} v_{i,j-1} & \text{if } x'_i = v_{i,j} \text{ and } j > 1 \\ v_{i,1} & \text{if } x'_i = v_{i,1} \end{cases}$$

- A binary datamorphism $mid_D(x, y)$ as the midpoint method

$$mid_D(x, y) = \langle z_1, \cdots, z_K \rangle, \qquad z_i = \begin{cases} x_i & \text{if } x_i = y_i \\ x_i & \text{if } x_i \neq y_i \text{ and } x_i \text{ is an odd-indexed element in } \Delta(x, y) \\ y_i & \text{if } x_i \neq y_i \text{ and } x_i \text{ is an even-indexed element in } \Delta(x, y) \end{cases}$$

- The distance between x and y, written $\| x, y \|_D$, is defined as the number of elements in x and y that are different.

Exploratory Test System for Feature-based Classifiers

Let $x = \langle d_1, \cdots, d_u, n_1, \cdots, n_v, r_1, \cdots, r_w \rangle \in D$.

$$x_D = \langle d_1, \cdots, d_u \rangle,\quad x_N = \langle n_1, \cdots, n_v \rangle, \text{ and } x_E = \langle r_1, \cdots, r_w \rangle.$$

Define $\oplus$ such that $x = x_D \oplus x_N \oplus x_E$.

Discrete numeric features

Discrete non-numeric features

Continuous numeric features

- Two unary datamorphisms up_i and $down_i$ for each feature f_i
  - Definition of the datamorphisms depends on the type of feature; see previous slides
- A binary datamorphism $mid_H(x, x')$ as the midpoint method

$$mid_H(x, x') = mid_D(x_D, x'_D) \oplus mid_N(x_N, x'_N) \oplus mid_E(x_E, x'_E)$$

- The distance function $\|\cdot,\cdot\|_H : D \times D \to R^+$ as follows.

$$\| x, x' \|_H = \| x_D, x'_D \|_D + \| x_N, x'_N \|_N + \| x_E, x'_E \|_E.$$

Theorem. The above set of datamorphisms and the distance metrics $\|\cdot,\cdot\|_N$ together satisfy the requirements of exploratory test systems on datamorphisms, and it is approximately complete.

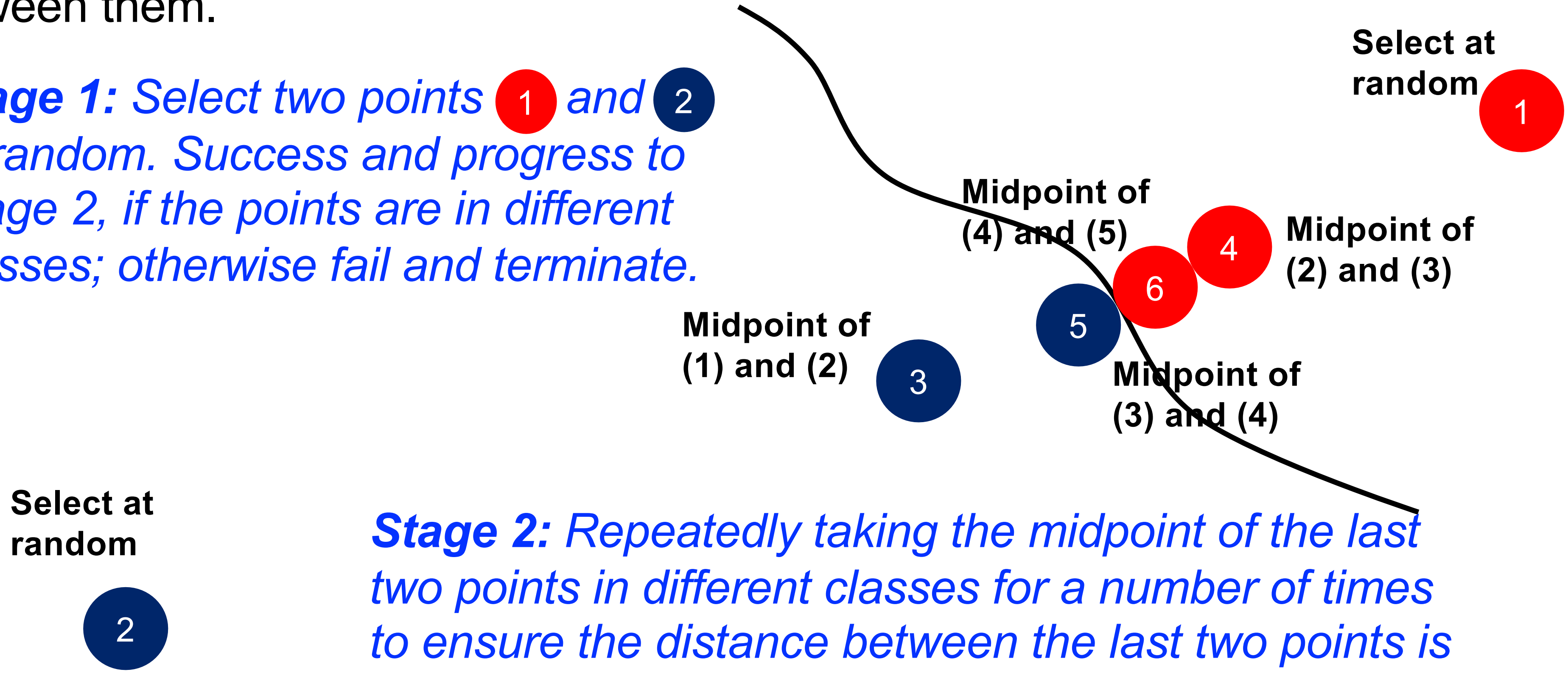
Exploratory Test Strategies

1. **Definitions of the strategies as algorithms**
   a) Random target
   b) Directed walk
   c) Random walk
2. **Proofs of the correctness of the algorithms**
3. **Evaluation of performance of the algorithms**

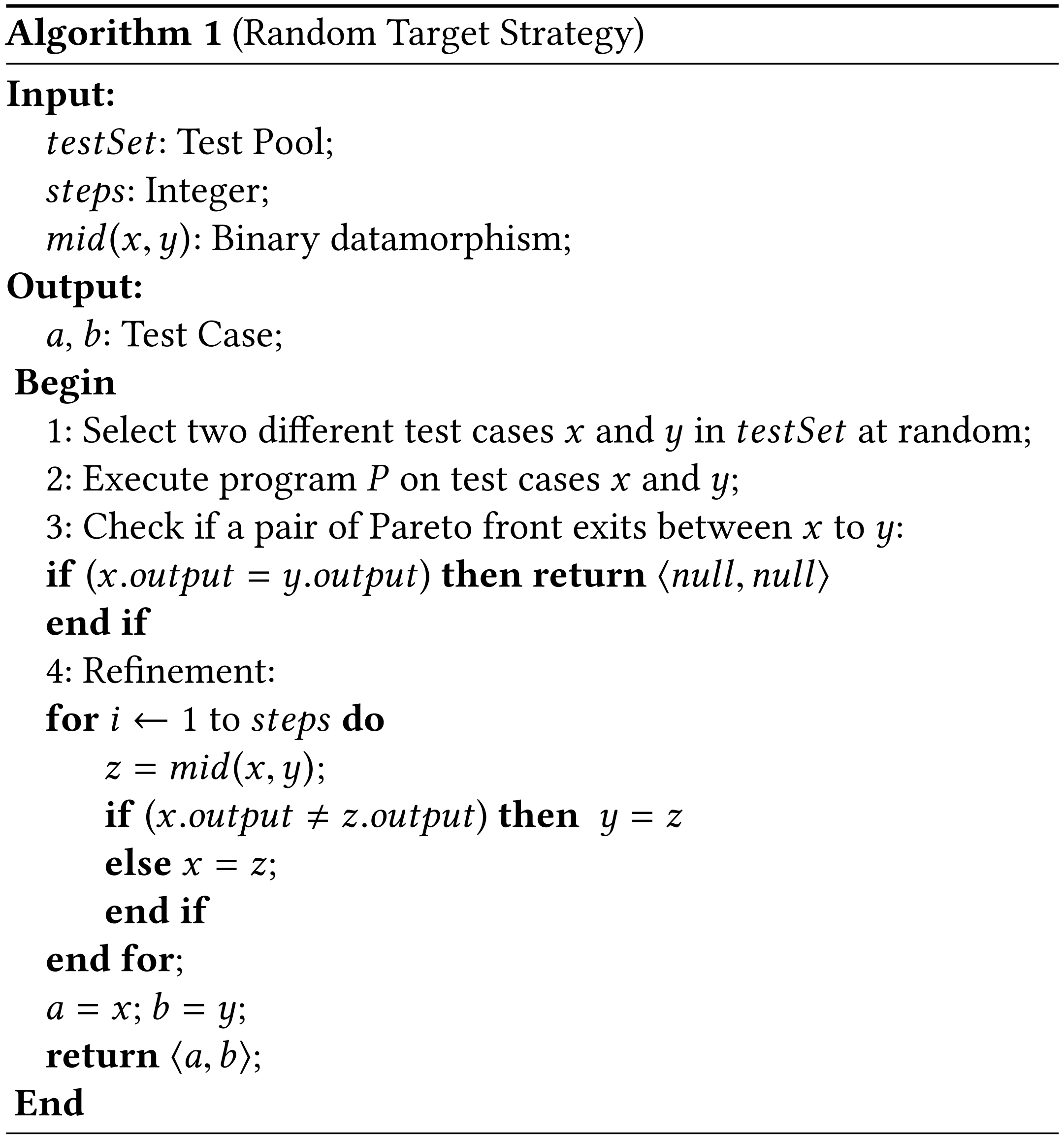
Strategy 1: Random Target

Select a number of pairs of points in the space D at random, if a pair of points are in different class, using the midpoint method repeatedly to find a pair border points between them.

