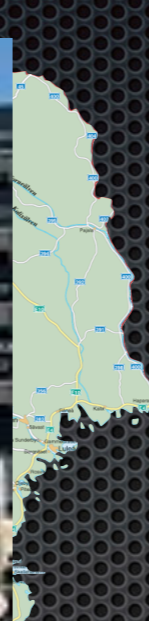


Test Diversity as a General Driver for Test Automation

Robert Feldt, Chalmers University of Technology,
Gothenburg, Sweden
robert.feldt@chalmers.se
@drfeldt on Twitter



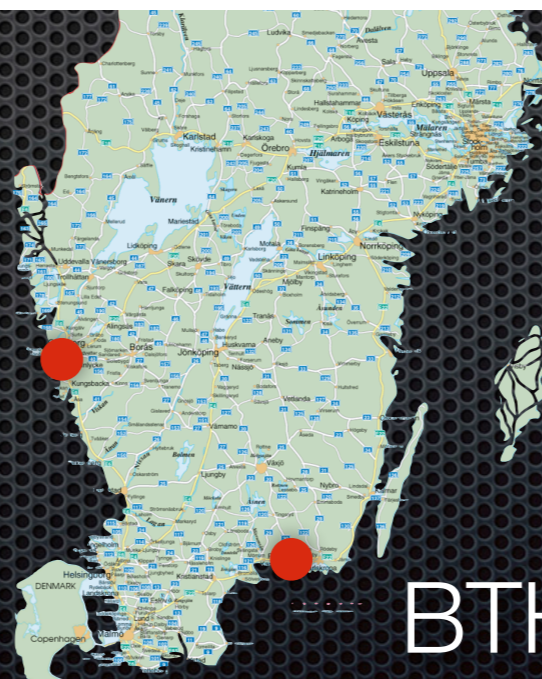
CHALMERS



40TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING

MAY 27 - JUNE 3 2018
GOTHENBURG, SWEDEN

Chalmers,
Göteborg



BTH, Karlskrona

Testing still (mainly) based on intuition & heuristics

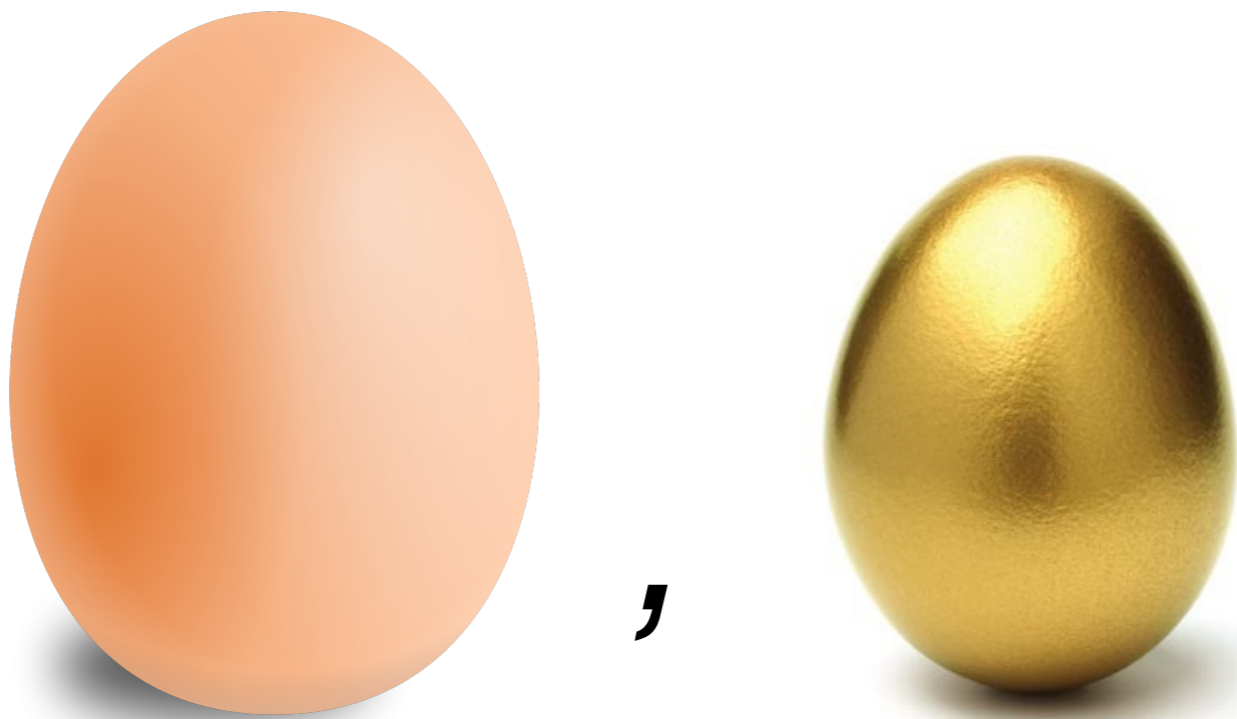
“To better cover system behaviour, run **different** test cases”

“Don’t put all your eggs in one basket”, spread the risk



To formalise, analyse, automate etc we need to **quantify!**

There are MANY distance functions

$$d_1(\text{egg}, \text{egg}) = \text{num}$$
An orange egg is on the left and a golden egg is on the right. They are separated by a comma and enclosed in large parentheses.

$$d_2(\text{avocado}, \text{avocado}) = \text{num}$$
A dark, almost black avocado is on the left and a bright green avocado is on the right. They are separated by a comma and enclosed in large parentheses.

They are (always) pair-wise and/or data-dependent

$d(\text{ }) = num$



Today we'll talk some about Test Set Diameter (TSDm):

- Works for any test information / data type
 - Inputs, Outputs, State, Traces...
- Measures distance of a whole multiset, not just pairs
- And shows that test sets selected by it
 - increases code and fault coverage

So what is Information Theory?

Application of probability theory & statistics to problems of quantification, storage and communication of information.

Reprinted with corrections from *The Bell System Technical Journal*,
Vol. 27, pp. 379–423, 623–656, July, October, 1948.

A Mathematical Theory of Communication

By C. E. SHANNON

INTRODUCTION

THE recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist¹ and Hartley² on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise in the channel, and the savings possible due to the statistical structure of the original message and due to the nature of the final destination of the information.

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one *selected from a set* of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.



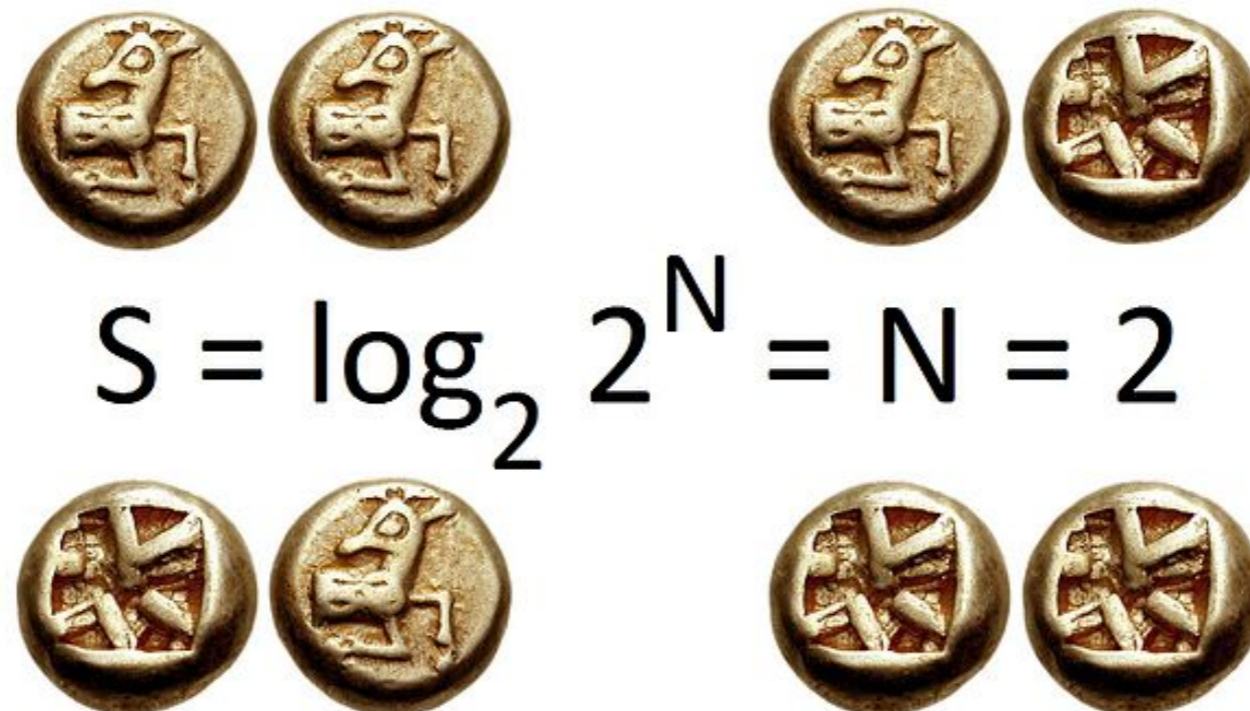
Entropy a key concepts of Information Theory

Information Entropy = *quantifies the amount of uncertainty in a random variable.*

= average amount of information conveyed by an event, when considering all possible outcomes.

*Entropy is measured in **bits**.*

*Alternatively called "**shannons**".*



Kolmogorov wanted a measure for single objects

THREE APPROACHES TO THE QUANTITATIVE DEFINITION
OF INFORMATION

A. N. Kolmogorov

Problemy Peredachi Informatsii, Vol. 1, No. 1, pp. 3-11, 1965

There are two common approaches to the quantitative definition of "information": combinatorial and probabilistic. The author briefly describes the major features of these approaches and introduces a new algorithmic approach that uses the theory of recursive functions.



“Actually, it is most fruitful to discuss the quantity of information ‘conveyed by an object’ x ‘about another object’ y .”

As the "relative complexity" of an object y with a given x , we will take the minimal length $l(p)$ of the "program" p for obtaining y from x . The definition thus formulated depends on the "programming method," which is nothing other than the function

$$\varphi(p, x) = y,$$

Kolmogorov complexity of object $x = K(x) =$ length of shortest program to generate x (given no input)

The “Compression trick”

Kolmogorov complexity is extremely powerful in theory but cannot be calculated in practice. Enter Cilibrasi and Vitanyi with the **Compression trick**:



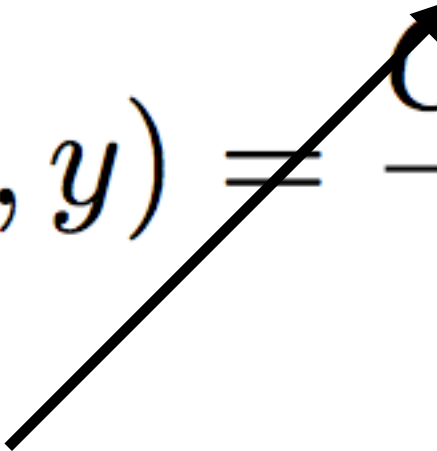
Assuming a good, general compressor, c , with no “bias”, we can approximate $K(x)$ with $C(x) = \text{length}(c(x))$.

We can apply this trick to a large number of theoretical results and formulas and get methods that often works surprisingly well in practice.

