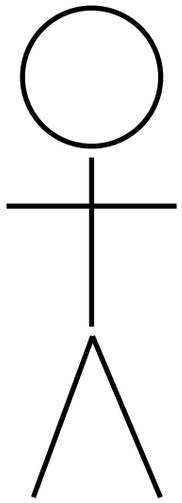
Feature-Specific vs General Diversity: A Tradeoff?

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

CHALMERS

Main message: There is a trade-off between two types of DIVERSITY



NID, NCD

Domain-specific
Feature-specific

General, even Universal

Analysable (theory, math)

Simple & Cheap (to Human)

Costly (to CPU)

Needs more information

Specific, problem adapted

Hard to analyse, no theorems

~Costly (to Human)

Cheap (to CPU)

Lean, directly applicable

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**!

NCD and it's extensions (NCDm) allows us to do this!

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…

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

where C(s) is length of string s after being compressed
with your favourite compressor
(zlib, bzip2, ppm, blosc, lz4, zstandard, …)

NCD in 5 lines of Julia code

```julia
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!

NCDm extension is very useful in testing!

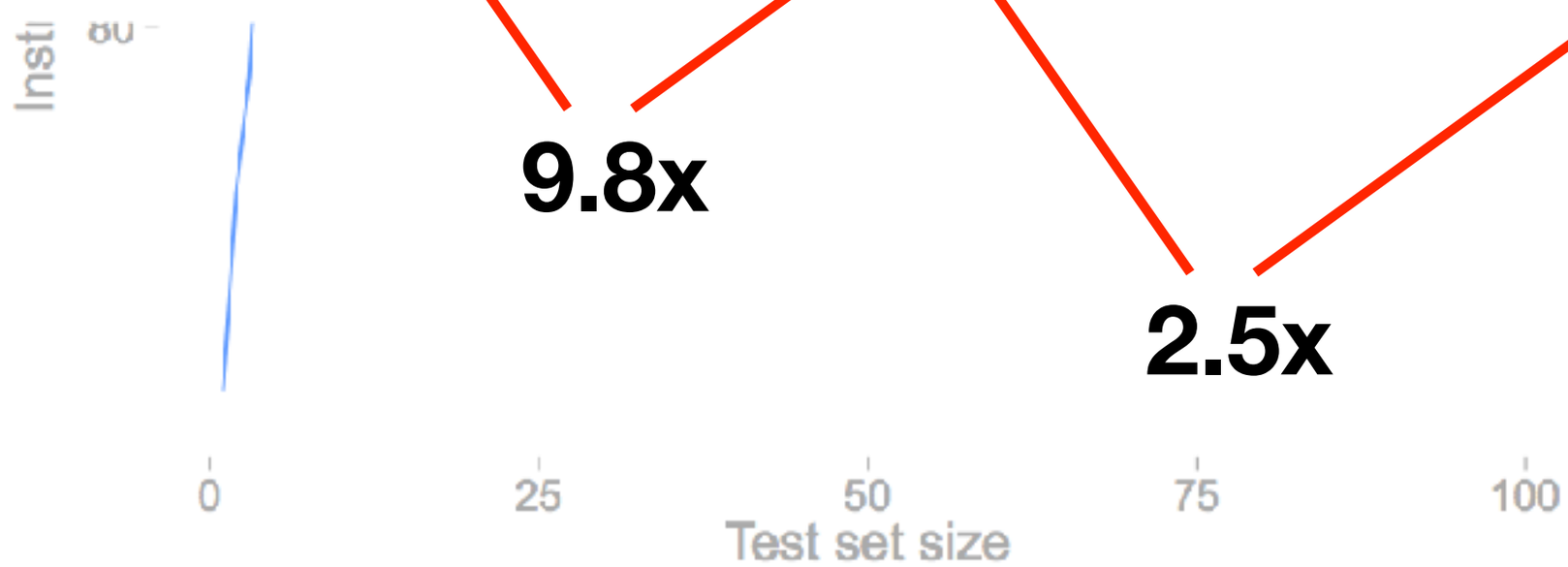
$d($  $) = num$

Test Set Diameter (TSDm):

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

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

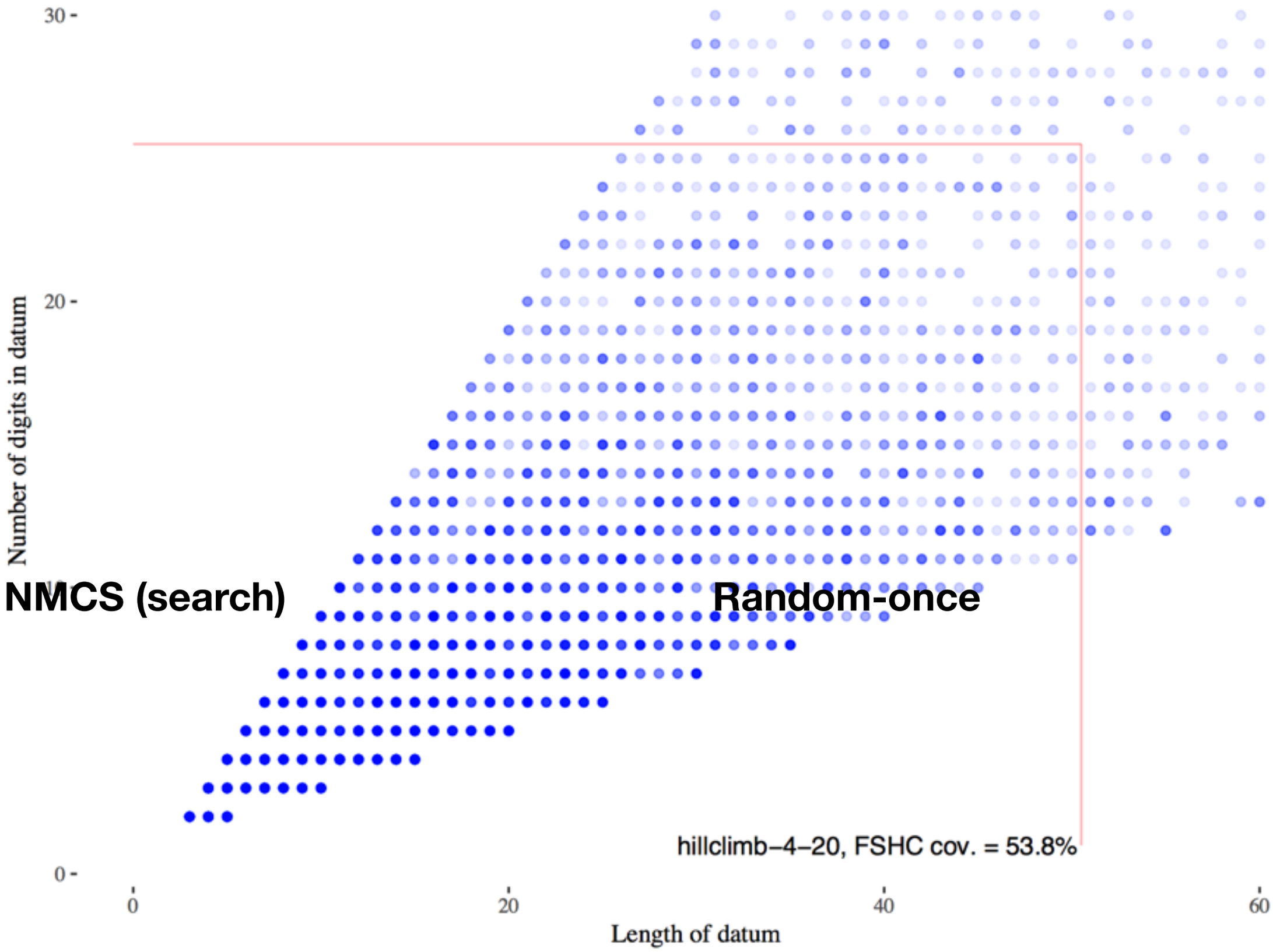
	Avg. Test Set Size					
	I-TSDm			Random		
SUT	90%	95%	99%	90%	95%	99%
JEuclid	29.9	40.9	90.3	82.2	135.3	217.3
NanoXML	1.9	19.4	75.1	18.7	38.2	207.2
ROME	9.1	21.7	51.3	21.9	51.0	129.0



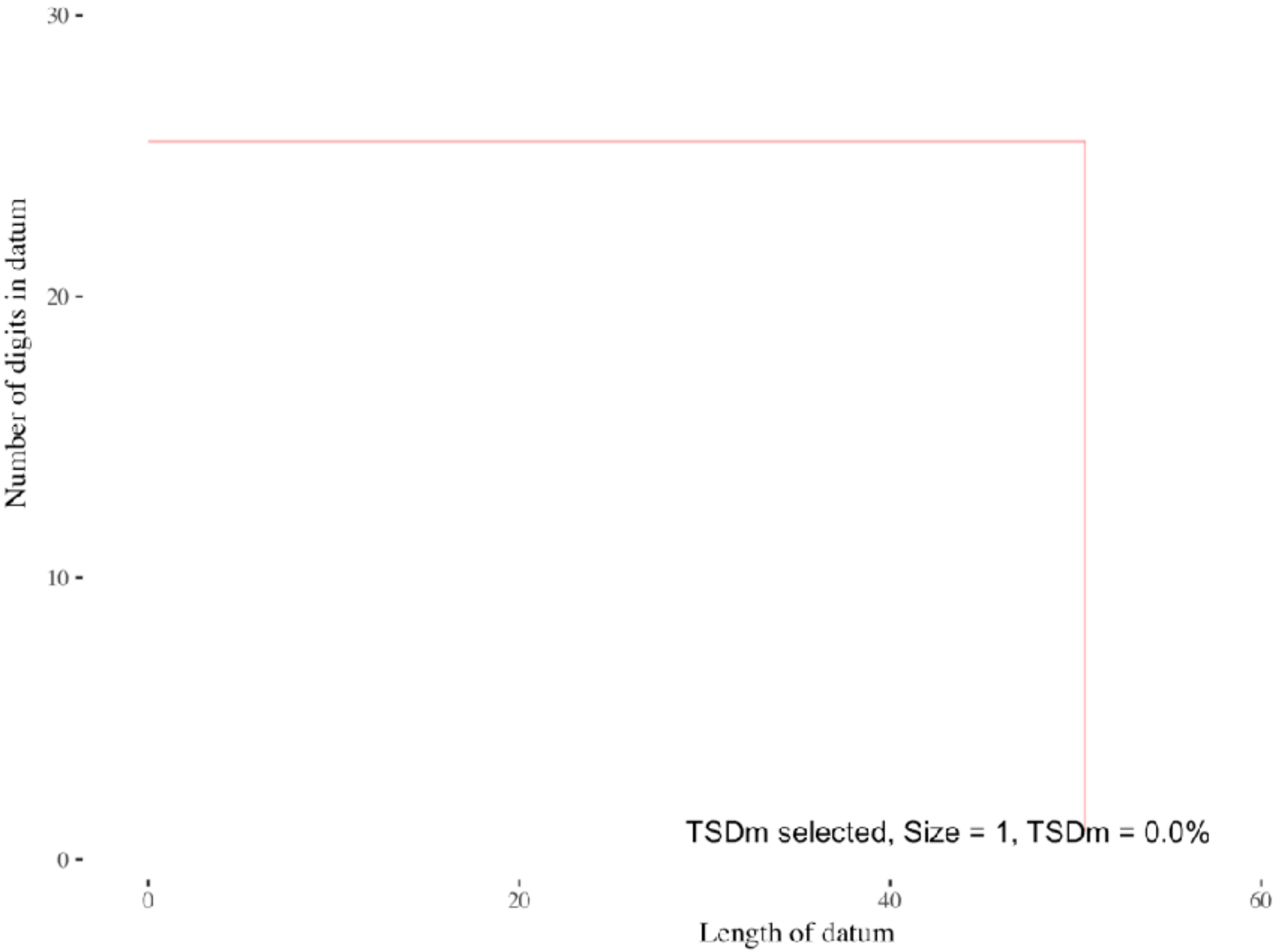
A simple expression generator (for testing calculators)