Stage 1: *Select two points* ● 1 *and* ● 2 *at random. Success and progress to Stage 2, if the points are in different classes; otherwise fail and terminate.*



Stage 2: *Repeatedly taking the midpoint of the last two points in different classes for a number of times to ensure the distance between the last two points is smaller than the target distance of the pareto*



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 $(x.output \neq z.output)$ then $y = z$
     else $x = z$;
     end if
  end for;
  $a = x$; $b = y$;
  return $\langle a, b \rangle$;
End
```

Correctness of The Random Target Algorithm

Assume that the exploratory test system has the following properties.

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

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

where $z = mid(x, y)$.

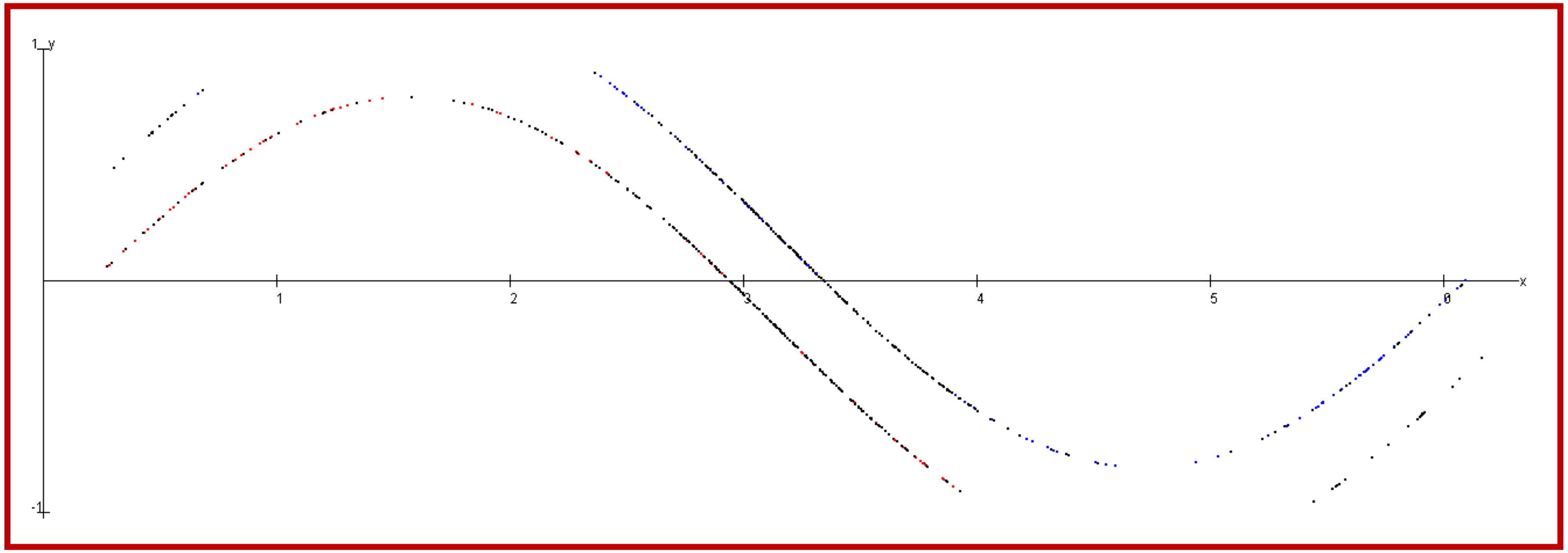
(2) There is a constant $d_m > 0$ such that

$$\forall x, y \in D. (dist(x, y) \leq d_m). \tag{4}$$

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 $d_m/c^n < \delta$.*

Example: Execution of The Random Target Strategy

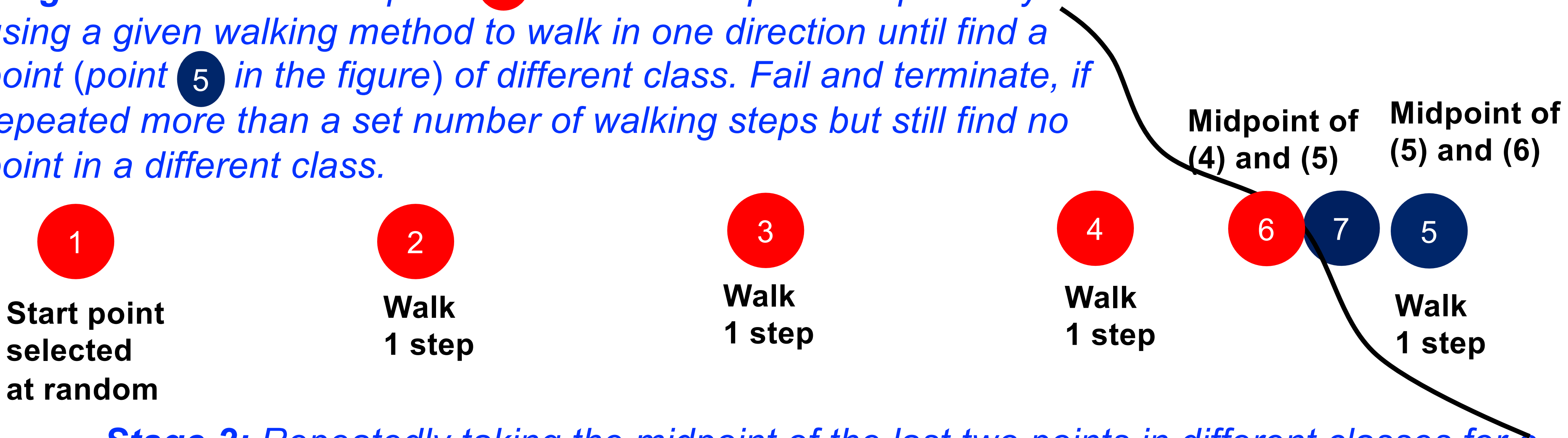
- 1000 random pairs selected from 300 random test cases
- Number of steps: 20
- Number of pairs in the generated pareto front: 641
- Success rate: 64.1%
- The distance between each pair in the Pareto front: $\delta \leq \frac{d_m}{c^{20}} = \frac{\sqrt{\pi^2 + 1}}{2^{19}}.$



Strategy 2: Directed Walk

Select a number of points in D at random as the start points. From each point, use a walking method to traverse in one direction until find a point in different class, then find the border points between them using the midpoint methods repeatedly.

Stage 1: *Start from one point 1 in the data space. Repeatedly using a given walking method to walk in one direction until find a point (point 5 in the figure) of different class. Fail and terminate, if repeated more than a set number of walking steps but still find no point in a different class.*



Stage 2: *Repeatedly taking the midpoint of the last two points in different classes for a number of times to ensure the distance between the last two points is smaller than the required distance of the Pareto.*

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
```

Correctness of The Directed Walk Strategy

Assume that the exploratory test system has the following properties

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

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

where $z = mid(x, y)$.