Information distance

Roughly speaking, two objects are deemed close if we can significantly “compress” one given the information in the other, the idea being that if two pieces are more similar, then we can more succinctly describe one given the other.

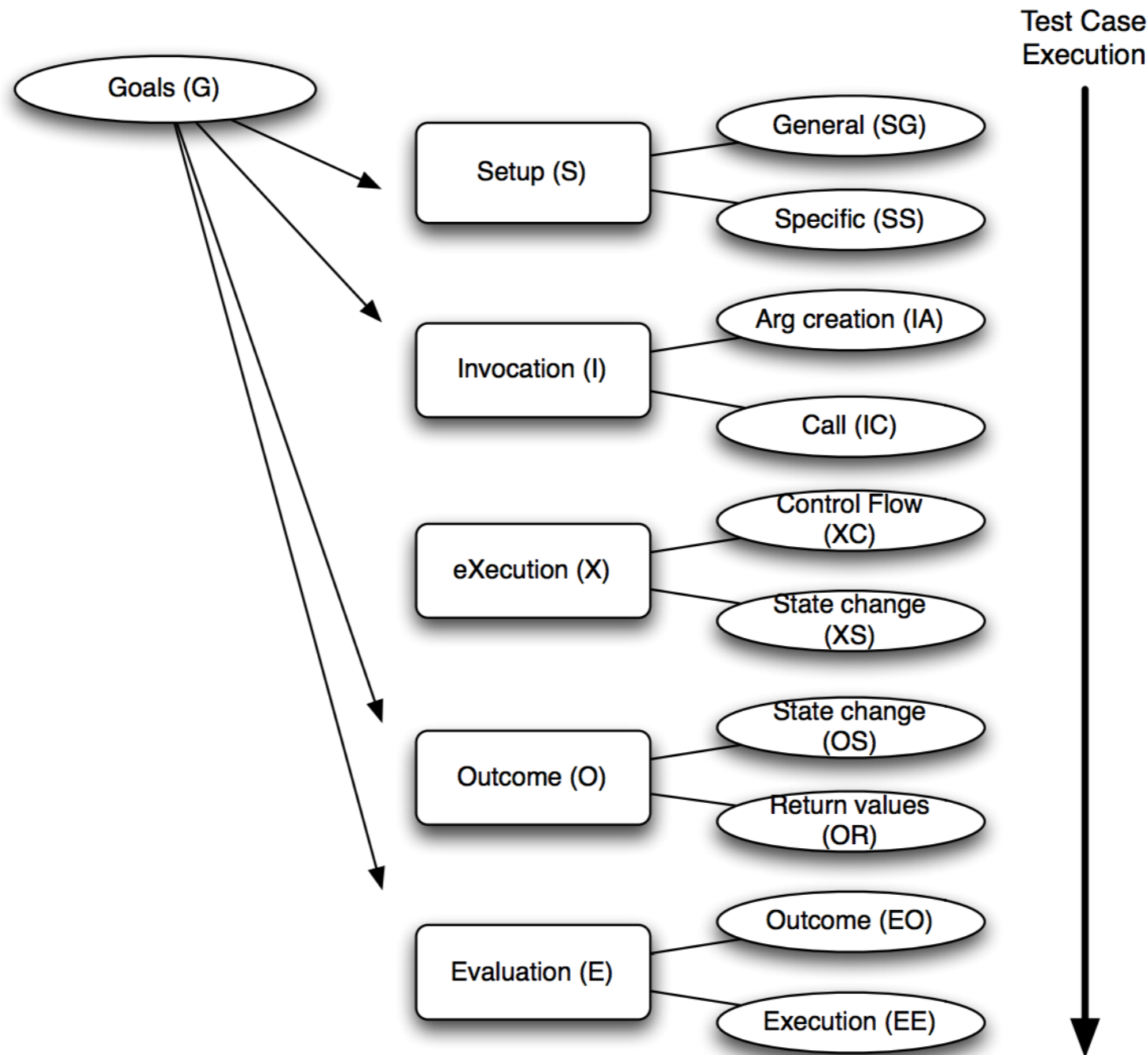
Already at ICST 2008 in Lillehammer...

$$\text{NCD}(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}$$


Information distance between two strings x & y is the length of the shortest program that outputs x given input y , or that outputs y given input x , whichever is largest

(zlib, bzip2, ppm, blosc, lz4, zstandard, ...)

Many sources of test case information



VAriability of Tests (VAT) Model of test information sources/types

Test Set Diameter:

Quantifying the Diversity of Sets of Test Cases

Robert Feldt, Simon Poulding, David Clark, and Shin Yoo



**DEPARTMENT
OF SOFTWARE
ENGINEERING**



KAIST



NCD for multisets (aka “bags”, “lists”, ...)

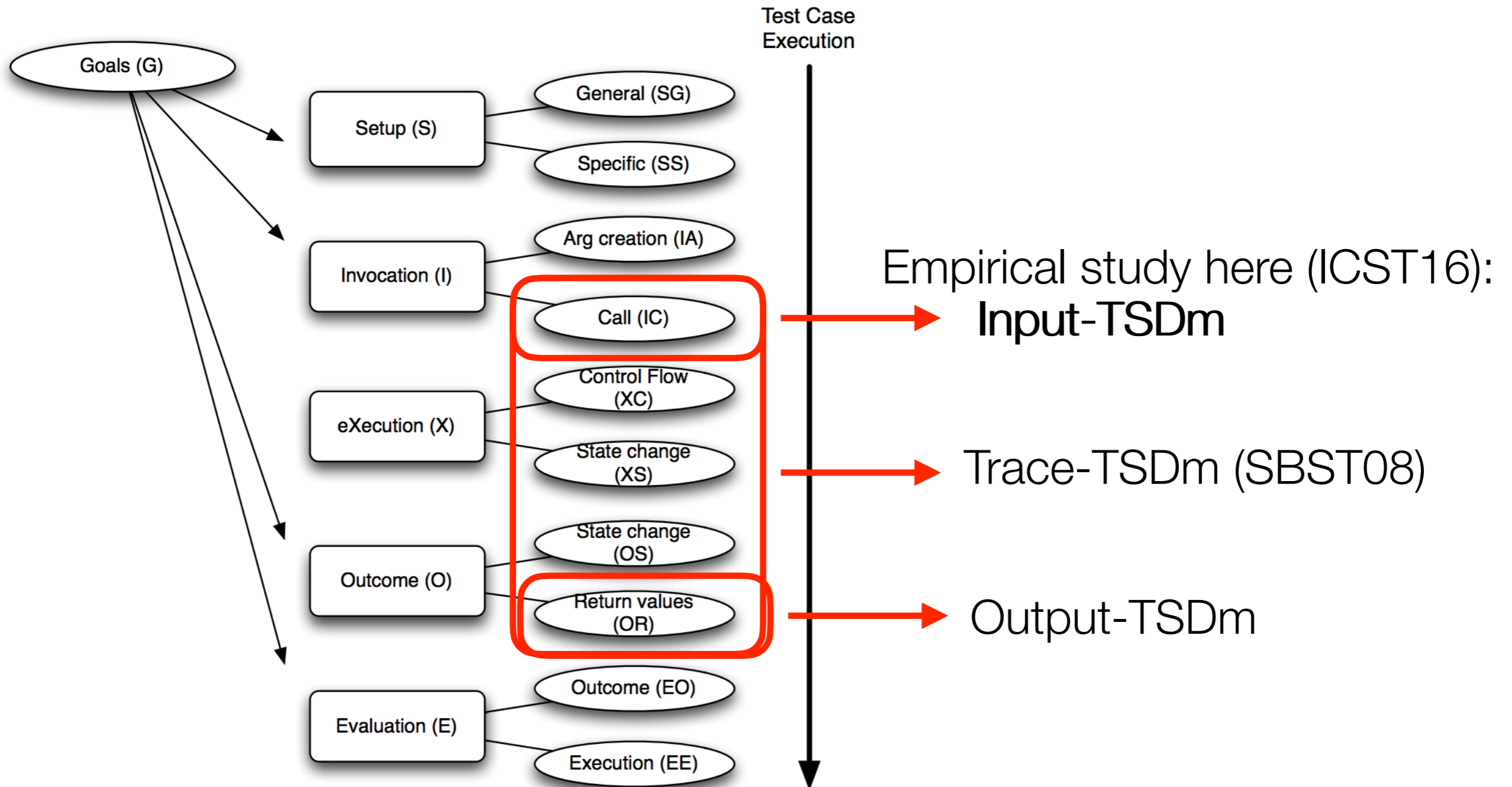
$$\text{NCD}_1(X) = \frac{C(X) - \min_{x \in X} \{C(x)\}}{\max_{x \in X} \{C(X \setminus \{x\})\}}$$

$$\text{NCD}(X) = \max \left\{ \text{NCD}_1(X), \max_{Y \subset X} \{\text{NCD}(Y)\} \right\}$$

The algorithm starts from the multiset $Y_0 = X = \{x_1, x_2, \dots, x_n\}$, and proceeds as:

- 1) Find index i that maximizes $C(Y_k \setminus \{x_i\})$.
- 2) Let $Y_{k+1} = Y_k \setminus x_i$.
- 3) Repeat from step 1 until the subset contains only two strings.
- 4) Calculate $\text{NCD}(X)$ as: $\max_{0 \leq k \leq n-2} \{\text{NCD}_1(Y_k)\}$.

TSDm = NCDm(subset of VAT info)



Empirical study on Input-TSDm

SUT	Input	Size (LOC)	Language	Measure
JEuclid	MathML (XML)	11,556	Java	Instruction Cov
ROME	RSS/Atom (XML)	11,704	Java	Instruction Cov
NanoXML	XML	1,630	Java	Instruction Cov
Replace	2 strings & 1 Regex	538	C	Fault cov (seeded)

RQ1 – Correlation to code coverage: Are higher levels of I-TSDm associated with higher levels of code coverage?

RQ2 – Structural coverage ability: Do test sets selected based on I-TSDm lead to higher code coverage than randomly selected test sets?