```
@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
```

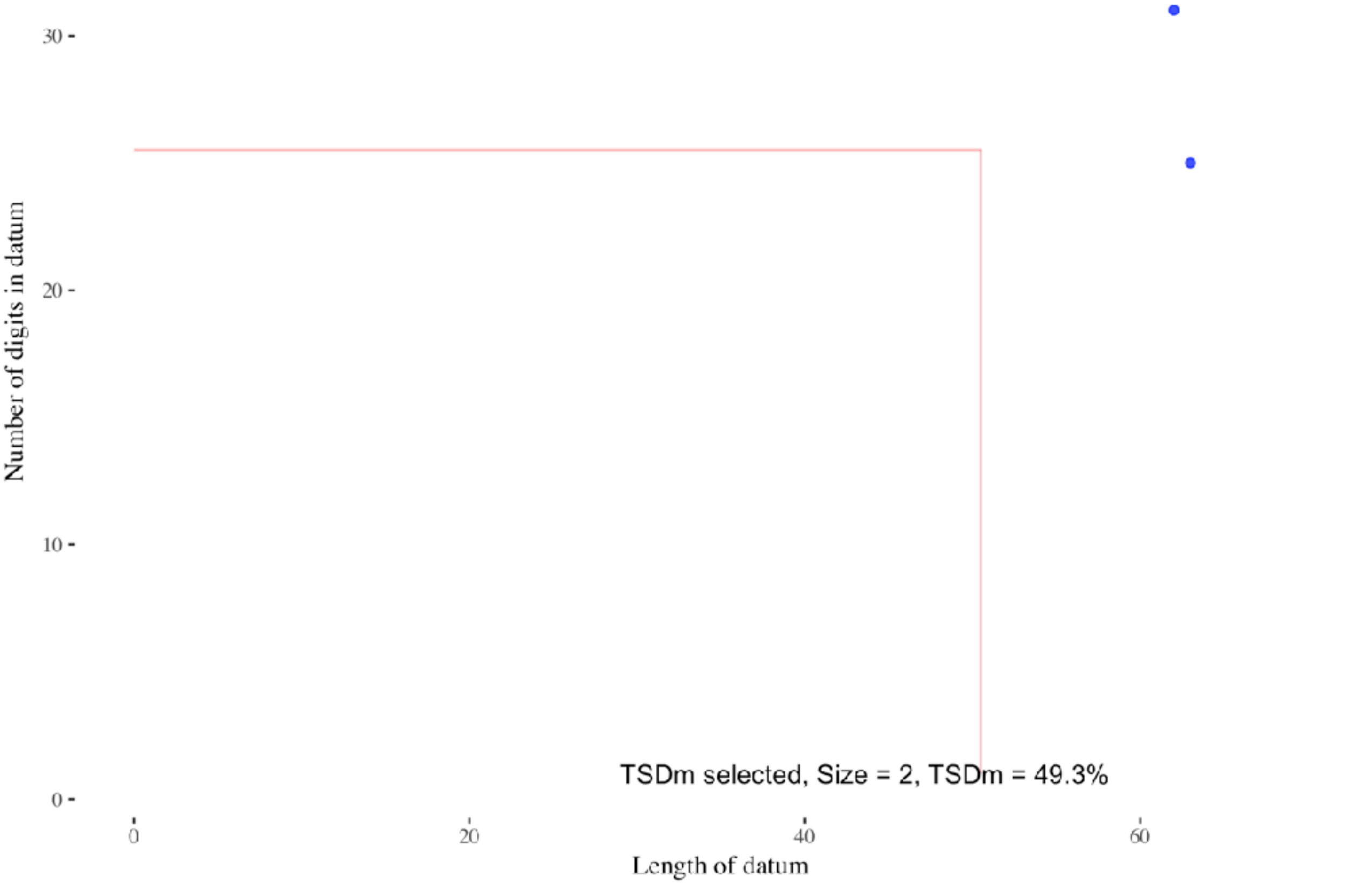
Hillclimb (search)



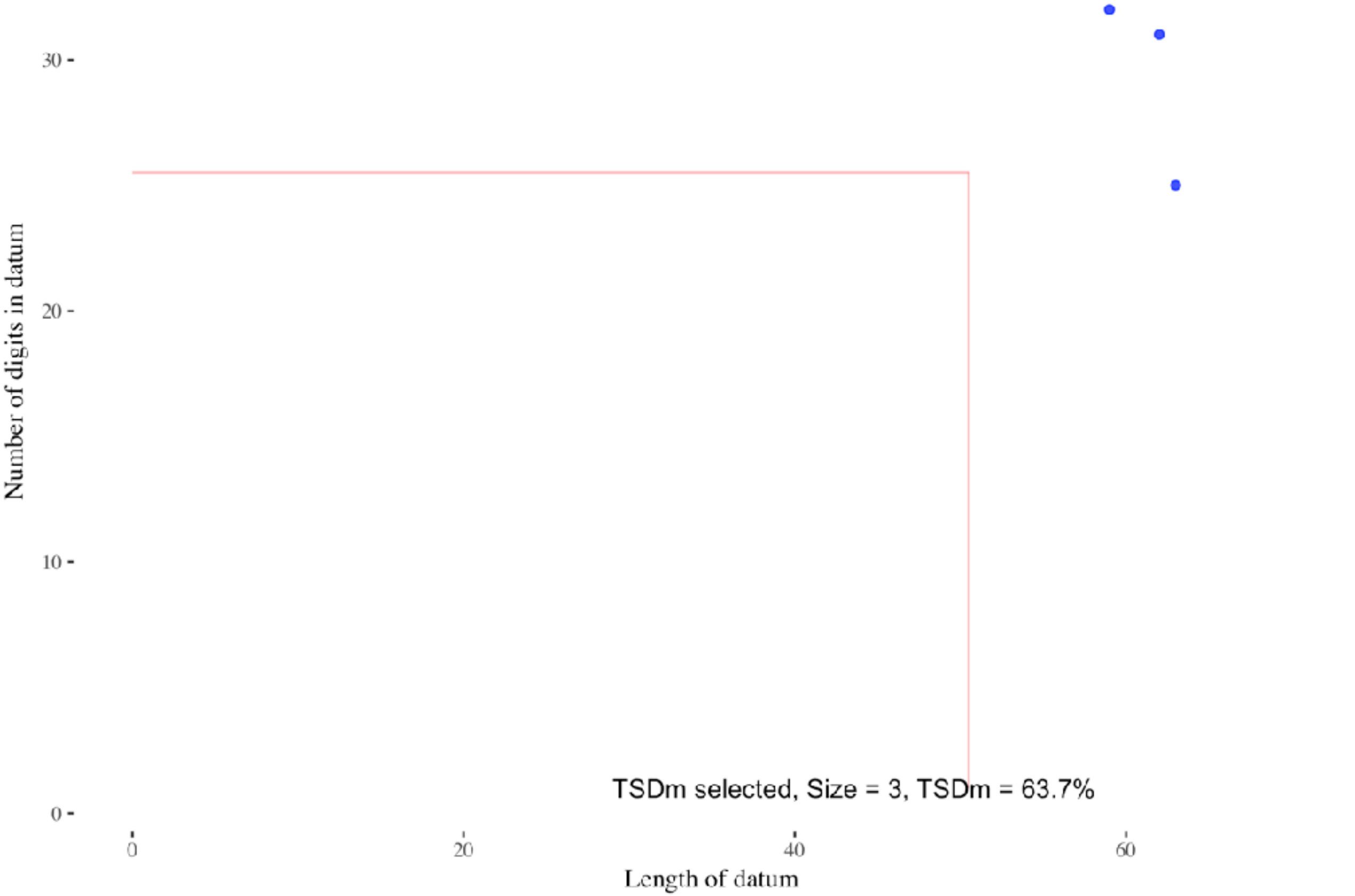