(2) There is a constant $d_s > 0$ such that

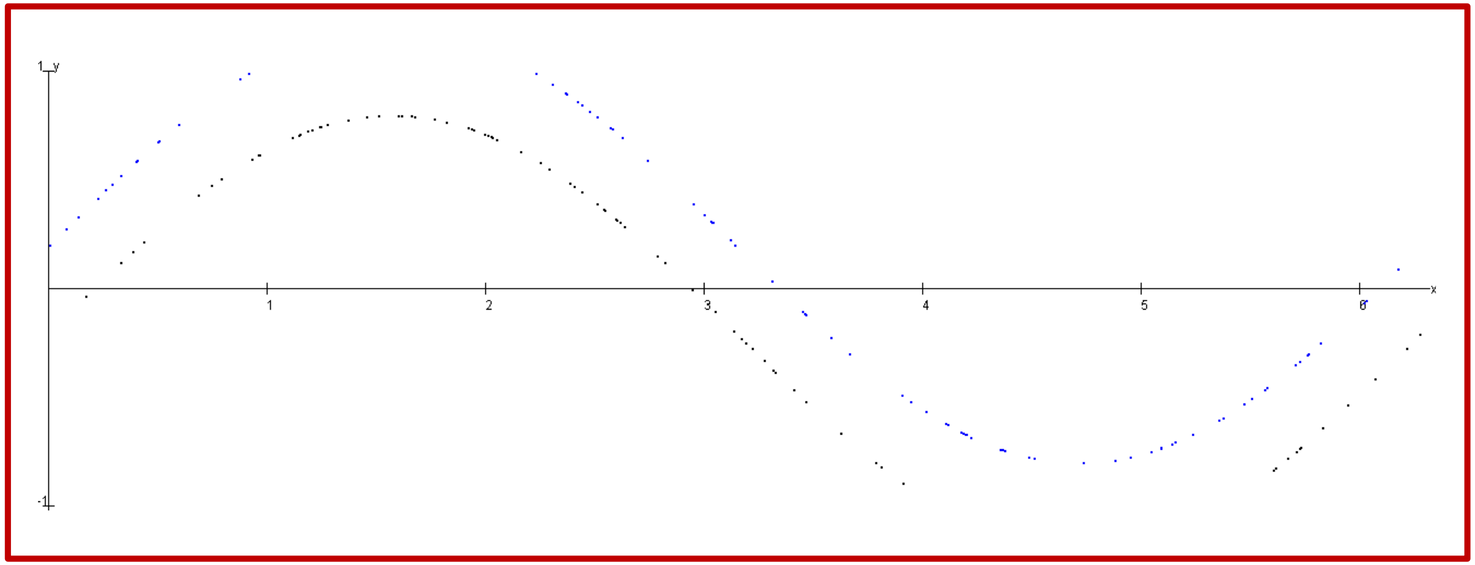
$$\forall x \in D. (dist(x, d(x)) \leq d_s) .$$

where d_s is called the step size of the traversal method $d(x)$.

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.*

Example: Execution of The Directed Walk Strategy

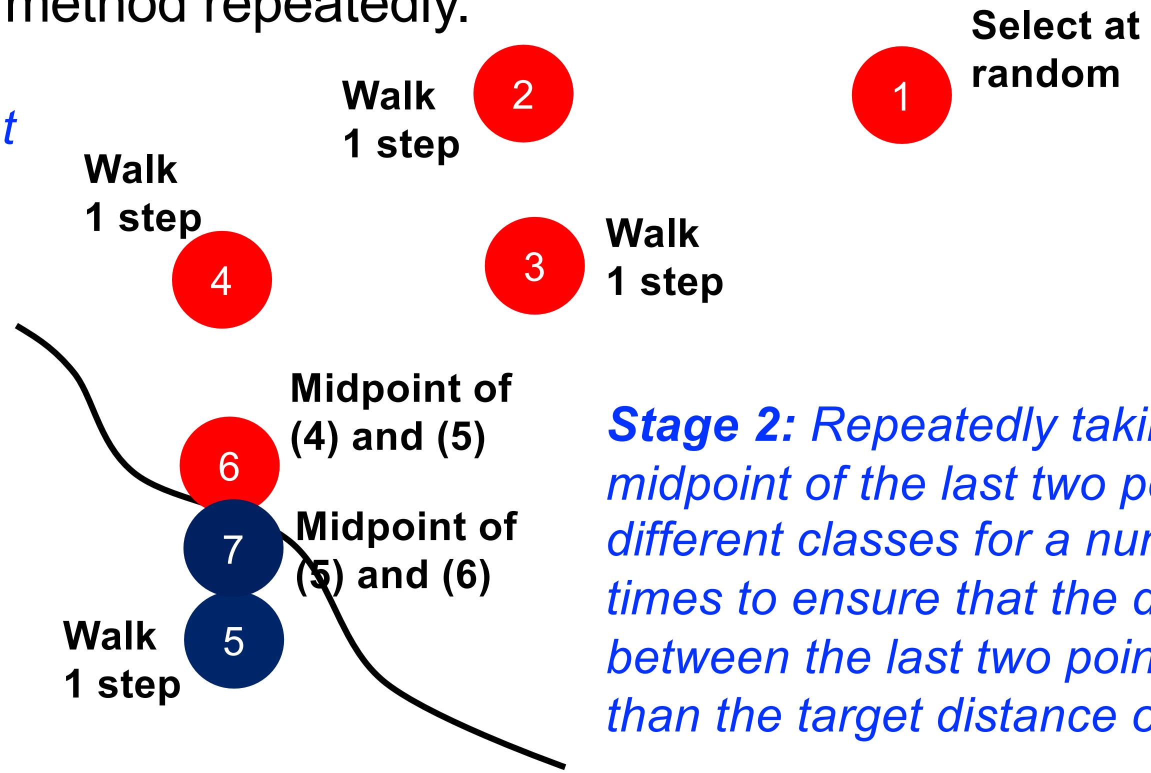
- 1000 start points selected at random; Walk direction: upward
- Walking distance: 20 steps; Number of refinement steps: 20
- Number of pairs in the generated pareto front: 161
- Success rate: 16.1%
- Distance between points in each pair: $\delta \leq \frac{d_s}{c^{20}} = 0.2 \times \frac{1}{2^{20}}.$



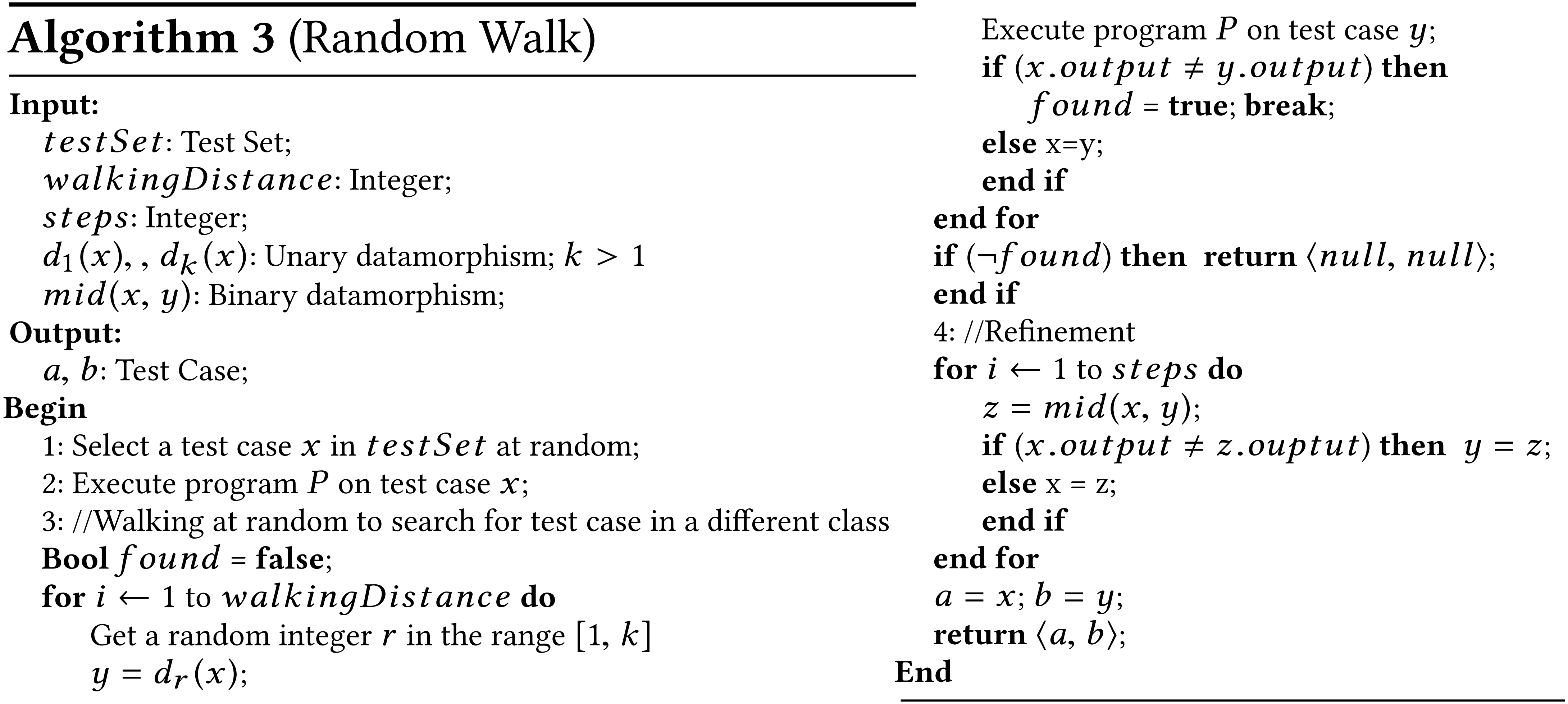
Strategy 3: Random Walk

Select a number of points in D at random as the start points. From each point, use a number of walking method to walk randomly (each step choice a walking method at random), until a point of different class is find, and then find a pair of border points using the midpoint method repeatedly.