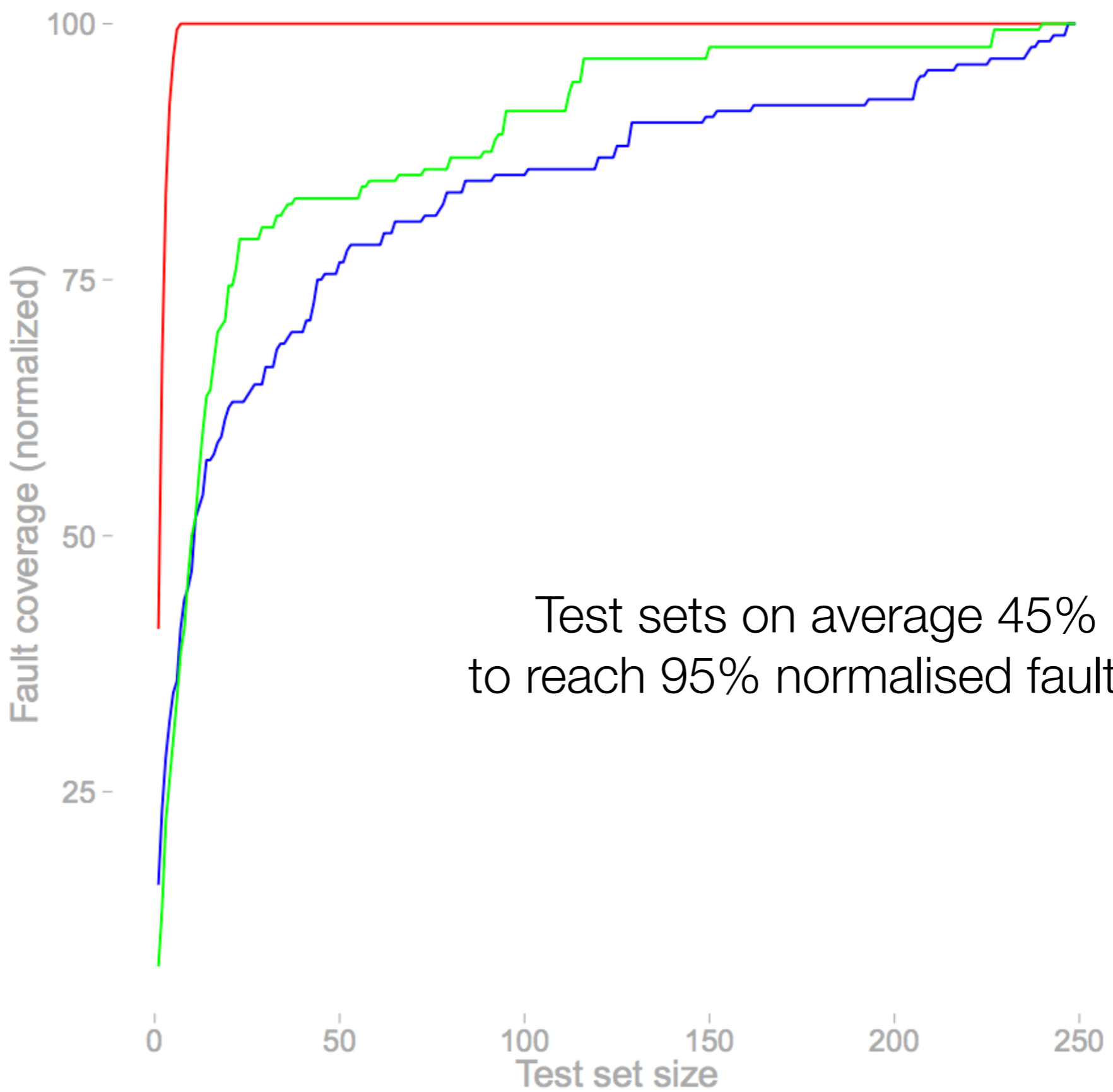
RQ4 – Fault finding ability: Do test sets selected based on I-TSDm lead to higher fault coverage than test sets based on random selection?

RQ2: Higher code coverage if select based on Input-TSDm?

SUT	Avg. Test Set Size					
	I-TSDm			Random		
	90%	95%	99%	90%	95%	99%
JEuclid	29.9	40.9	90.3	82.2	135.3	217.3
NanoXML	1.9	19.4	75.1	18.7	38.2	207.2
ROME	9.1	21.7	51.3	21.9	51.0	129.0



RQ4: Higher fault coverage if select based on Input-TSDm?



Test sets on average 45% smaller to reach 95% normalised fault coverage

Conclusions of the TSDm study

- We proposed & evaluated Test Set Diameter
- General & Universal Measure for **Diversity of Test Sets**
 - Works for any type of data and information source
 - Family of diversity metrics
 - Easy to implement but fairly slow
- Evaluated TSDm on sets of test inputs
 - One of the more ambitious tasks in testing
 - Reduces test set size 2x to 10x compared to random
- Useful & important concept for SW Quality in general:
 - Not only for automated test creation
 - Also analyse manual test suites & tester behaviour

TSDm has already been applied by others :)

Comparing White-box and Black-box Test Prioritization

Christopher Henard
University of Luxembourg
christopher.henard@uni.lu

Mike Papadakis
University of Luxembourg
michail.papadakis@uni.lu

Mark Harman
University College London
mark.harman@ucl.ac.uk

Yue Jia
University College London
yue.jia@ucl.ac.uk

Yves Le Traon
University of Luxembourg
yves.letraon@uni.lu

ABSTRACT

Although white-box regression test prioritization has been well-studied, the more recently introduced black-box prioritization approaches have neither been compared against each other nor against more well-established white-box techniques. We present a comprehensive experimental comparison of several test prioritization techniques, including well-established white-box strategies and more recently introduced black-box approaches. We found that Combinatorial Interaction Testing and diversity-based techniques (Input Model Diversity and Input Test Set Diameter) perform best among the black-box approaches. Perhaps surprisingly, we found little difference between black-box and white-box performance (at most 4% fault detection rate difference).

We also found the overlap between black- and white-box faults to be high: the first 10% of the prioritized test suites already agree on at least 60% of the faults found. These are positive findings for practicing regression testers who may not have source code available, thereby making white-box techniques inapplicable. We also found evidence that both black-box and white-box prioritization remain robust over multiple system releases.

Although white-box techniques have been extensively studied over two decades of research on regression test optimization [25, 30, 47, 65], black-box approaches have been less well studied [35, 36, 46]. Recent advances in black-box techniques have focused on promoting diversity among the test cases, with results reported for test case generation [9, 16, 18, 50] and for regression test prioritization [14, 35, 56, 69]. However, these approaches have neither been compared against each other, nor against more traditional white-box techniques in a thorough experimental study. Therefore, it is currently unknown how the black-box approaches perform, compared to each other, and also compared to the more traditionally-studied white-box techniques.

Black-box testing has the advantage of not requiring source code, thereby obviating the need for instrumentation and source code availability. Conversely, one might hypothesize that accessing source code information would allow white-box testing to increase source code coverage and, thereby, to increase early fault revelation. It has also been claimed that white-box techniques can be expensive [49] and that the use of coverage information from previous versions might degrade prioritization effectiveness over multiple releases [59]. These hypotheses and claims call out for a thorough com-

NCD in 5 lines of Julia code

```
using Libz
compress(str) = readbytes(ZlibDeflateInputStream(takebuf_array(IOBuffer(str))))
C(str) = length(compress(str))
lexorder(strs) = join(sort(strs), "")
ncd(x, y, c = C) = ( c(lexorder([x, y])) - min(c(x), c(y)) ) / max(c(x), c(y))
```

NCDm would be another ~15 lines to do the looping!

Searching for (Test) Diversity

Robert Feldt, Simon Poulding



DEPARTMENT
OF SOFTWARE
ENGINEERING

Searching for test data with feature diversity

Robert Feldt and Simon Poulding

Chalmers University of Technology and Blekinge Institute of Technology
`robert.feldt@chalmers.se`, `robert.feldt@bth.se`,
WWW home page: <http://www.robertfeldt.net>

Abstract. There is an implicit assumption in software testing that more diverse and varied test data is needed for effective testing and to achieve different types and levels of coverage. Generic approaches based on information theory to measure and thus, implicitly, to create diverse data have also been proposed. However, if the tester is able to identify features of the test data that are important for the particular domain or context in which the testing is being performed, the use of generic diversity measures such as this may not be sufficient nor efficient for creating test inputs that show diversity in terms of these features. Here we investigate different approaches to find data that are diverse according to a specific set of features, such as length, depth of recursion etc. Even though these features will be less general than measures based on information theory, their use may provide a tester with more direct control over the type of

<https://arxiv.org/abs/1709.06017>

Actually, our current paper (in submission) is not yet on arXiv, it has more experiments and is the one I'll use here...

Searching for Test Data with Diverse Feature Values

Anon Anonymous¹, AA2, AA3, and AA4²

Abstract—There is an implicit assumption in software testing that more diverse and varied test data is needed for effective testing and to achieve different types and levels of coverage. Generic approaches based on information theory to measure and thus, implicitly, to create diverse data have also been proposed. However, often a tester knows of or can identify specific features of the test data that are important for the domain or context where the testing is being performed. In such cases, the use of generic diversity measures may not be sufficient nor efficient. Here we investigate different approaches to find data that are diverse according to a specific set of features. Even though these features will be less general than diversity measures based on information theory, their use may provide a tester with more direct control over the type of test data diversity. Our experiment evaluates seven different feature-specific diversification methods on 10 different test data generation problems and feature spaces. Our results show that diversification search based on evolutionary strategies can be efficient and robust to variation in the generation task. However, methods based on random resampling are not far behind in terms of diversity and can offer other benefits.

Index Terms—Keywords: Software Testing, Diversity, Search-based Software Testing, Empirical Study

inputs that are in a part... d for
which the count of nume... range
(feature 2). Maybe the t... long
numeric inputs there are... good
and varied testing, we s... s, but
that are spread within the... es, as
opposed to diverse string... mber
of numeric characters. T... small
test inputs so we want to... area
in the space spanned by the features.

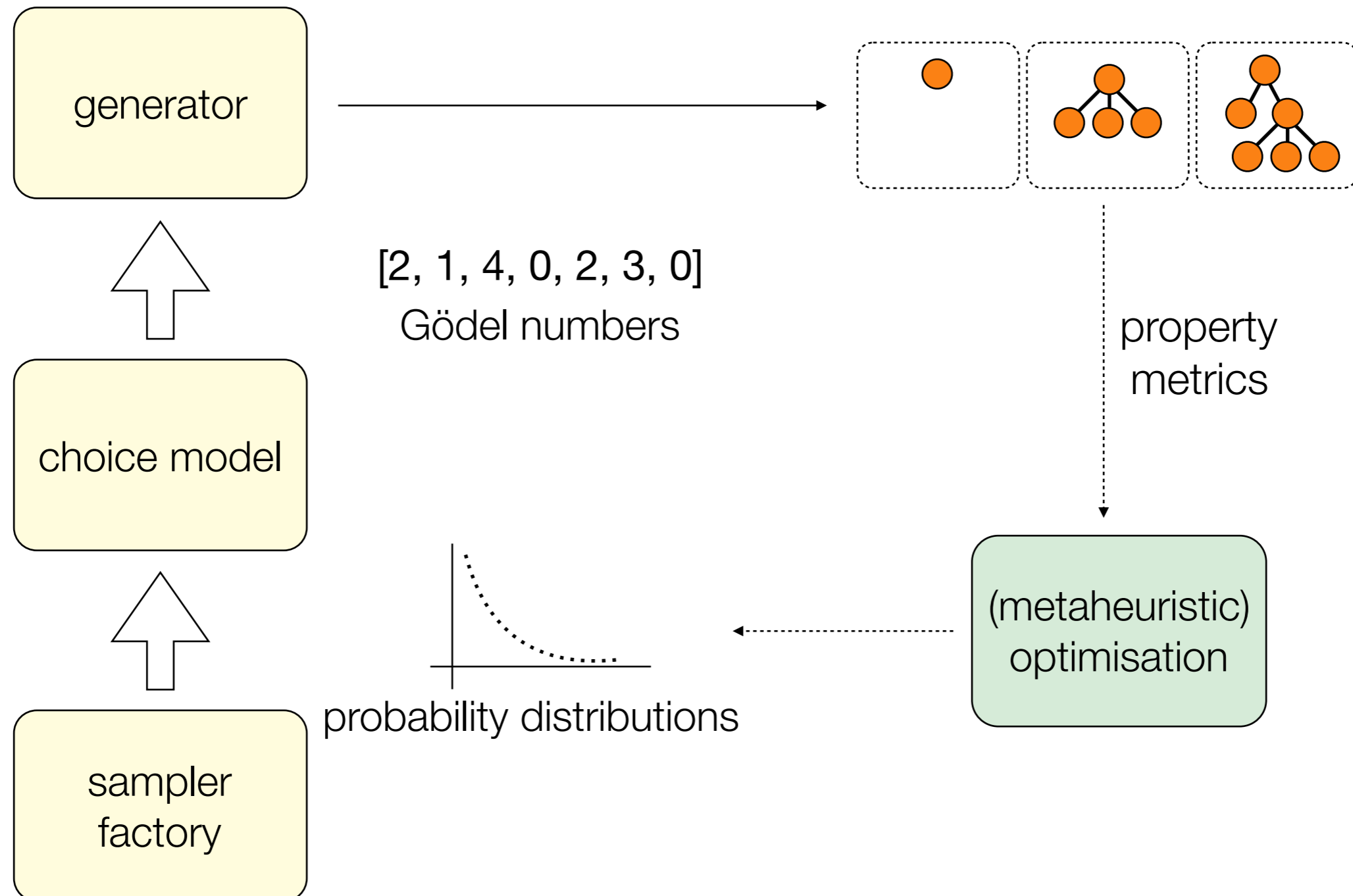
Here we target that specific form of test diversity (TD) that we call the *Feature-Specific TD* problem: how to sample as diverse and complete set of test inputs as possible in a specific area of a specified feature space? This problem is in contrast to the *General TD* problem where we seek diversity in general without requiring diversity within in a particular set of features.