Length vs Num digits



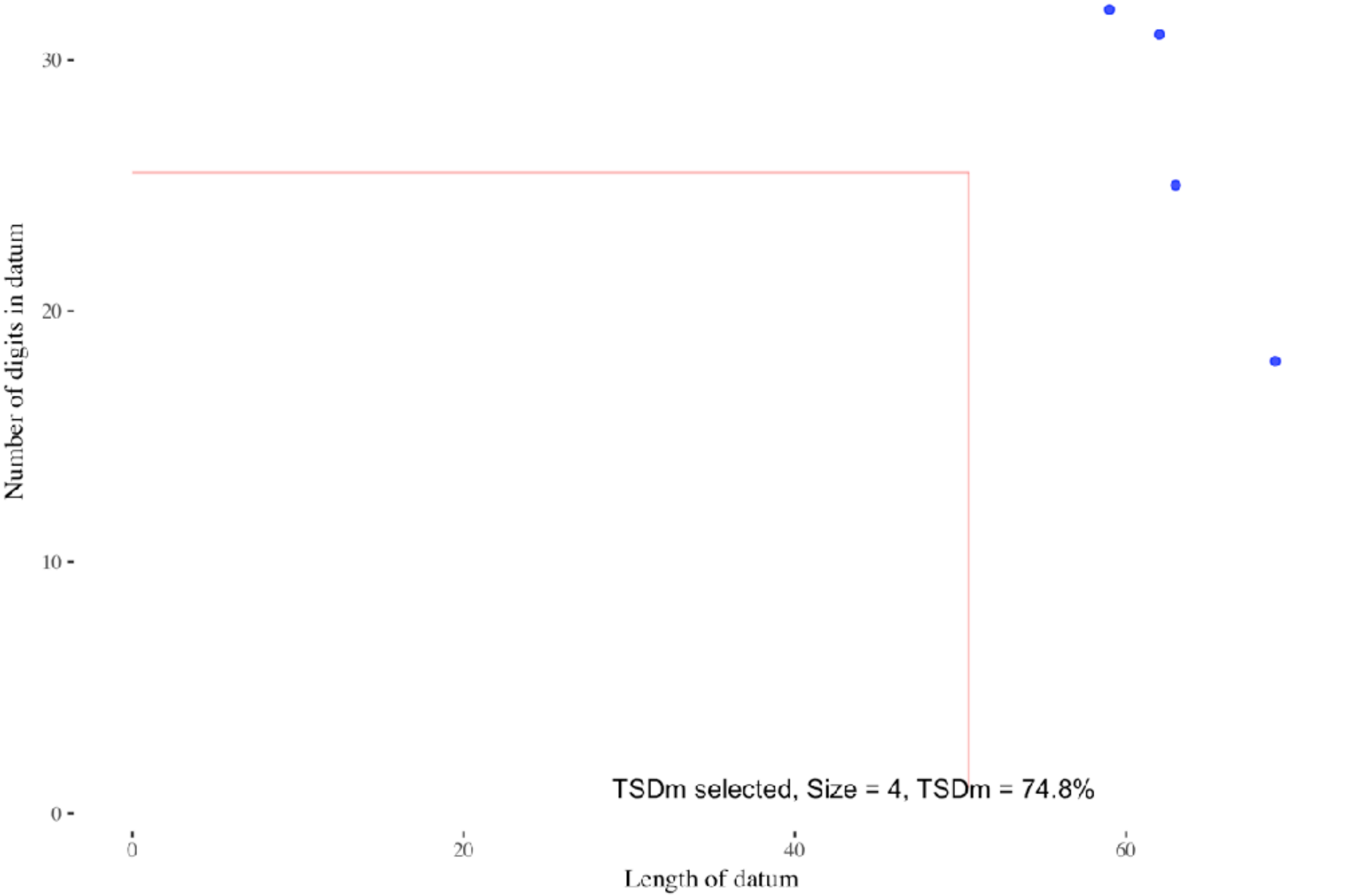
Length vs Num digits



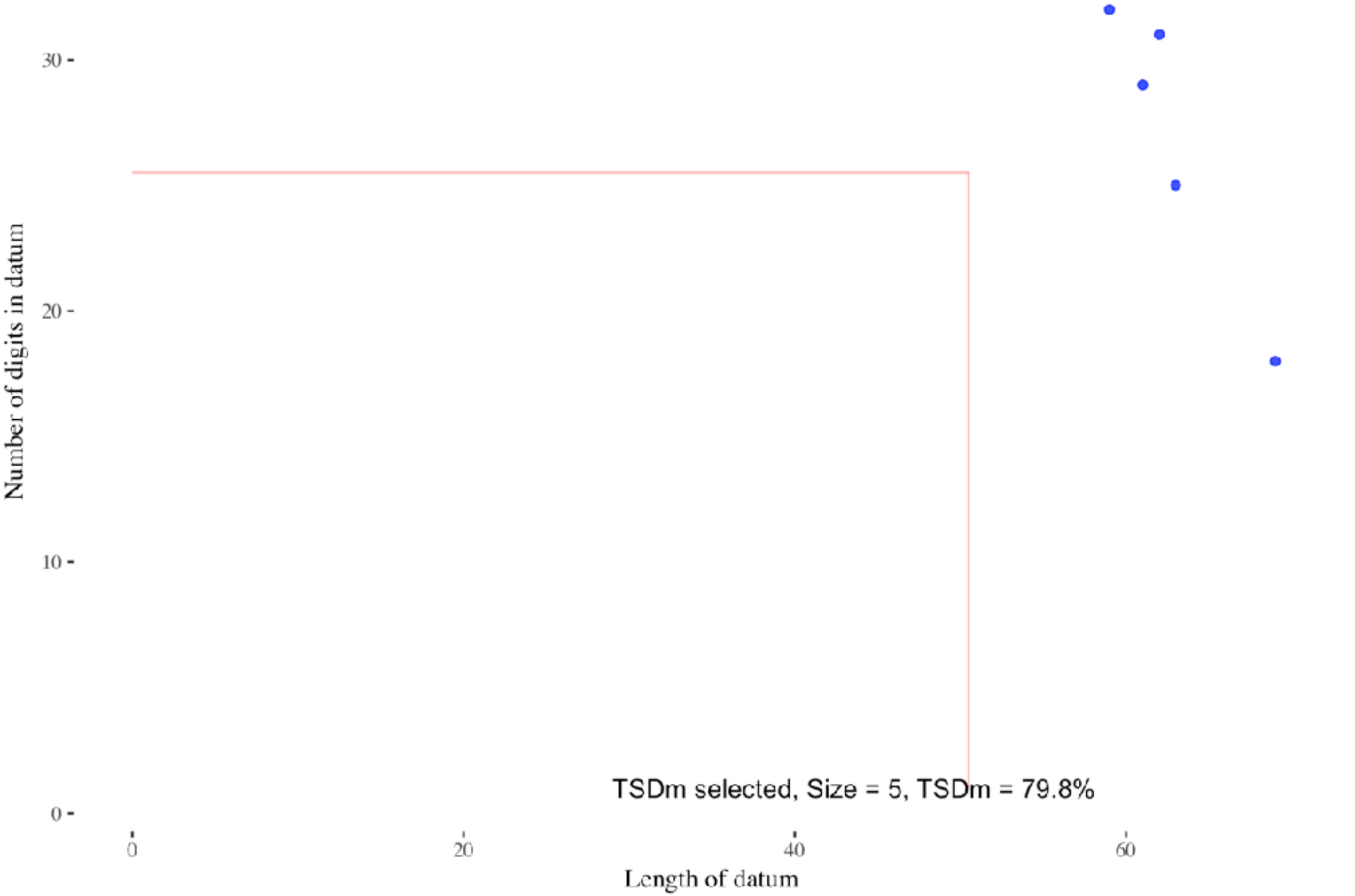
Length vs Num digits