***Stage 1:** Start from one point 1, repeatedly using a walking method selected at random until find a point of different class (point 5 in the figure). Fail, if repeated more than a set number of walking steps but still find no point in a different class*



***Stage 2:** Repeatedly taking the midpoint of the last two points in different classes for a number of times to ensure that the distance between the last two points is smaller than the target distance of the pareto*



Algorithm 3 (Random Walk)

```
Input:
    $testSet$: Test Set;
    $walkingDistance$: Integer;
    $steps$: Integer;
    $d_1(x), , d_k(x)$: 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
    if ($\neg found$) 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
```

Correctness of The Random Walk Algorithm

Assume that the exploratory test system has the following properties

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

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

where $z = mid(x, y)$.

(2) There is a constant $d_s > 0$ such that

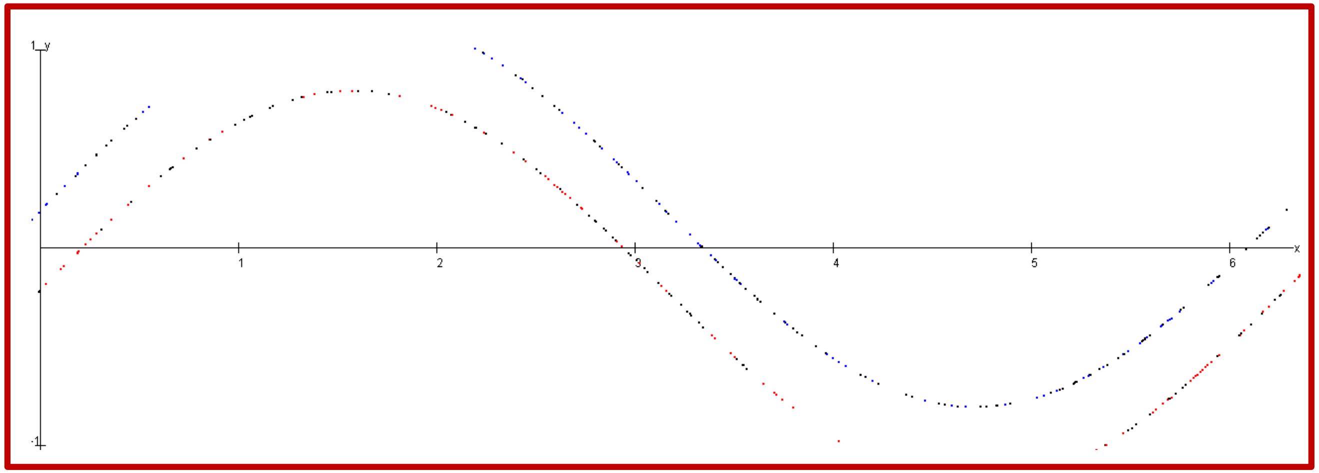
$$\forall x \in D. \forall d_i \in W.(dist(x, d_i(x)) \leq d_{sm}). \quad (6)$$

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.*

Example: Execution of The Random Walk Strategy

- 1000 random walks with 300 starting points selected at random
- Walking distance: 20 steps; Number of refinement steps: 20
- Walk directions: upward, downward, leftward, rightward
- Number of pairs in the generated pareto front: 805
- Success rate: 80.5%
- Distance between points in each pair: $\delta \leq \frac{d_s}{c^{20}} = 0.2 \times \frac{1}{2^{20}}.$



Evaluation of the Strategies

RQ1: Capability

Are the exploratory strategies capable of discovering the borders between subdomains?

RQ2: Cost

Are the exploratory strategies costly for discovering the borders between subdomains?

We measure the cost using the average number of test executions of the classifier for discovering each pair in the Pareto front.

Capability is the probability of a test strategy returning a Pareto front pair when executed.

$$E_m = \frac{\|PF\|}{W}$$

Size of Pareto front of model m

Capacity of testing model m

Number of walks (executions of the strategy)

Cost is the amount of computational resources needed to find a pair in a Pareto front.

$$Time(W) = E_m \times C_m \times W \times s_m$$

Time needed to invoke the model m once

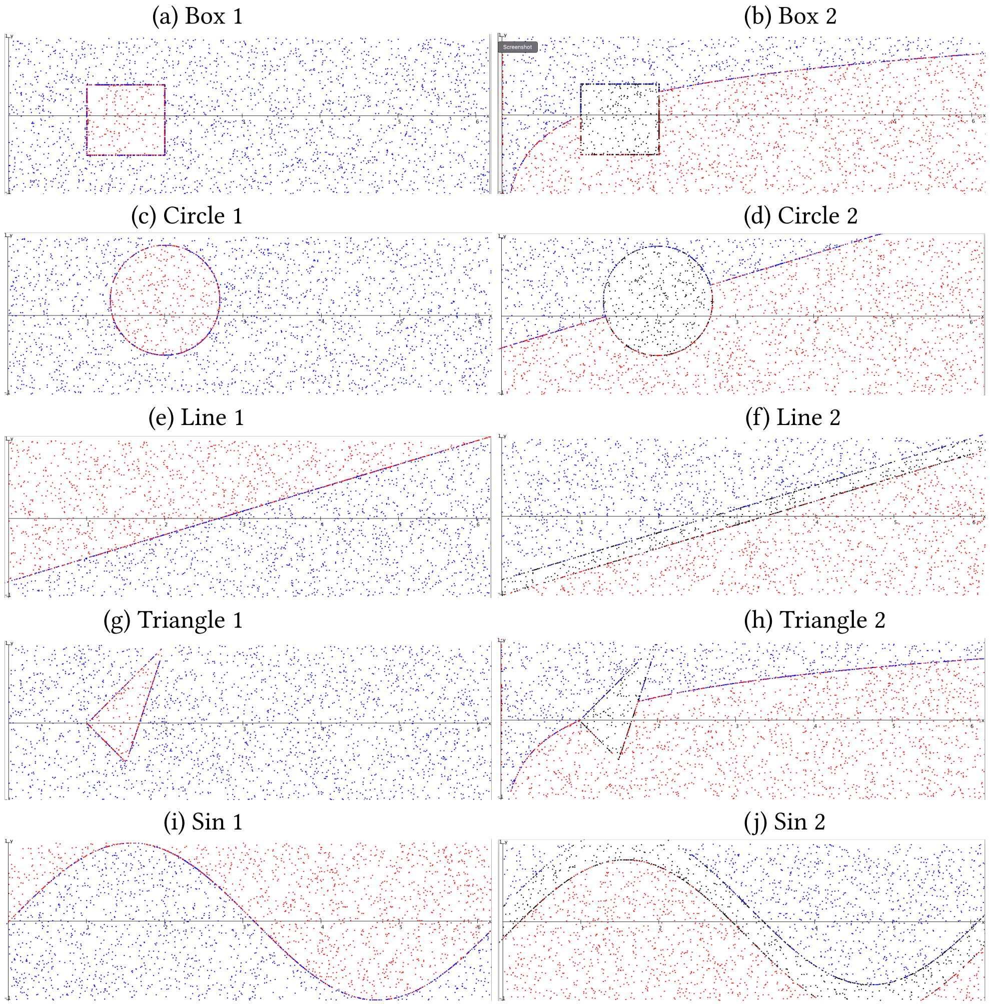
Time needed to take W walks

Cost = Average number of invocations of model m for each pair in PF

Subjects of The Empirical Evaluations (1)

Controlled Experiment with 10 manually coded classifiers

- *Input domain*: Two-dimensional real numbers in the range of $[0, 2\pi] \times [-1, 1]$.
- *Output classes*: {Red, Blue, Black}


(a) Box 1 (b) Box 2 (c) Circle 1 (d) Circle 2 (e) Line 1 (f) Line 2 (g) Triangle 1 (h) Triangle 2 (i) Sin 1 (j) Sin 2

Subjects of The Empirical Evaluations (2)

Case study with ML models built from real datasets

Datasets are drawn at random from Kaggle website.

- **Red Wine Quality**

  Quality of red varieties of the Portuguese "Vinho Verde" wine (*Cortez et al., 2009*).

- **Mushroom Edibility**

  Edibility of hypothetical samples of 23 species of gilled mushrooms in the Agaricus and Lepiota family drawn from The Audubon Society Field Guide (*North American Mushrooms Society, 1981*)

- **Bank Churners**

  Data of creditcard customers used to predict churners, who are bank customers who leave the credit card service.