These approaches are likely to be complementary. General TD allows testers to generate data regardless of specific features, whereas Feature-Specific TD allows testers to control



GödelTest Framework

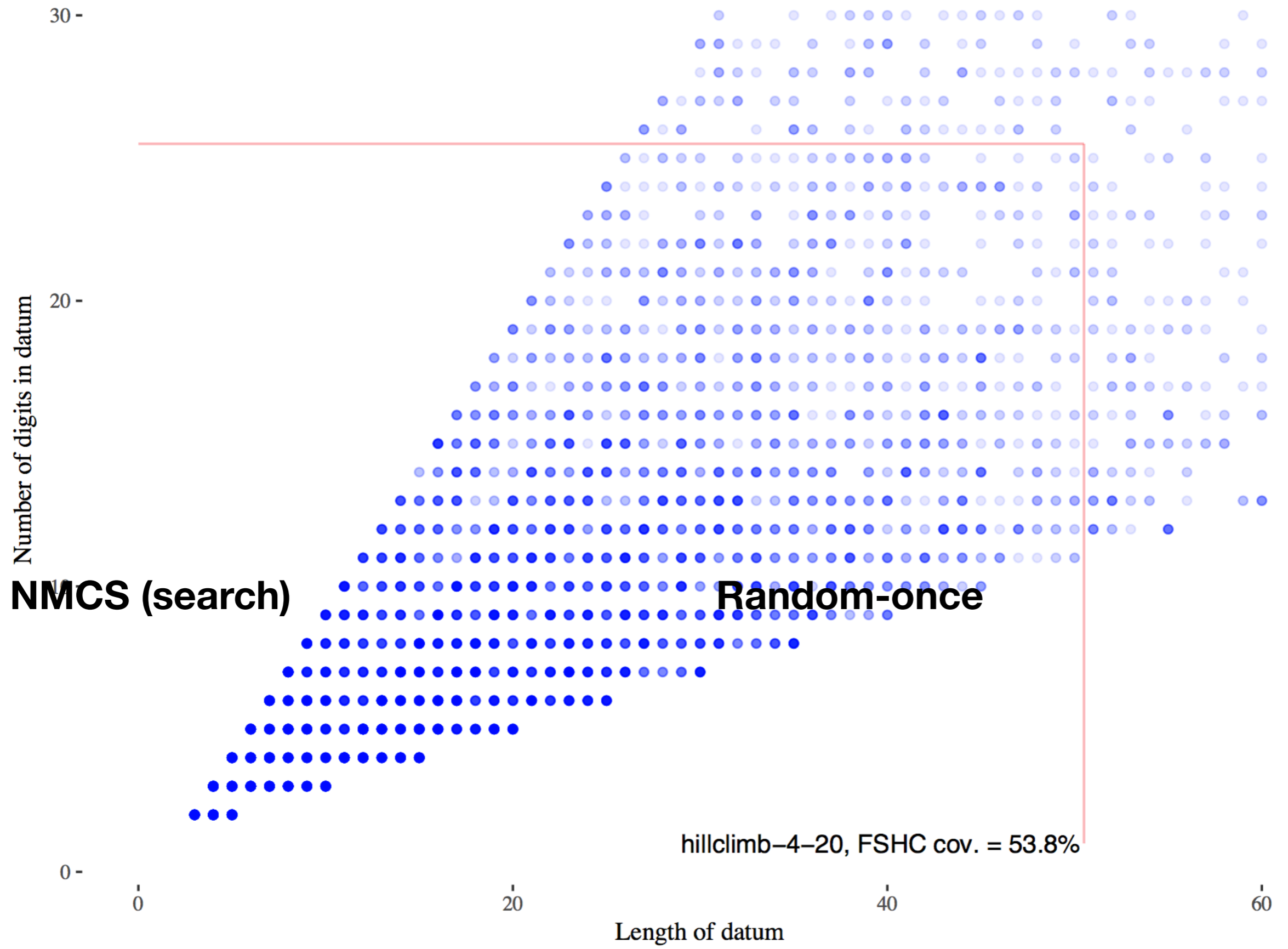
Extracts a model of choice points from a non-deterministic generator; optimises the choice model using optimisation to meet specific objectives

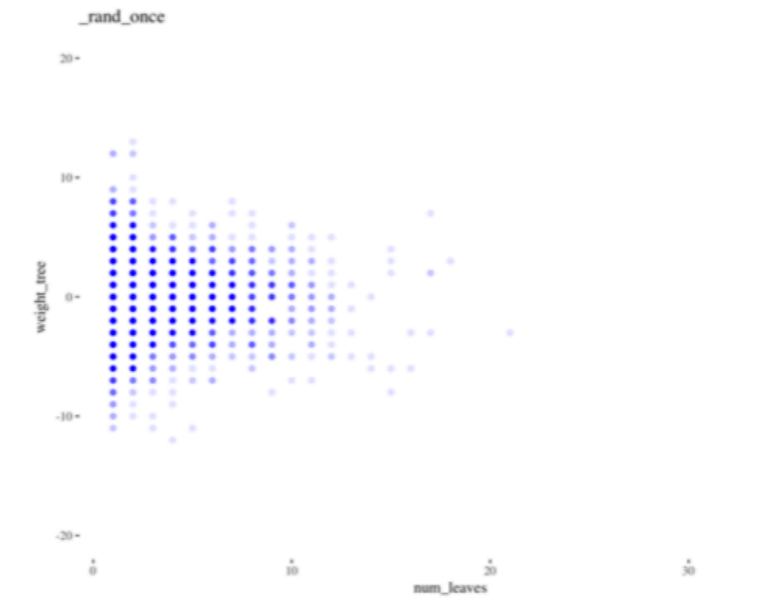
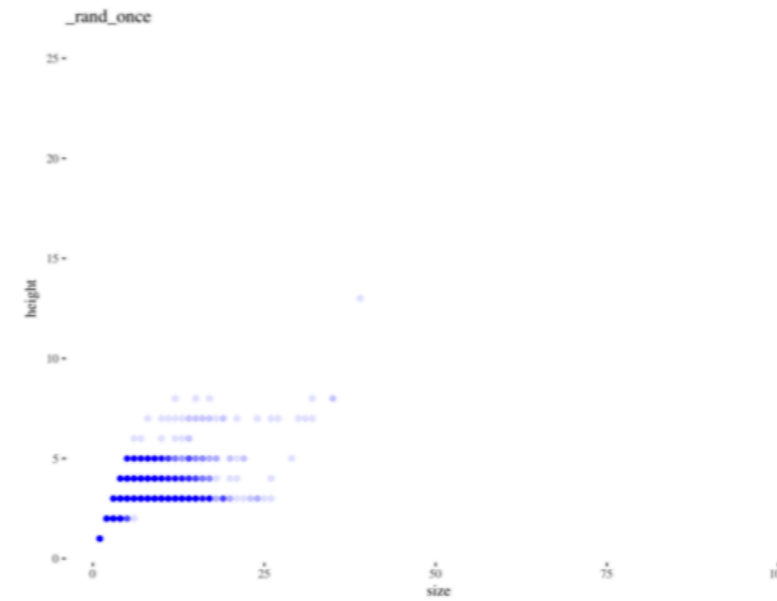
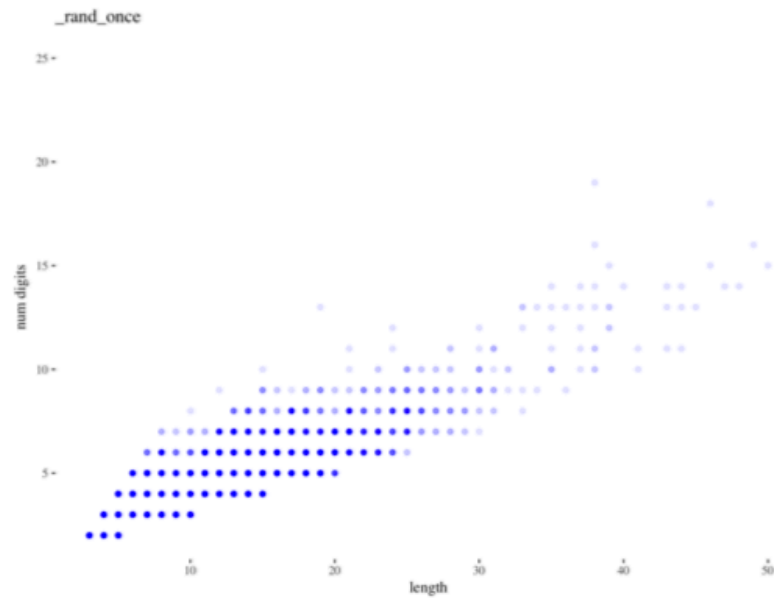


A simple expression generator (for testing calculators)

```
@generator ExprGen begin
  start() = expression()
  expression() = operand() * operator() * operand()
  operand() = "(" * expression() * ")"
  operand() = (choose(Bool) ? "-" : "") *
              join(plus(digit))
  digit() = choose(Int, 0, 9)
  operator() = "+"
  operator() = "-"
  operator() = "/"
  operator() = "*"
end
```

Hillclimb (search)