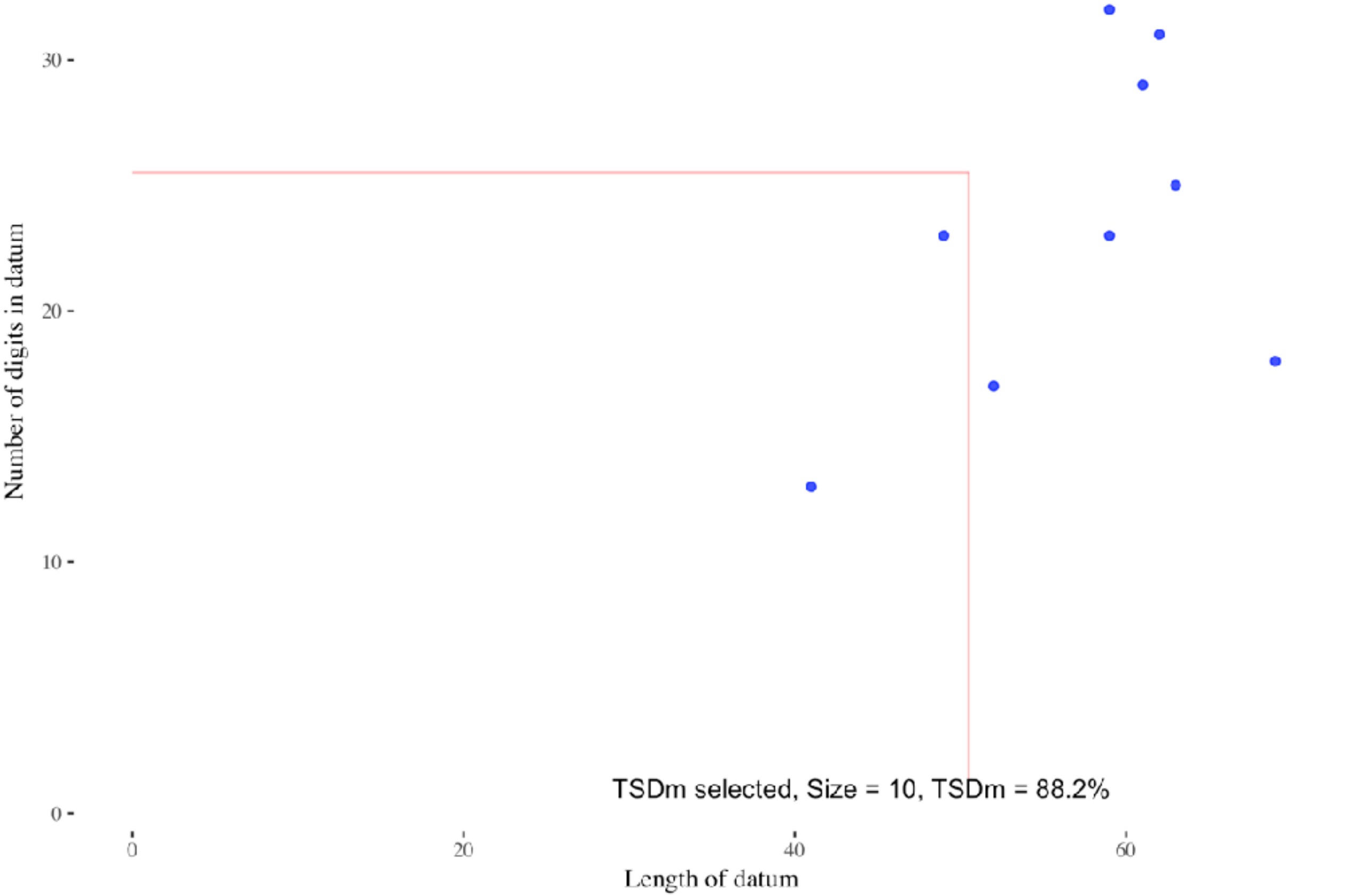
Length vs Num digits



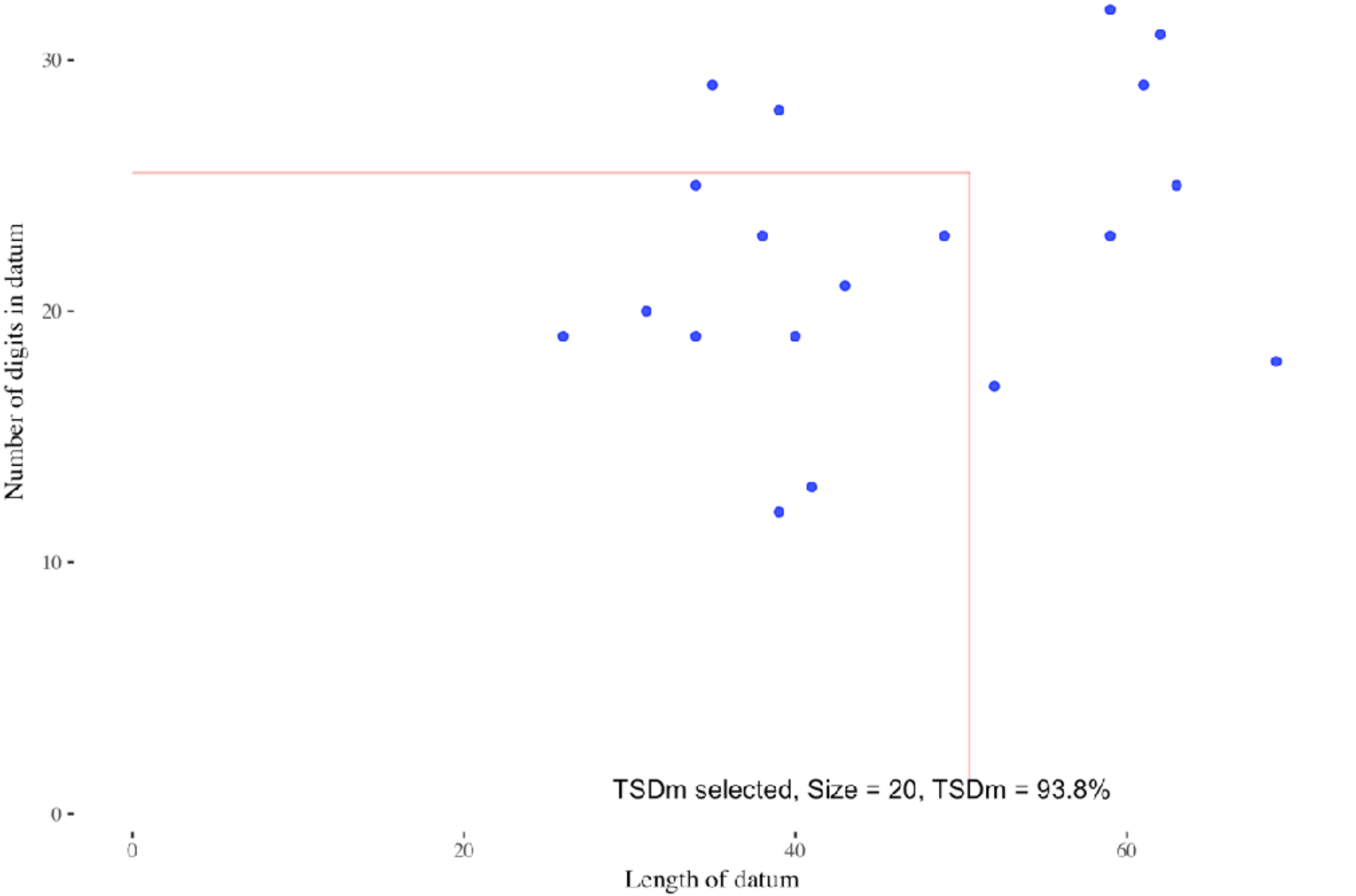
Length vs Num digits



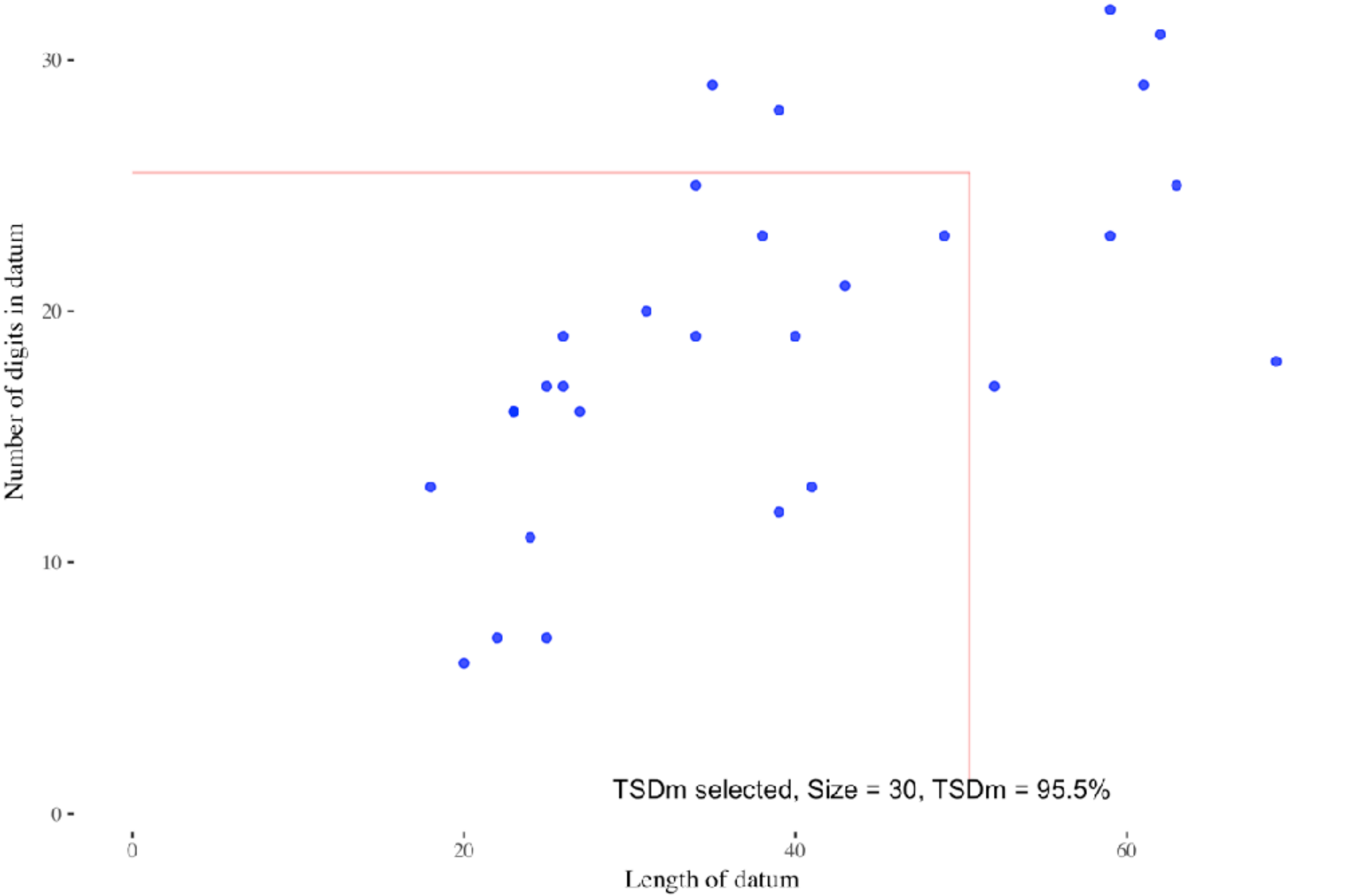