Dataset	Records	Classes	DF	NF	CF	Features
Red Wine Quality	1599	8	0	0	11	11
Mushroom Edibility	8124	2	22	0	0	22
Bank Churners	10127	2	5	11	3	19

Machine Learning Models Constructed for Each Dataset

Name	Type	Details
LR	Logistic Regression	Trained on whole data set
LR2	Logistic Regression	Used train-test 90-10 split
KNN	K-Nearest Neighbors	Trained on whole data set
KNN2	K-Nearest Neighbors	Used train-test 90-10 split
DT	Decision Tree	Trained on whole data set
DT2	Decision Tree	Used train-test 90-10 split
NB	Naive Bayes	Trained on whole data set
NB2	Naive Bayes	Used train-test 90-10 split
SVM	Surportting vector machine	Trained on whole data set
SVM2	Surportting vector machine	Used train-test 90-10 split
SV	Ensemble via Soft voting	Trained on whole data set; LR+KNN+DT
SV2	Ensemable via Soft Voting	Used train-test 90-10 split; LR+KNN+DT
HV	Ensemble via Hard Voting	Trained on whole data set; LR+KNN+DT
HV2	Ensemble via Hard Voting	Used train-test 90-10 split; LR+KNN+DT
Stack1	Ensemble via Stacking	Used train-test 90-10 split; KNN as Meta; LR2+KNN2+DT2+HV2
Stack3	Ensemble via Stacking	Used train-test 90-10 split; LR as Meta; KNN2+DT+SV2+HV2

A total of 48 machine learning models are built and used in the case study.

Experiment Process

- For each subject application, three exploration strategies are executed with various parameters
- For each setting of parameters, the exploration strategy algorithm is executed repeatedly for 10 times
- For each execution of the strategy on each model, the number of invocations of the model under test and the size of Pareto front generated are recorded
- The average of the data collected in 10 executions is used to analyse the results

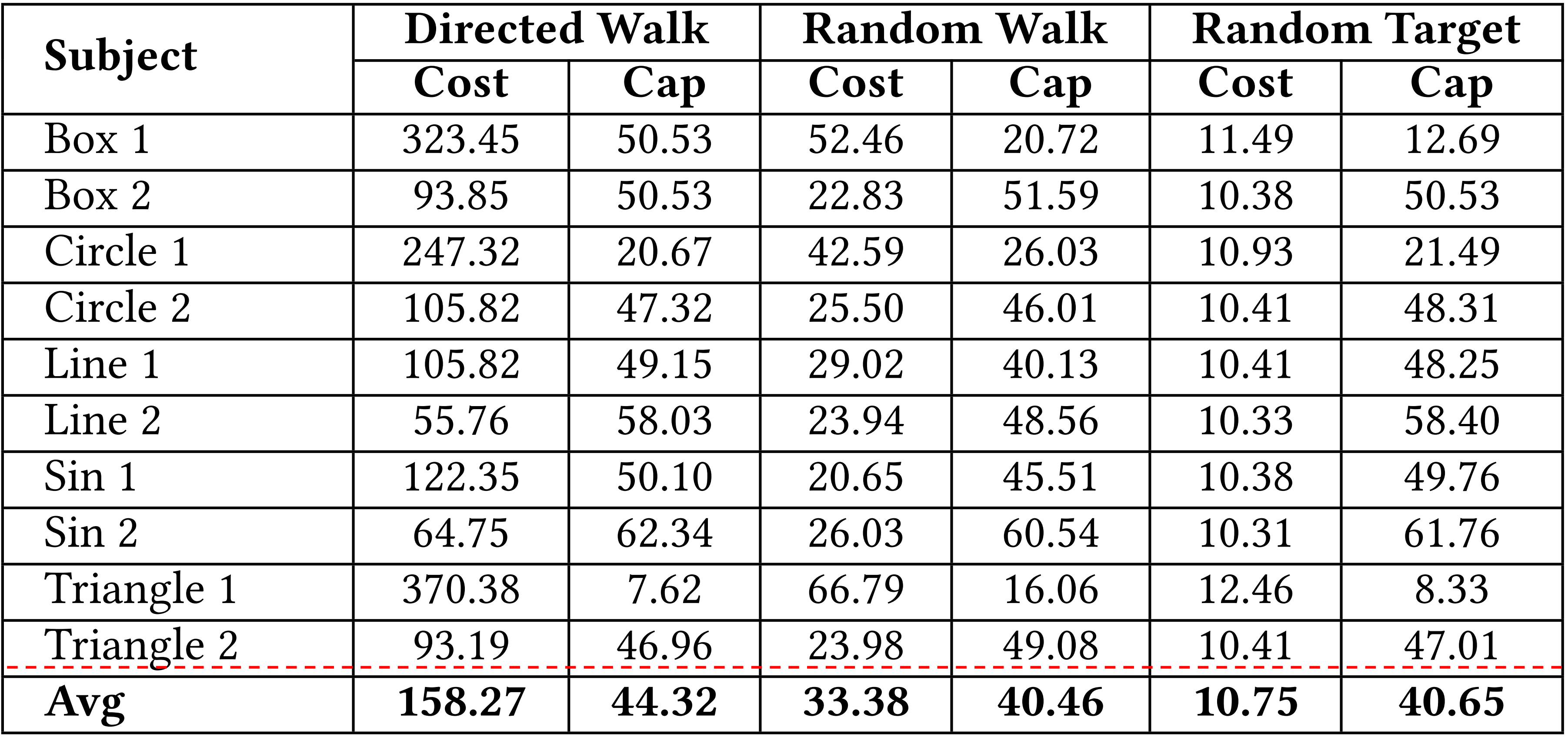
Used the testing tool Morphy

- The exploratory test system are written in Java
- Morphy test scripts are written to automatically conducted the experiments
- Tests are executed using Morphy

Morphty tool, test code, test scripts and data are on GitHub:

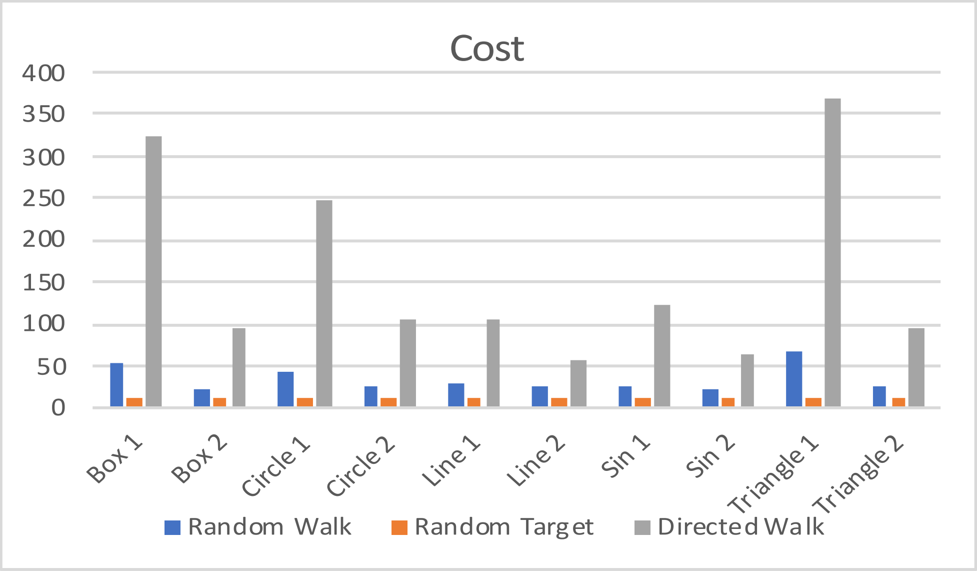
https://github.com/hongzhu6129/ExploratoryTestAI.git

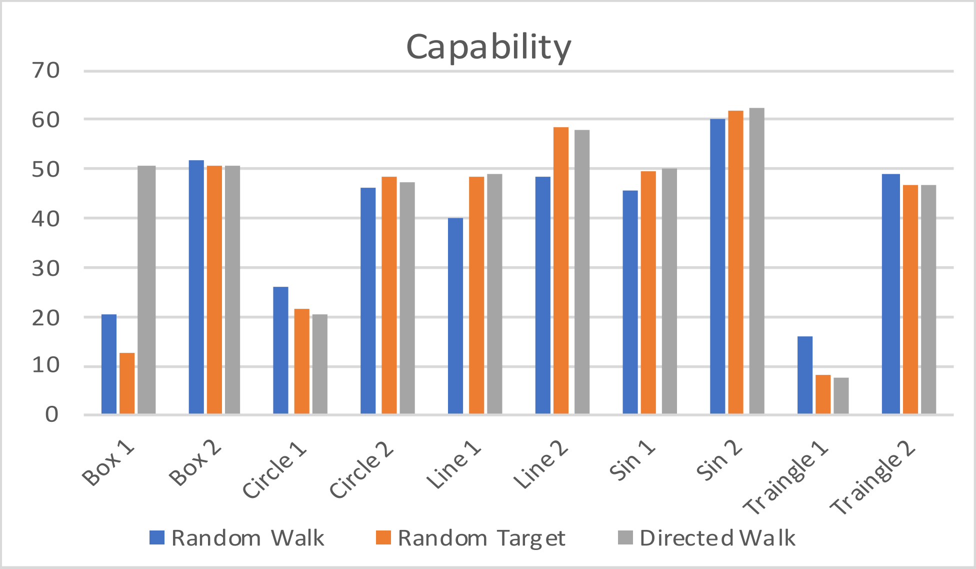
Main Results: Coded Classifiers



Subject	Directed Walk		Random Walk		Random Target	
	Cost	Cap	Cost	Cap	Cost	Cap
Box 1	323.45	50.53	52.46	20.72	11.49	12.69
Box 2	93.85	50.53	22.83	51.59	10.38	50.53
Circle 1	247.32	20.67	42.59	26.03	10.93	21.49
Circle 2	105.82	47.32	25.50	46.01	10.41	48.31
Line 1	105.82	49.15	29.02	40.13	10.41	48.25
Line 2	55.76	58.03	23.94	48.56	10.33	58.40
Sin 1	122.35	50.10	20.65	45.51	10.38	49.76
Sin 2	64.75	62.34	26.03	60.54	10.31	61.76
Triangle 1	370.38	7.62	66.79	16.06	12.46	8.33
Triangle 2	93.19	46.96	23.98	49.08	10.41	47.01
Avg	**158.27**	**44.32**	**33.38**	**40.46**	**10.75**	**40.65**