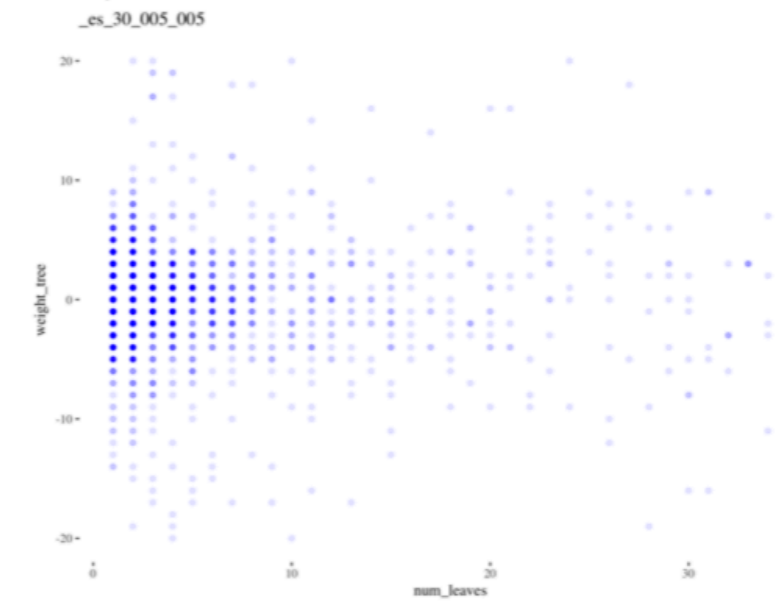
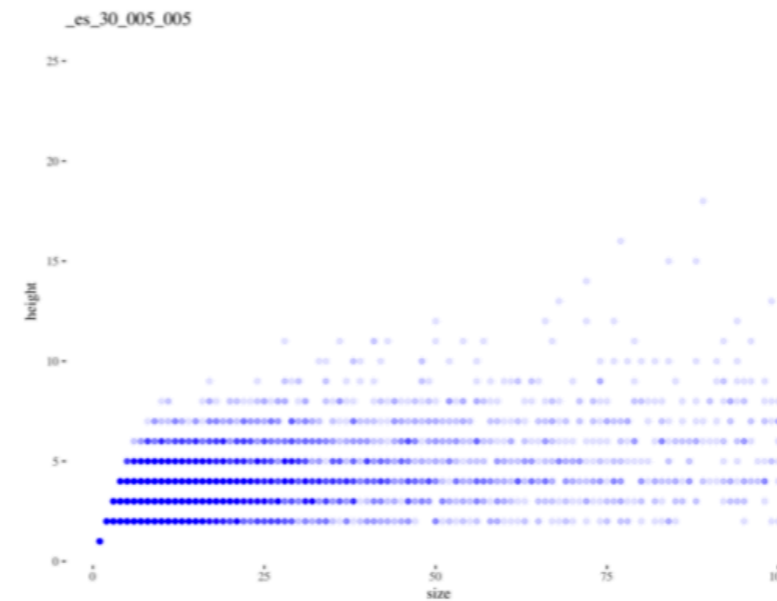




(a) Length (x-axis) and number of digits (y-axis) for the arithmetic expression input domain, using a random method.

(b) Tree size (x-axis) and height (y-axis) for the general tree input domain, using a random method.

(c) Number of leaves (x-axis) and the tree balance (y-axis) for the BST input domain, using a random method.

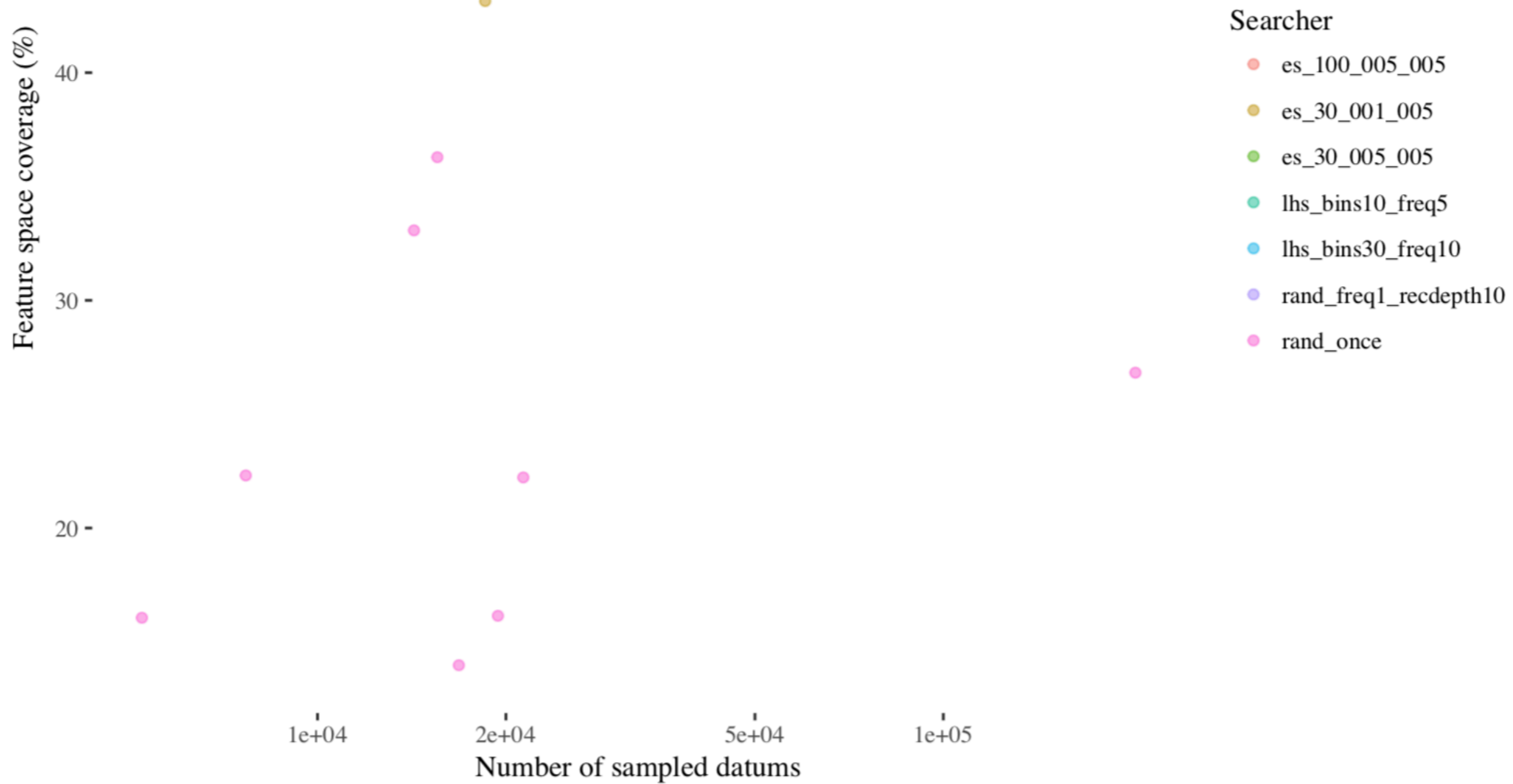


(d) Length (x-axis) and number of digits (y-axis) for the arithmetic expression input domain, using an evolutionary strategy searcher.

(e) Tree size (x-axis) and height (y-axis) for the general tree input domain, using an evolutionary strategy searcher.

(f) Number of leaves (x-axis) and the tree balance (y-axis) for the BST input domain, using an evolutionary strategy searcher.

Fig. 2: Plots of six feature spaces, for our three input domains. Here, we show the different feature values (blue dots) sampled when using different search-based methods, such as Random (2a, 2b and 2c, referred as *RandOnce*) and an Evolutionary Strategy (2d, 2e and 2f, referred as *ES(30)*). The darkness of the dots means that the searcher sampled more data in those respective areas.



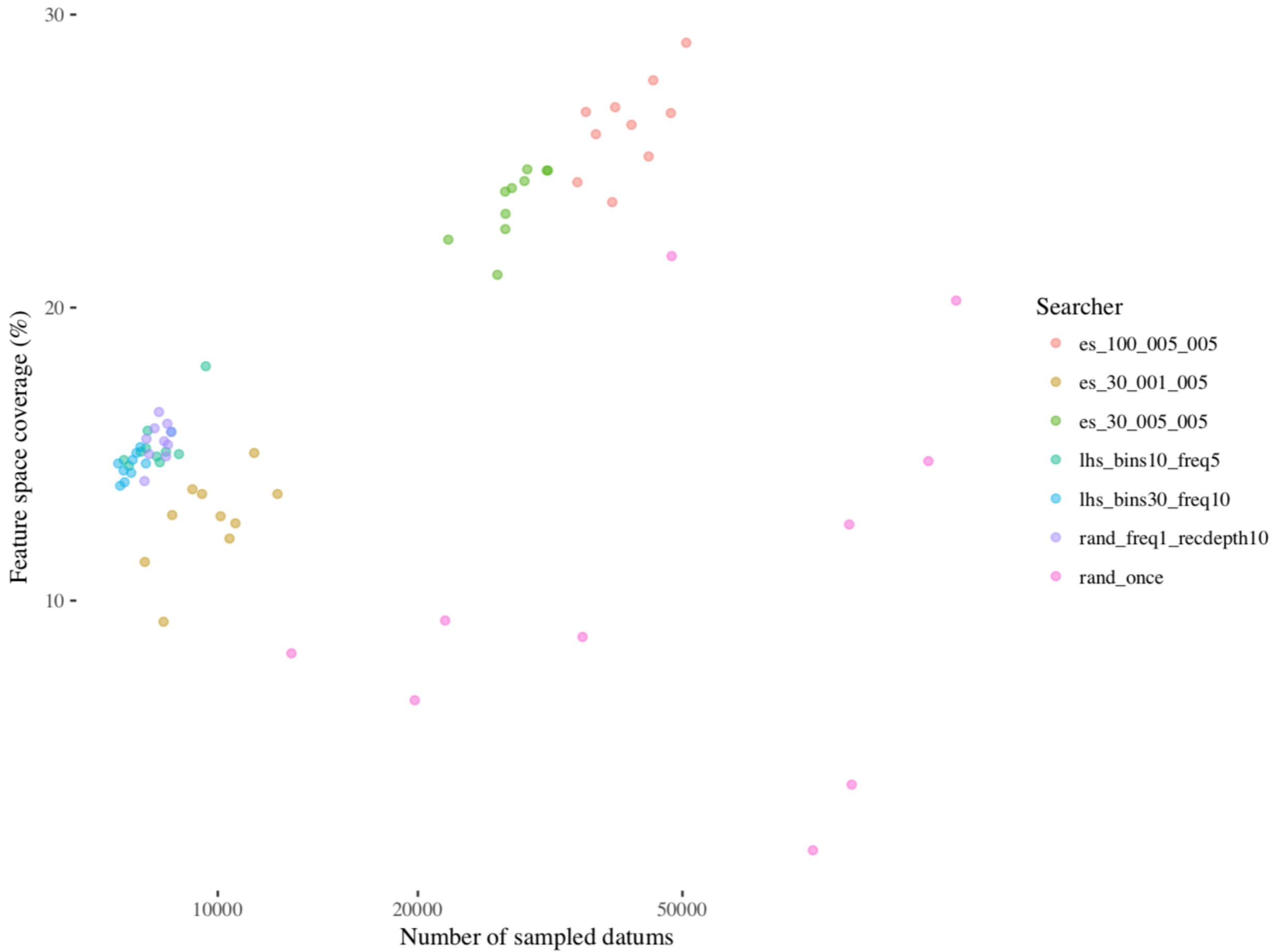


TABLE I: Descriptive statistics on the performance of the 7 investigated diversification methods on the 10 different 2-dimensional feature spaces for 3 different generators when given 60 seconds of search time. Each cell of the table has the average coverage of the preferred feature space (mean FSHC over 10 runs), and the rank of the method (in parentheses). The rightmost column shows the average FSHC for the row and the bottom row shows the average coverage and rank for each method.