Length vs Num digits



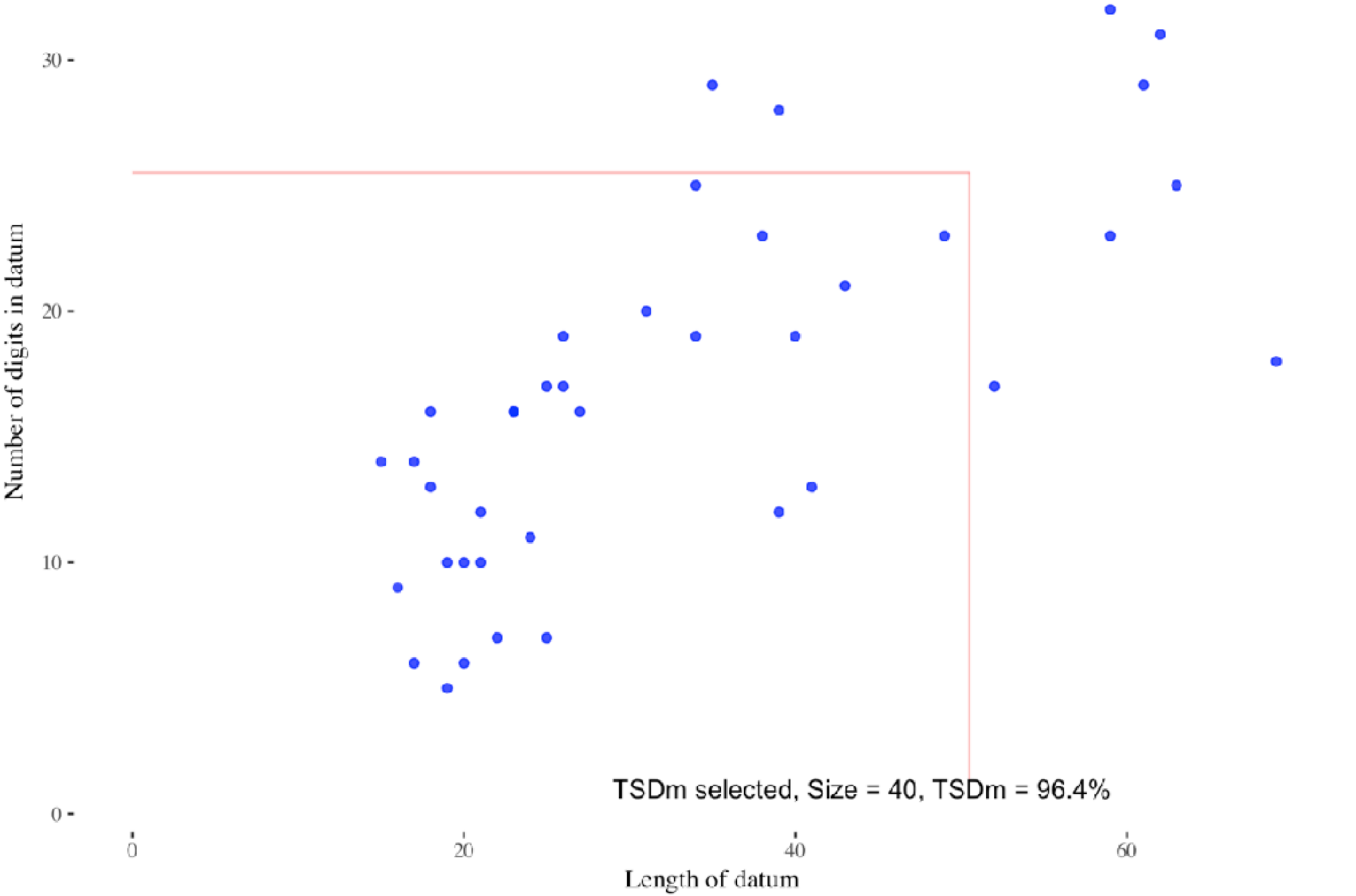
Length vs Num digits



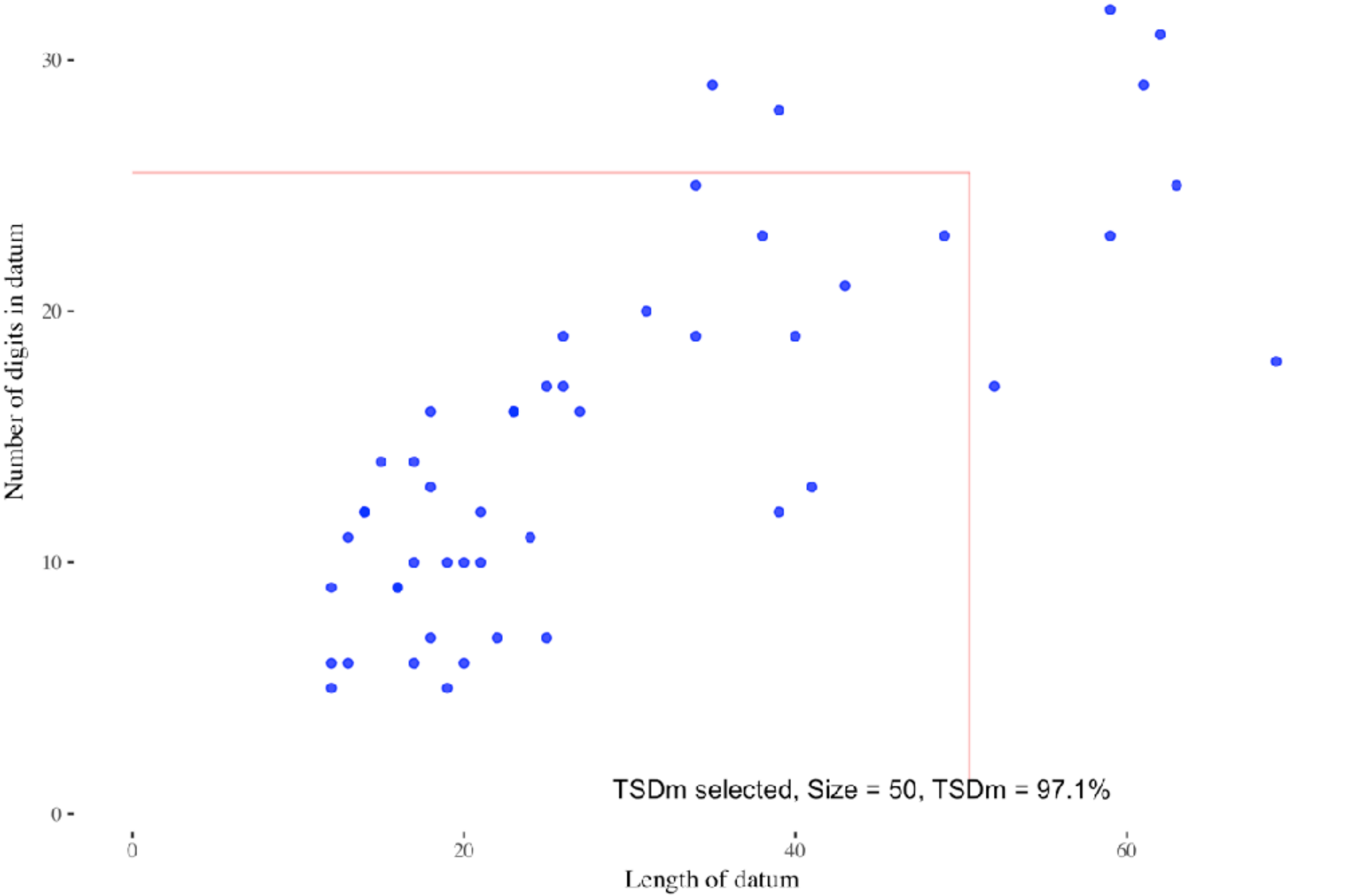
Length vs Num digits