Main Results: Coded Classifiers

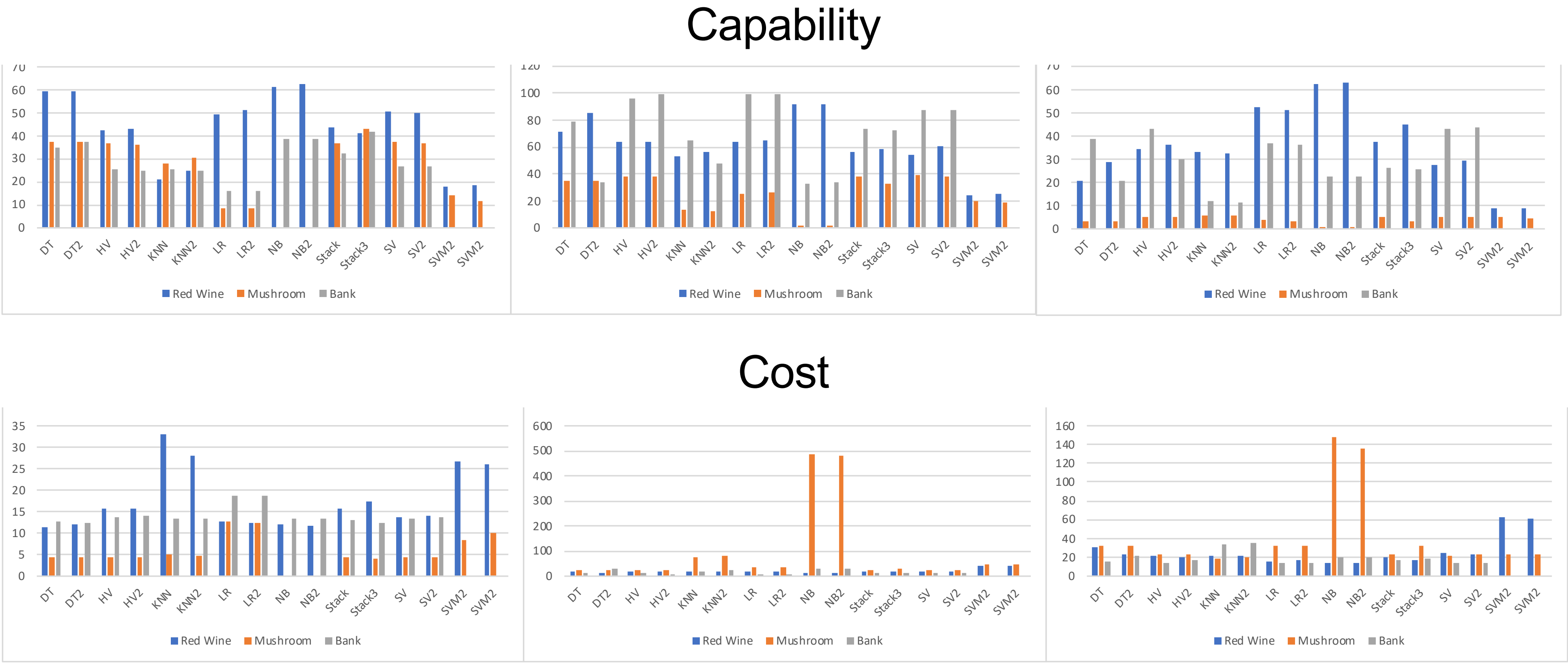




Main Results: Real Machine Learning Models

Strategy	Subject	Cost				Capability			
		Max	Min	Avg	StDev	Max	Min	Avg	StDev
Directed Walk	Red Wine Quality	63.03	14.12	25.70	0.15	62.89	8.79	35.74	0.24
	Mushroom Edibility	32.63	18.90	25.57	0.38	5.79	0.80	4.10	0.06
	Bank Churners	35.56	14.07	19.26	0.21	43.43	0.00	25.75	0.21
Random Target	Red Wine Quality	33.14	11.47	17.39	0.46	62.51	18.18	43.62	0.72
	Mushroom Edibility	12.61	3.92	6.23	0.26	43.05	0.00	25.18	0.59
	Bank Churners	18.81	12.40	14.06	0.18	41.66	0.00	25.60	0.64
Random Walk	Red Wine Quality	40.87	14.31	20.71	0.39	91.87	24.12	61.61	0.87
	Mushroom Edibility	488.50	21.42	92.01	6.35	38.87	2.15	25.87	0.63
	Bank Churners	30.34	8.10	15.94	0.28	99.43	0.00	62.83	0.47

Main Results: Real Machine Learning Models



(a) Random Target (b) Random Walk (c) Directed Walk

Main Findings 1: Answers to Research Questions

- RQ1: The strategies are capable of discovering borders between subdomains.
  - The overall average of the capabilities of all three strategies: 34.48%.
  - Directed walk: 21.86%
  - Random target: 31.47%
  - Random walk: 50.10%
- RQ2: Applying exploratory strategies is cost efficient for discovering borders between classes.
  - The overall average cost: 26.32 (of three strategies over all subjects)
  - The best cost: 6.23. (achieved in the testing of mushroom edibility models using the random target strategy)
  - The worst cost: 92.01 (observed also when testing mushroom edibility but using the random walk strategy).

Main Findings 2: Factors that Determine Capability

- Directed walk strategy:

  The probability that there is a border between two subdomains in the right direction from a test case and within the walking distance

- Random target strategy:

  The probability that two random test cases fall in two different subdomains

- Random walk strategy:

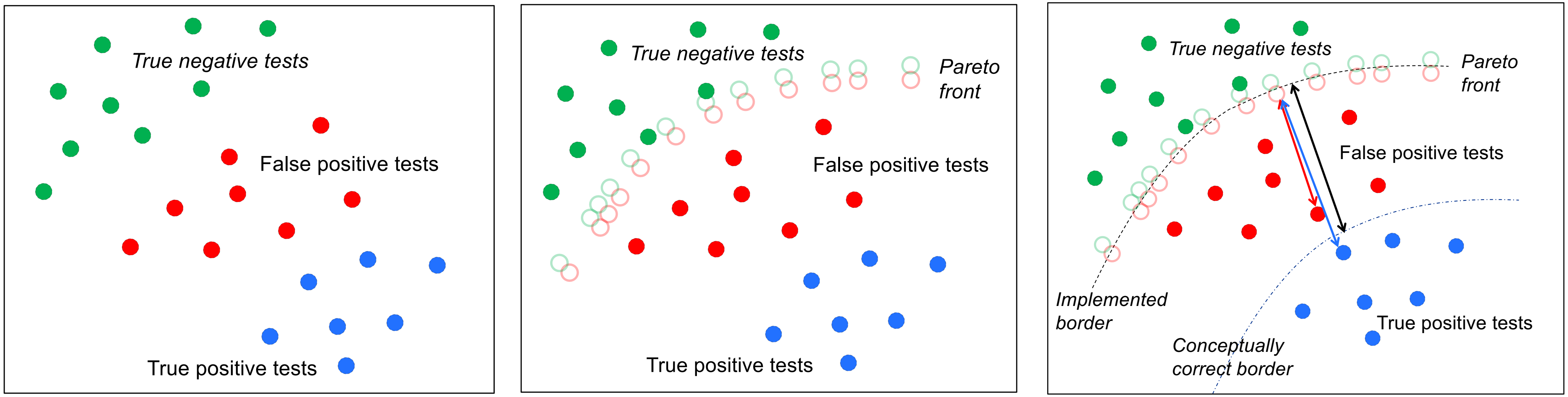
  The probability that there is a border nearby to a randomly selected test case

Main Findings 3: Properties of The Strategies

The data of the case study of real machine learning models are consistent with the data of the controlled experiments on both capability and cost of the strategies.