Generator	FeatureSpace	ES(30)	ES(100)	LHS(10,5)	LHS(30,10)	RandFreq1	ES(30,0.001)	RandOnce	Avg.
BST	height+balance	81.4 (1)	79.9 (4)	80.3 (2)	80.1 (3)	79.9 (5)	79.0 (6)	64.2 (7)	77.8
BST	leaves+height	67.7 (4)	65.6 (5)	68.7 (1)	68.4 (2)	67.8 (3)	62.8 (6)	51.8 (7)	64.7
BST	leaves+nodes	38.9 (2)	39.5 (1)	36.1 (4)	36.4 (3)	35.3 (5)	33.3 (6)	26.8 (7)	35.2
BST	leaves+balance	94.0 (3)	94.0 (2)	94.5 (1)	93.9 (4)	93.1 (5)	87.9 (6)	85.9 (7)	91.9
BST	size+height	41.7 (1)	40.7 (4)	40.9 (2)	40.9 (3)	40.3 (5)	35.6 (6)	24.0 (7)	37.7
BST	size+balance	40.2 (1)	39.5 (4)	40.1 (2)	39.7 (3)	39.4 (5)	36.5 (6)	28.4 (7)	37.7
Expr	length+digits	55.3 (2)	55.4 (1)	51.1 (3)	51.0 (5)	51.1 (4)	46.1 (6)	28.8 (7)	48.4
Tree	height+leaves	29.5 (2)	31.7 (1)	26.0 (3)	25.5 (5)	26.0 (4)	18.4 (7)	19.2 (6)	25.2
Tree	size+height	23.6 (2)	26.2 (1)	15.3 (4)	14.7 (5)	15.4 (3)	12.7 (6)	10.7 (7)	16.9
Tree	size+leaves	12.8 (2)	13.8 (1)	9.8 (3)	9.6 (4)	9.6 (5)	7.4 (6)	5.6 (7)	9.8
		48.5 (2.0)	48.6 (2.4)	46.3 (2.5)	46.0 (3.7)	45.8 (4.4)	42.0 (6.1)	34.5 (6.9)	

Diversity to Guide Robustness Testing

Robert Feldt, Simon Poulding



DEPARTMENT
OF SOFTWARE
ENGINEERING

Generating Controllably Invalid and Atypical Inputs for Robustness Testing

Simon Poulding

Software Engineering Research Lab
Blekinge Institute of Technology
37179 Karlskrona, Sweden
Email: simon.poulding@bth.se

Robert Feldt

Software Engineering Research Lab
Blekinge Institute of Technology
37179 Karlskrona, Sweden
Email: robert.feldt@bth.se

Abstract—One form of robustness in a software system is its ability to handle, in an appropriate manner, inputs that are unexpected compared to those it would experience in normal operation. In this paper we investigate a generic approach to generating such unexpected test inputs by extending a framework

In previous work we have described GödelTest, a framework for generating complex, highly-structured test data [2]. A key feature of GödelTest is a clear separation between *generator* code that defines how to build a test input and a *choice model* that controls choices that can be made during the generation

<http://ieeexplore.ieee.org/document/7899038/>

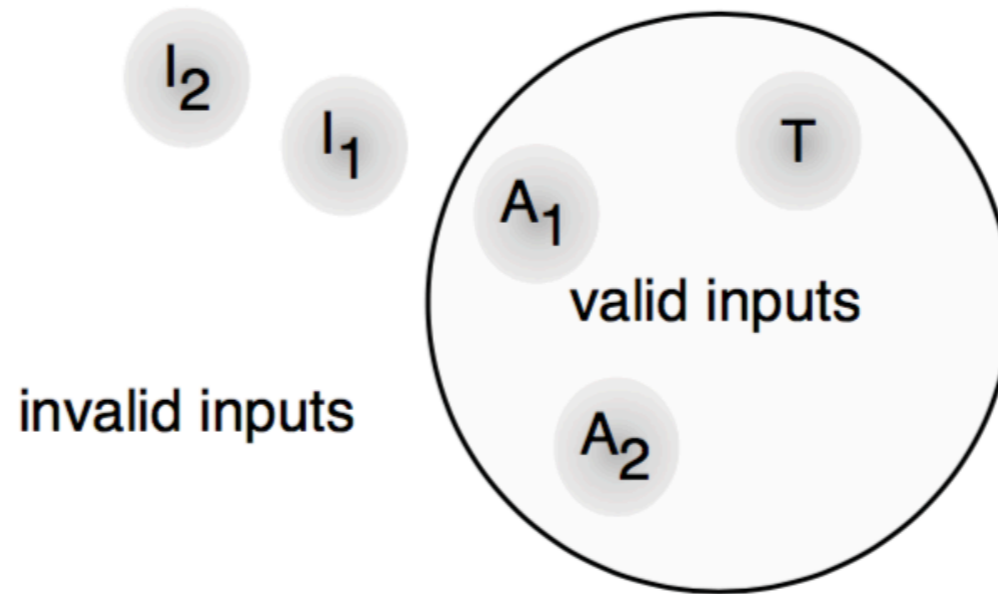


Fig. 2. The intended relationship between the typical (T), atypical (A), and invalid (I) test set categories.

TABLE I
 BYTES OF COMPRESSED WARNING AND ERROR MESSAGES PER
 CHARACTER OF TEST INPUT FOR THE TEST SET CATEGORIES.

Test Set Category	Compressed Message Bytes per Test Input Character	Hypothesis Test	
		vs.	p -value
T	8.59×10^{-4}	—	—
A_1	7.44×10^{-3}	T	$< 10^{-5}$
A_2	9.34×10^{-3}	A_1	1.87×10^{-3}
I_1	1.10×10^{-2}	A_1	$< 10^{-5}$
I_2	1.23×10^{-2}	I_1	3.83×10^{-3}

Finding a boundary between valid and invalid regions of the input space

Bogdan Marculescu*

Robert Feldt*†

*Blekinge Institute of Technology

School of Computing

Karlskrona, Sweden

†Chalmers and the University of Gothenburg

Dept. of Computer Science and Engineering

Gothenburg, Sweden

Abstract—In the context of robustness testing, the boundary between the valid and invalid regions of the input space can be an interesting source of erroneous inputs. Knowing where a specific software under test (SUT) has a boundary is also essential for validation in relation to requirements. However, finding where a SUT actually implements the boundary is a non-trivial problem that has not gotten much attention.

This paper proposes a method of finding the boundary between the valid and invalid regions of the input space, by developing pairs of test sets that describe that boundary in detail.

The proposed method consists of two steps. First, test data generators, directed by a search algorithm to maximise distance to known, valid test cases, generate valid test cases that are closer to the boundary. Second, these valid test cases undergo mutations to try to push them over the boundary and into the invalid part of the input space. This

representation. Different stages of software development have different notions of what the input domain is, and where the boundary between the valid and invalid input spaces lies. This region of the input space is identified as a rich source of software errors.

In this paper we will focus on robustness, as a non-functional quality criterion. We will use the definition of robustness proposed by Aviženis [2]: “dependability with respect to erroneous inputs”, a deeper discussion of robustness testing can be found in [3]. In line with Ammann and Offutt [1], we argue that the boundary between the valid and invalid input spaces is a rich source of inputs that could be considered valid at one stage of development, e.g. specification, but invalid in the implementation and thus could be considered erroneous

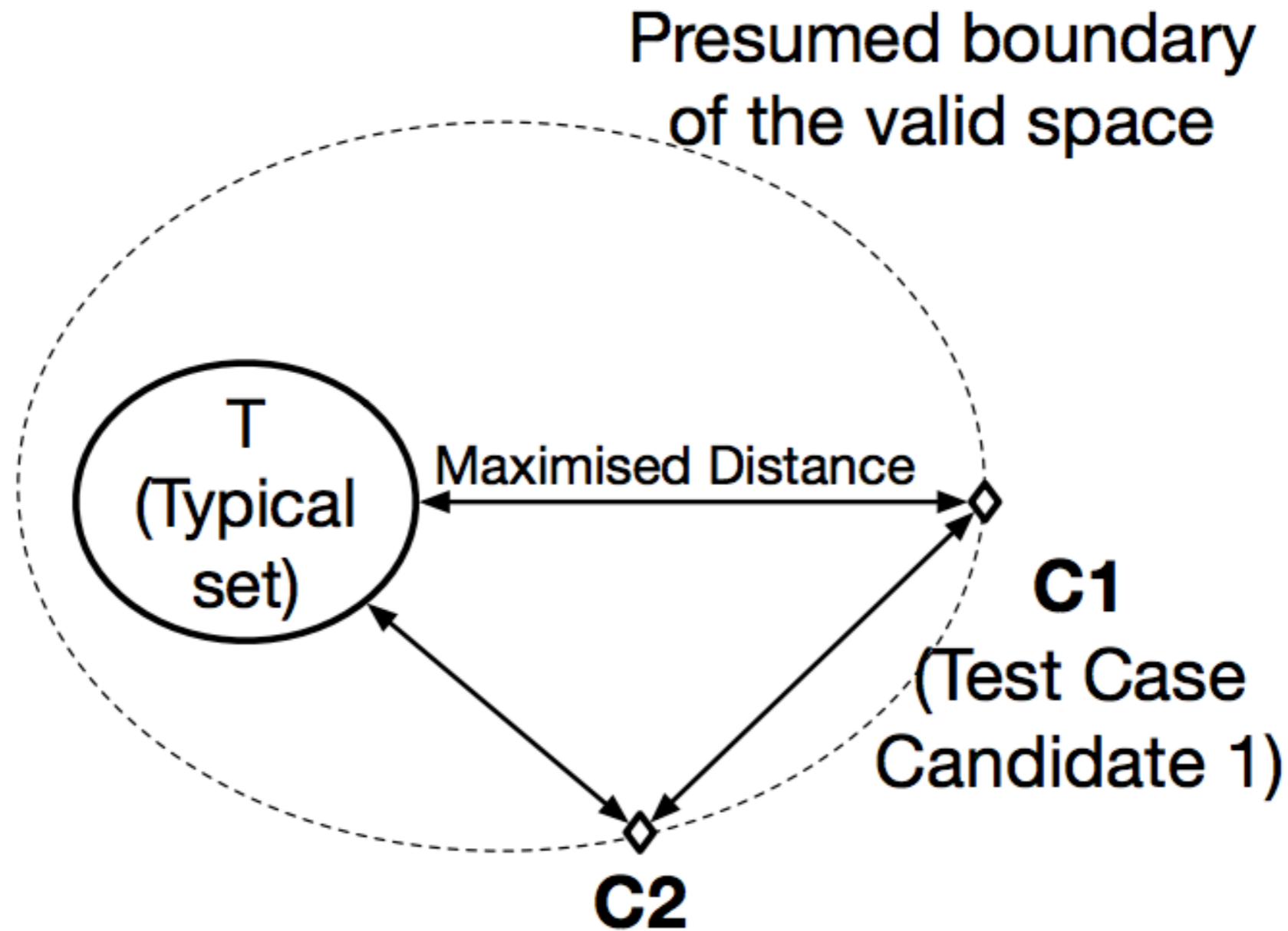


Fig. 2. Step 1: Finding the inner boundary of the valid set, as defined by the generator.