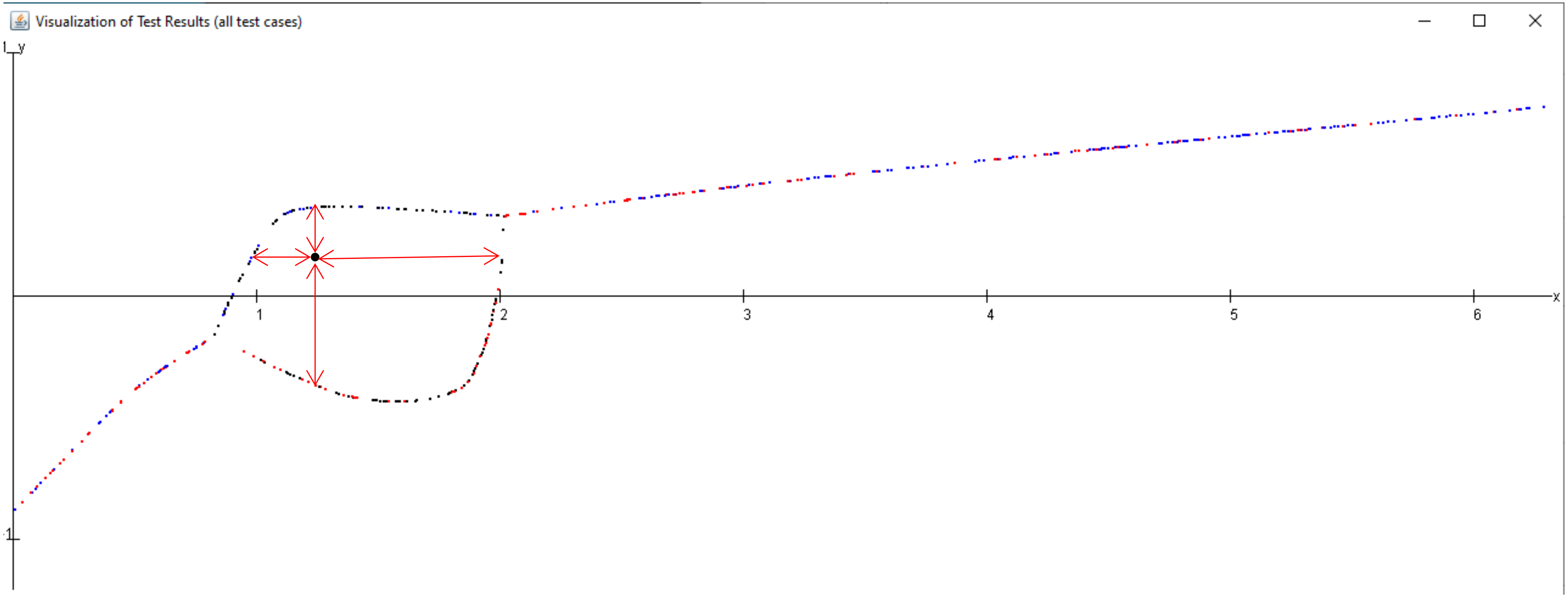
- The capability and cost are invariant in the number of walks.
  - Both cost and capability are constants that only vary with the model under test.
- The dimensions of the input data spaces of the real-world examples are significantly larger than those coded classifiers.
  - The strategies are scalable to high dimensional data spaces.

Uses of Pareto Front: 1. Measuring Error Extent



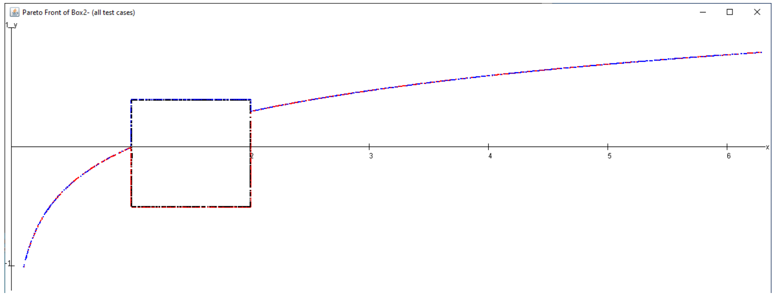
Hong Zhu, and Ian Bayley, ***Discovering boundary values of feature-based machine learning classifiers through exploratory datamorphic testing***, Journal of Systems and Software, Vol. 187, Article 111231, May 2022.

Uses of Pareto Front: 2. Explanation

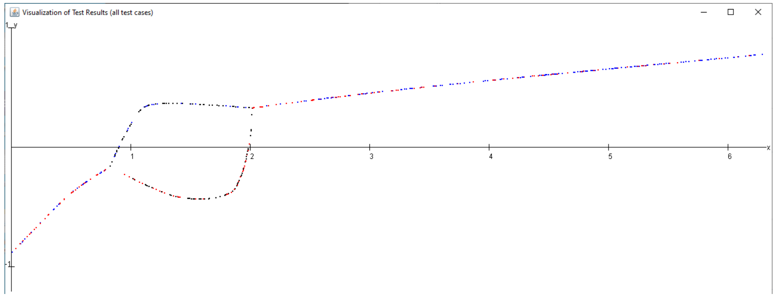


Deep Neural Network (DNN)

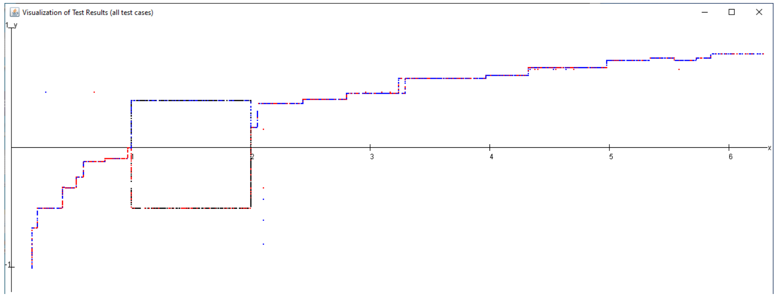
Uses of Pareto Front: 3. Visualisation



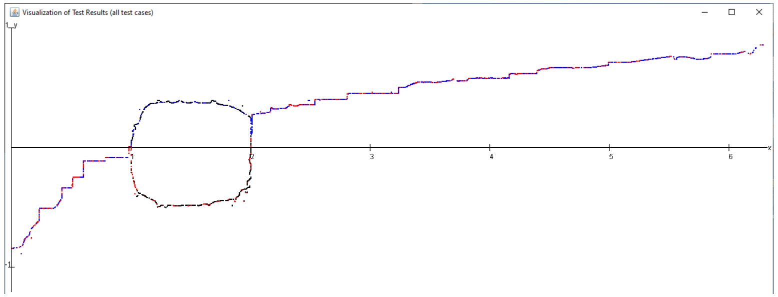
Original Coded Classifier



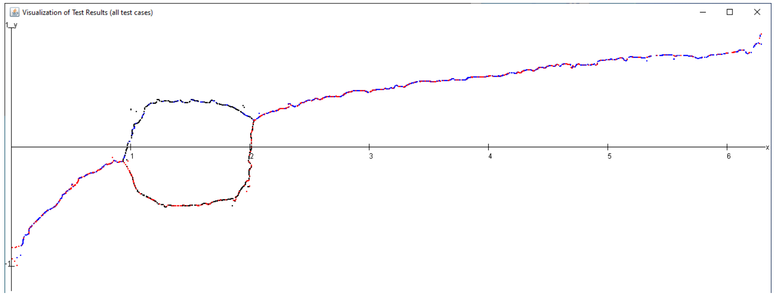
Deep Neural Network (DNN)



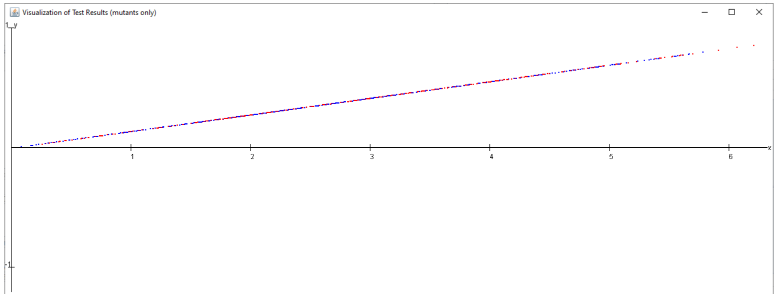
Decision Tree (DT)



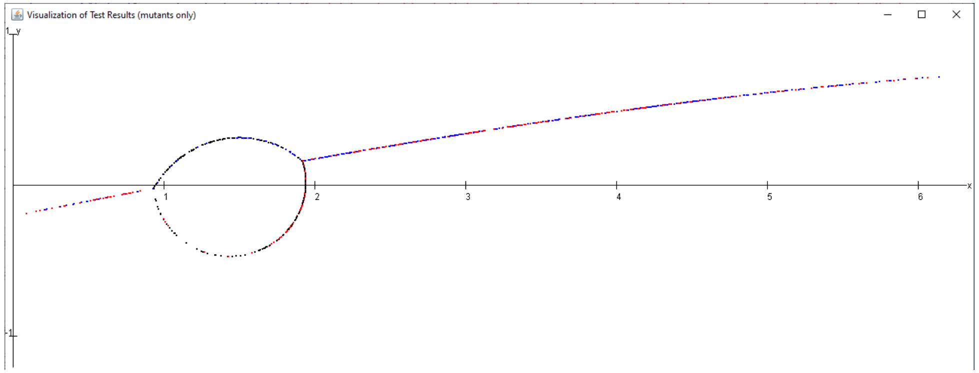
Hard Voting of LR, KNN and DT (HV)



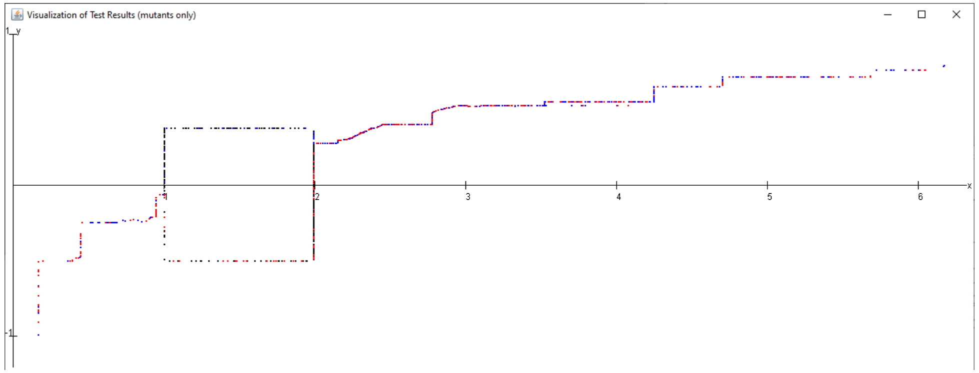
K-Nearest Neighbour (KNN)



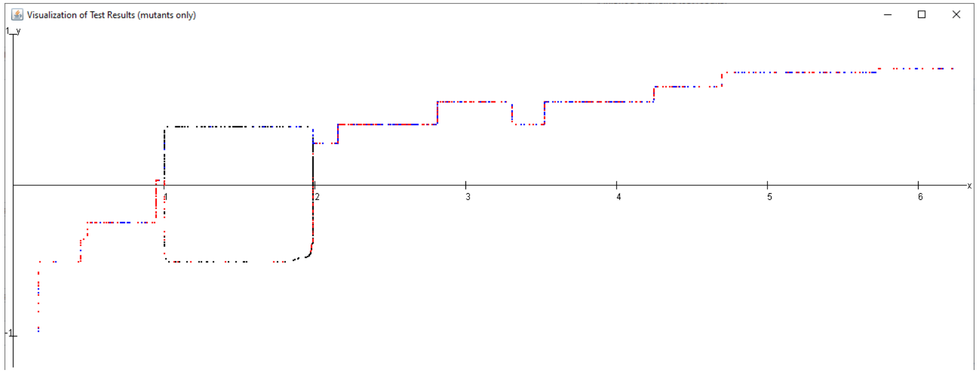
Logistic Regression (LR)



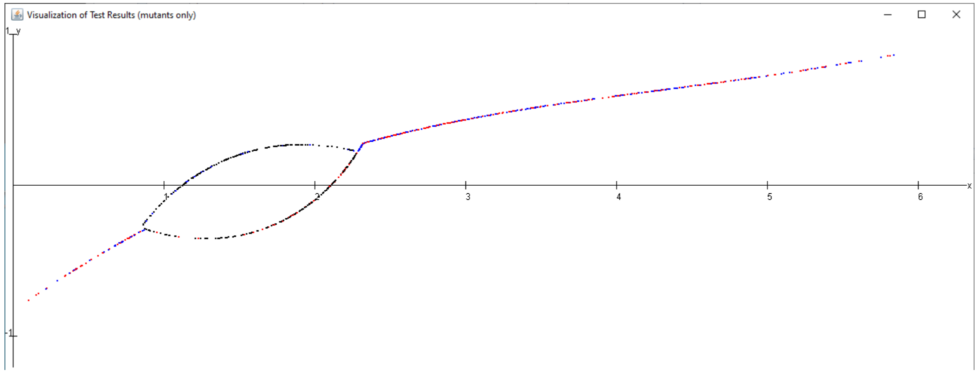
Naïve Bayes (NB)



Stacking KNN over LR, DT and HV (Stack)

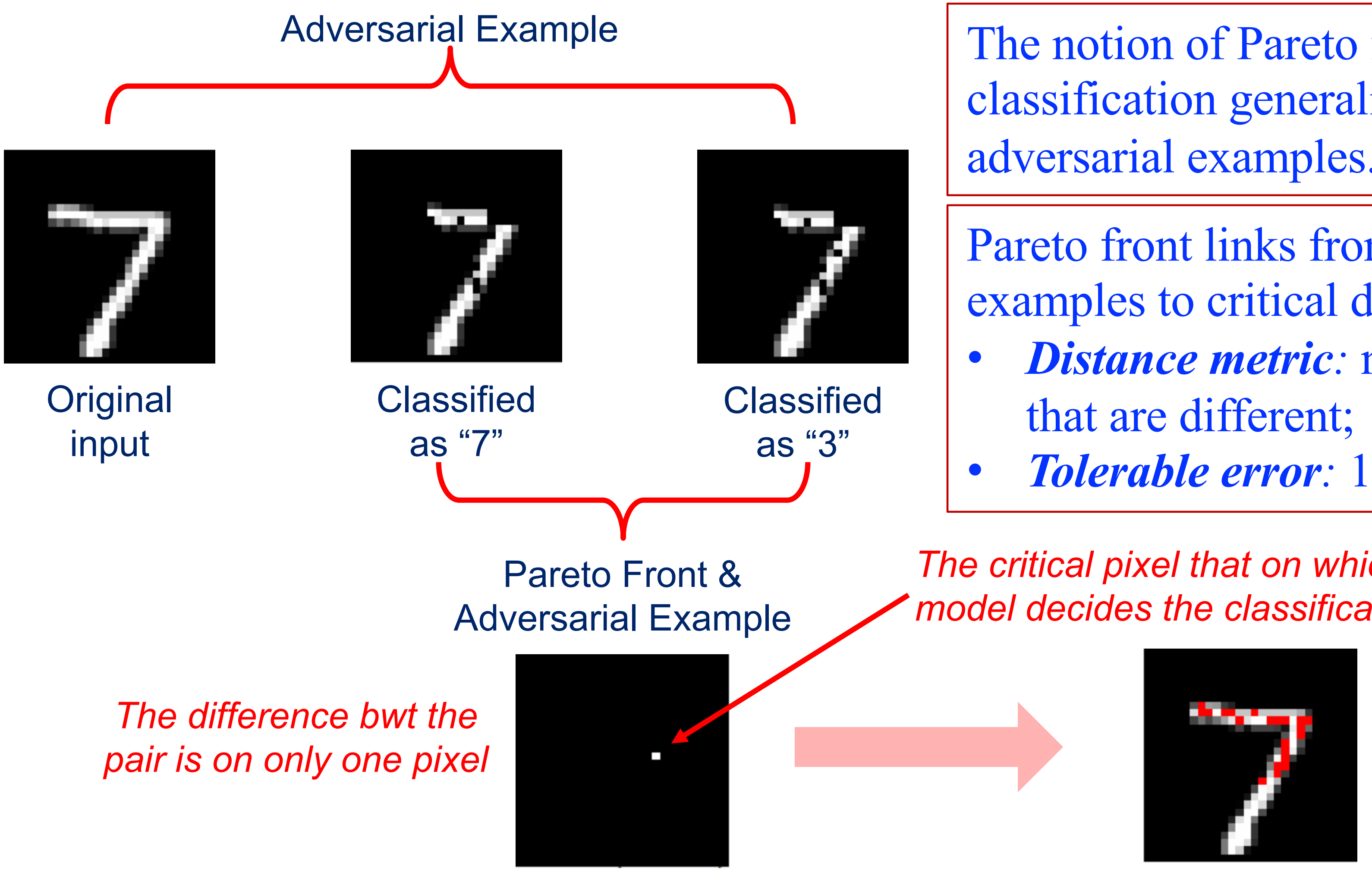


Soft Voting of LR, KNN and DT (SV)



Supporting Vector Machine (SVM)

Uses 4: Testing ML Model's Robustness



The difference bwt the pair is on only one pixel

The critical pixel that on which the ML model decides the classification

The notion of Pareto front of classification generalises the notion of adversarial examples.

Pareto front links from adversarial examples to critical decision region

- ***Distance metric***: number of pixels that are different;
- ***Tolerable error***: 1

Decision region = the set of critical pixels

Scenario-based Exploratory Functional Testing

CISOSE 2023 Invited Track

Session 5, 17th July 2023 (Monday) 14:00pm

(Auditorium)

Hong Zhu, et al., **A Scenario-Based Functional Testing Approach to Improving DNN Performance,** Proc. of SOSE 2023 (In press)

Thank You