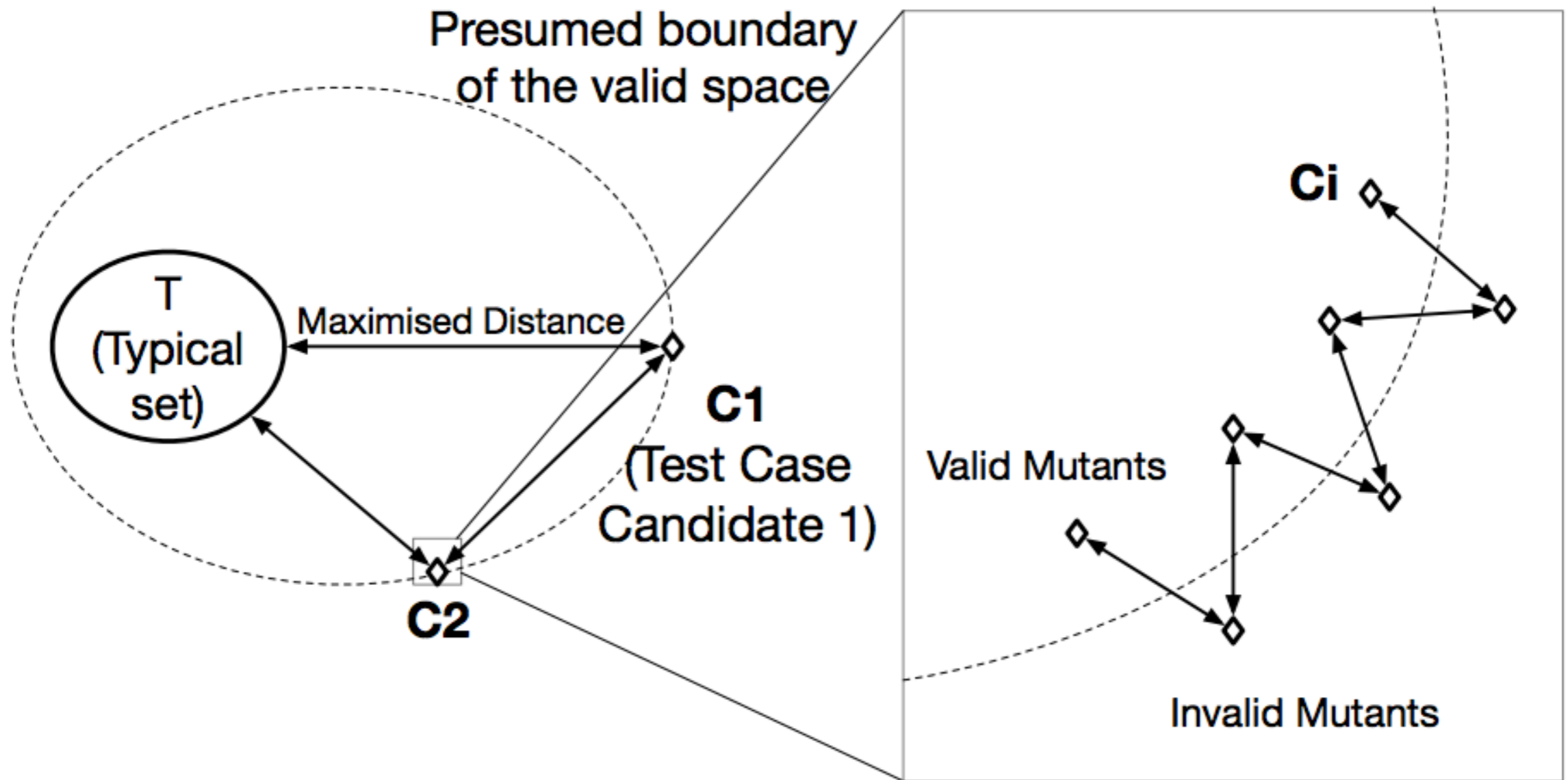


Fig. 3. Step 2: Oscillating between valid and invalid candidates that can be obtained through one basic mutation step. The boundary between the valid and invalid spaces, in this region of the input space, can be assumed to be between the two sets of candidates.

Visualizing test diversity to support test optimisation

Francisco G. de Oliveira Neto, Robert Feldt, Linda Erlenhov
Dept. of Computer Science and Engineering
Chalmers | University of Gothenburg
Gothenburg, Sweden
{gomesf,robert.feldt,linda.erlenhov}@chalmers.se

Jose Benardi de Souza Nunes
Department of Computing Systems
Federal University of Campina Grande
Campina Grande, Brazil
jose.nunes@computacao.ufcg.edu.br

Abstract—Diversity has been used as an effective criteria to optimise test suites for cost-effective testing. Particularly, diversity-based (alternatively referred to as similarity-based) techniques have the benefit of being generic and applicable across different Systems Under Test (SUT), and have been used to automatically select or prioritise large sets of test cases. However, it is a challenge to feedback diversity information to developers and testers since results are typically many-dimensional. Furthermore, the generality of diversity-based approaches makes it harder to choose when and where to apply them. In this paper we address these challenges by investigating: i) what are the trade-off in using different sources of diversity (e.g., diversity of test requirements or test scripts) to optimise large test suites, and ii) how visualisation of test diversity data can assist testers for test optimisation and improvement. We perform a case study on three industrial projects and present quantitative results on the fault detection capabilities and redundancy levels of different sets of test cases. Our key result is that test similarity maps, based on pair-wise diversity calculations, helped industrial practitioners identify issues with their test repositories and decide on actions to improve. We conclude that the visualisation of diversity information can assist testers in their maintenance and optimisation activities.

Index Terms—Keywords: Software Testing, Diversity, Search-based Software Testing, Empirical Study

Automated diversity-based test optimisation techniques calculate distance values and choosing from (dis)similar tests. Even though there are some proposals of methods that calculate diversity for whole sets of tests at once [4] the vast majority of approaches is based on pair-wise calculations. Not only does this lead to performance challenges, due to the $\mathcal{O}(n^2)$ execution cost, it also makes diversity information hard to visualise and thus difficult to use. Thus it is hard to use diversity information to improve test scenarios with a relatively small number of tests. For example, 10,000 diversity values can be represented as a 100-dimensional ¹ vector.

When testers and developers are faced with a large process or digest the information, it is difficult to get acted upon or have it been found for debugging. If developers could not see the results had been reached, then selection techniques are not the original set being selected.



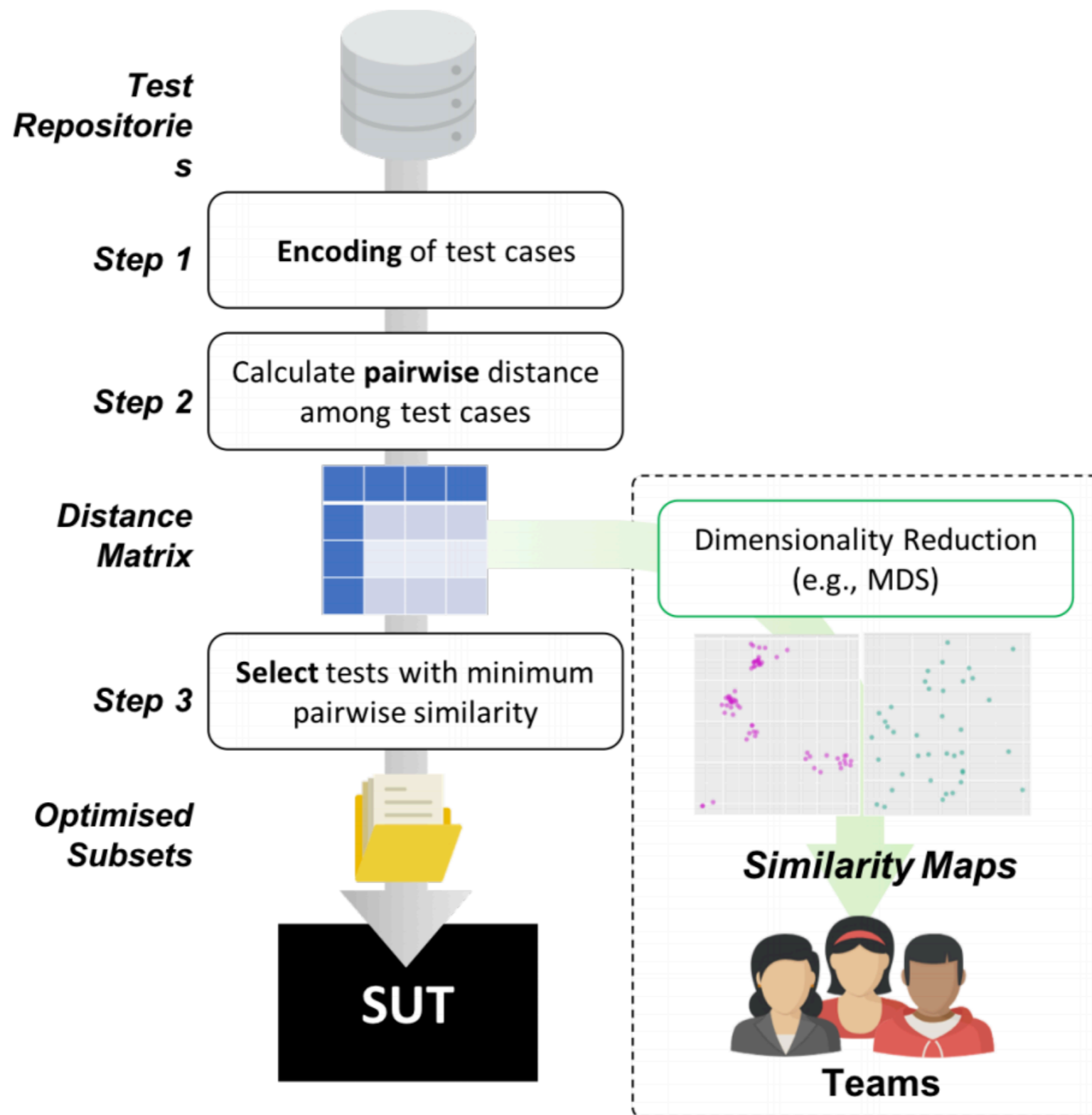


Fig. 1. The general steps for diversity-based test optimisation. Our paper proposes the creation of similarity maps and presentation of the diversity information to stakeholders.

https://hub.docker.com/r/robertfeldt/mdist/



Search

Explore Help

Sign up

Sign In

PUBLIC REPOSITORY

robertfeldt/mdist

Last pushed: 3 months ago

Repo Info

Tags

Short Description

Calculate distances and similarities between files.

Full Description

Simple command line interface to calculate distances between files. Include a large number of distance functions, both classical ones (Levenshtein, Q-Grams etc) and compression based ones (NCD based on different compressors).

This is a command line interface to the [MultiDistances.jl](#) julia library.

Docker Pull Command

```
docker pull robertfeldt/mdist
```

Owner



robertfeldt

`tc1 = "a b c d"`
`tc2 = "a b c e"`
`tc3 = "h i j k l m n"`
`tc4 = "o p q r s"`
`tc5 = "w x y z"`
`tc6 = "a b y z"`

	t1	t2	t3	t4	t5	t6
t1	0.000	0.333	0.916	0.900	0.888	0.571
t2	0.333	0.000	0.916	0.900	0.888	0.571
t3	0.916	0.916	0.000	0.923	0.916	0.916
t4	0.900	0.900	0.923	0.000	0.900	0.900
t5	0.888	0.888	0.916	0.900	0.000	0.571
t6	0.571	0.571	0.916	0.900	0.571	0.000

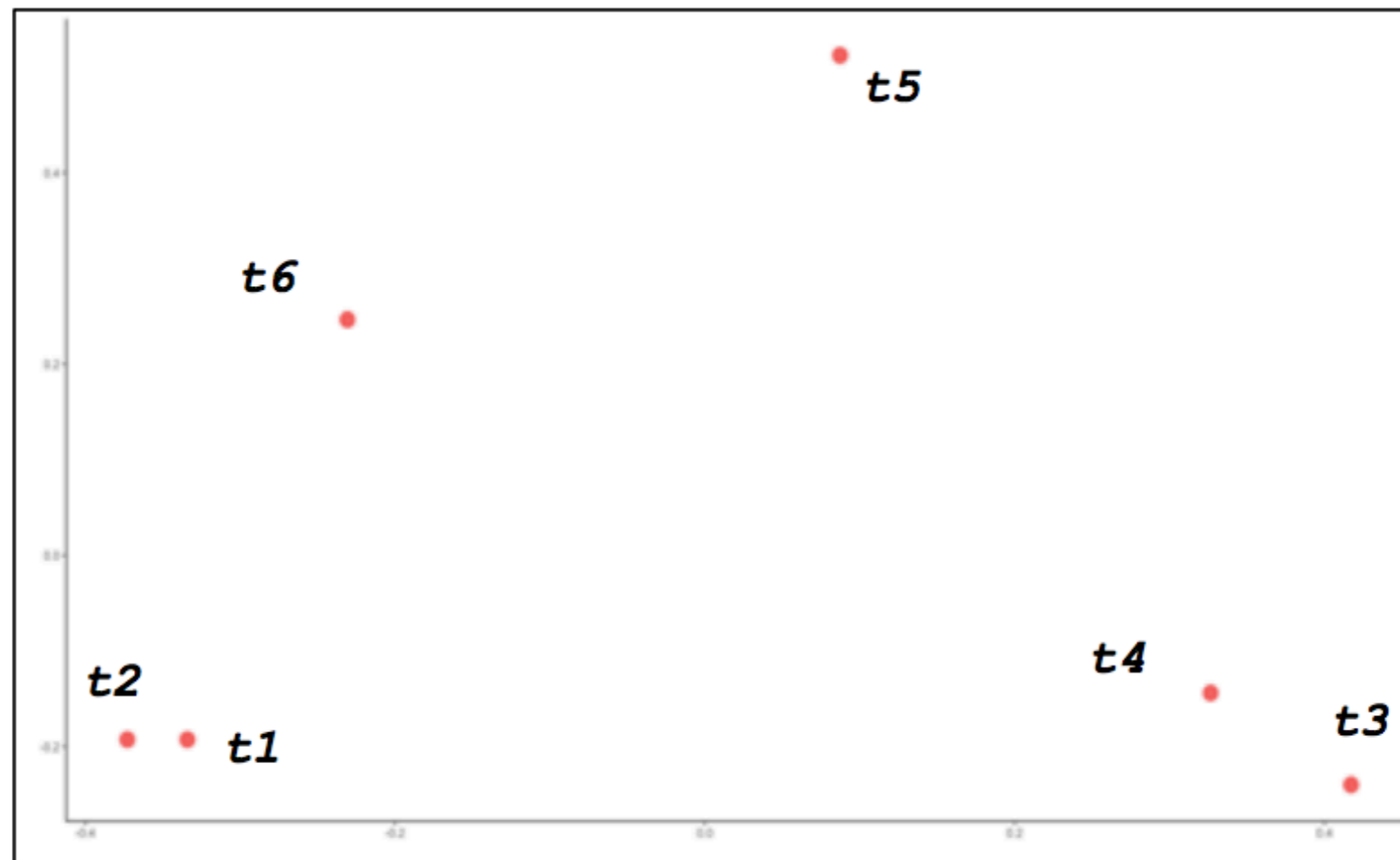


Fig. 2. An example of a similarity map obtained from a toy test suite. We use Jaccard Index based on k-grams of length 5 to calculate the distance matrix. Note that the goal with a similarity map is to observe the relative distances between the tests; the scales on the y-axis and x-axis are thus not important.

SUMMARY
THE PRO
WHERE T
THAT

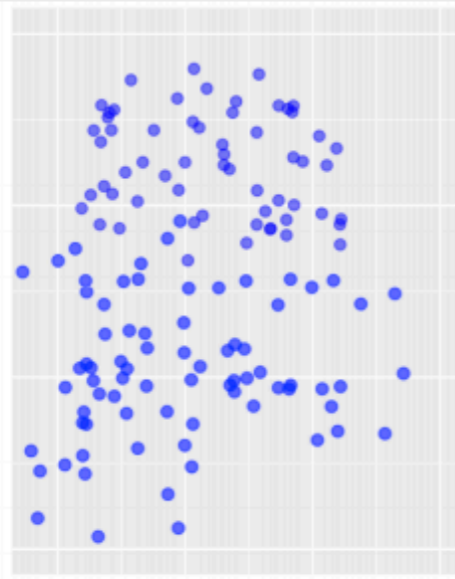
Projects

Project 1

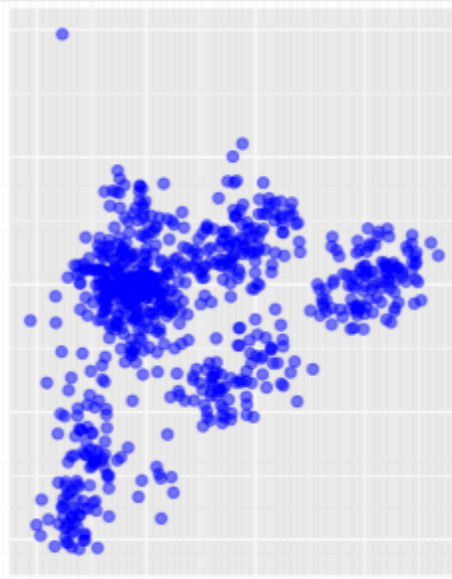
Project 2

Project 3

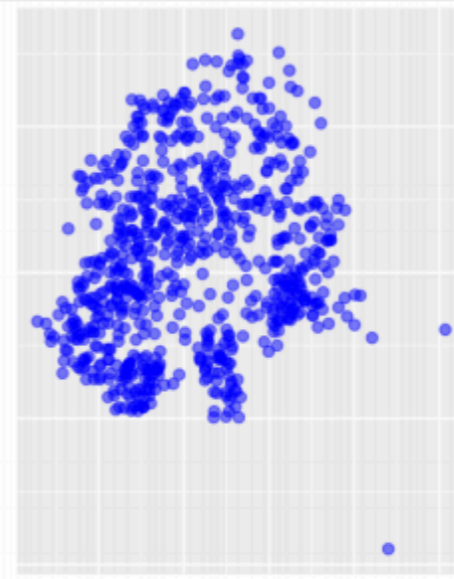
Project 1 - Requirements



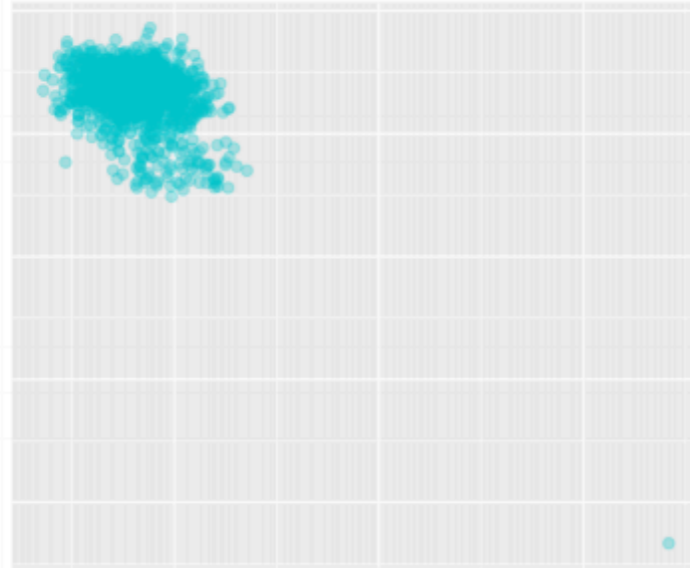
Project 1 - Names



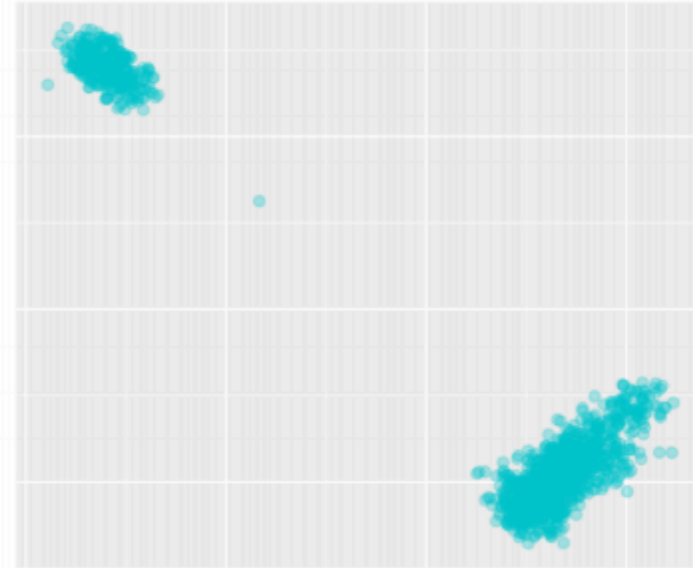
Project 1 - Steps



Project 2 - Names



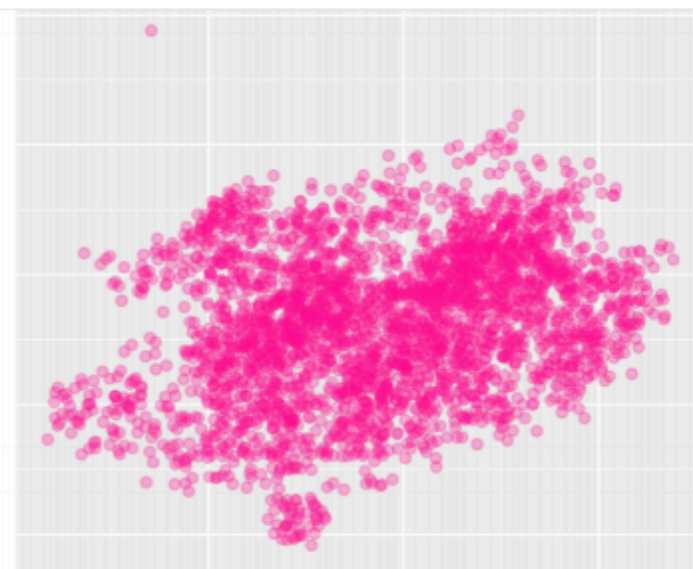
Project 2 - Steps



Project 3 - Requirements



Project 3 - Names



SE STUDY.
ED YEARS
LLY, NOTE
E NOT

t activity

1-2017

2-2017

3-2017

Conclusions

- Information theory can provide
 - theoretically justified metrics for (automated) testing,
 - practically useful (since universal) metrics that work for any data type,
 - new ways to formalise & understand testing problems.
- Coupling these metrics with search is powerful!
- It has helped us formalise, automate, and evaluate:
 - Value of diversity in testing,
 - Robustness testing,
 - (soon to be submitted) Boundary Value testing.
- Focusing on available information also has added value in industry collaborations.

robert.feldt@chalmers.se