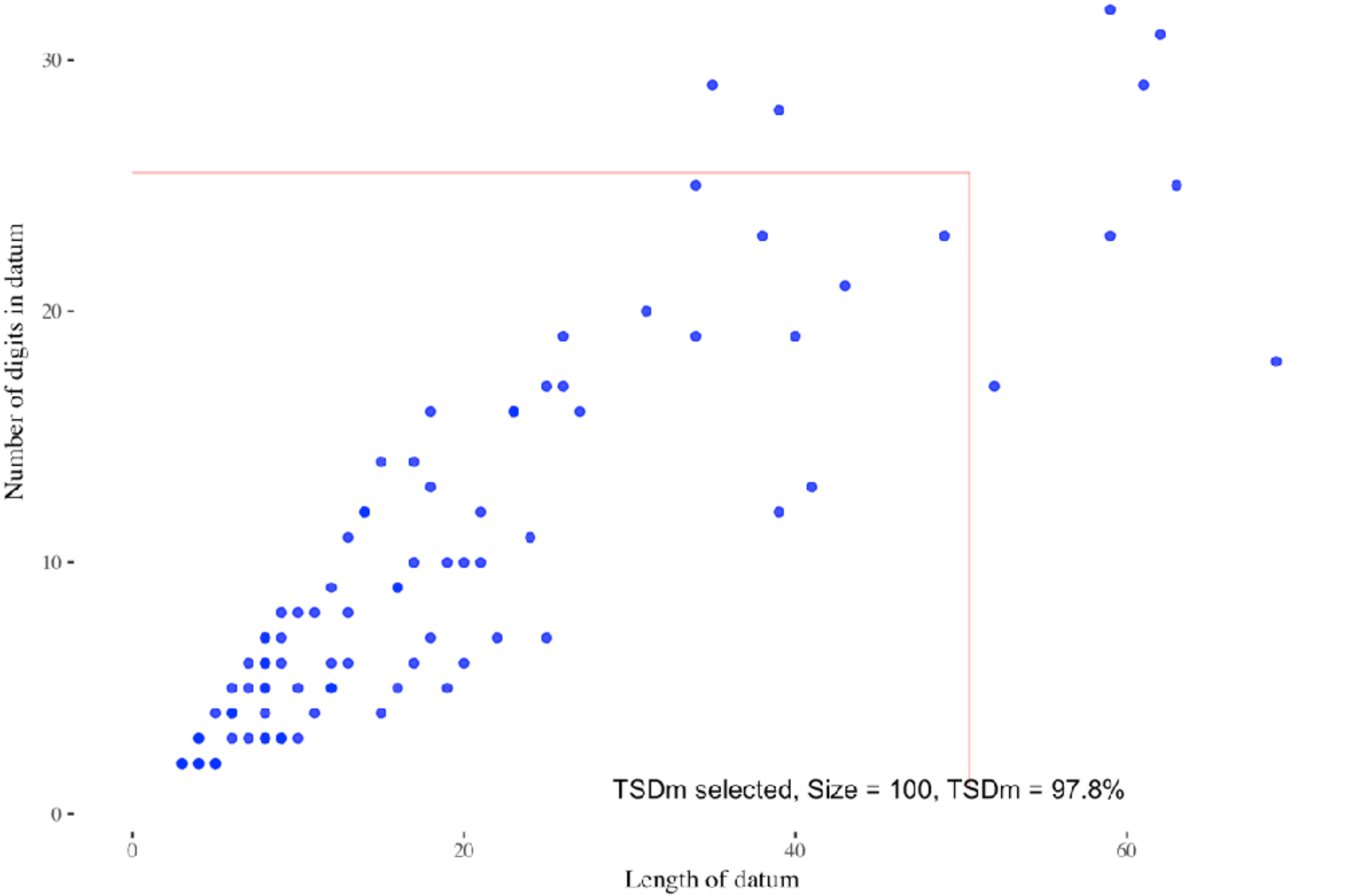
Length vs Num digits



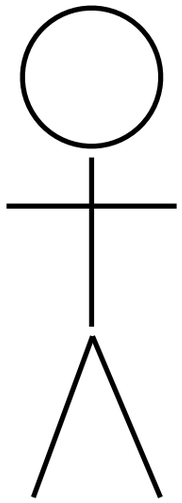
Length vs Num digits



Length vs Num digits



Main message: There is a trade-off between two types of DIVERSITY



NID, NCD

Domain-specific
Feature-specific

General, even Universal

Specific, problem adapted

Ar ath)

Hard to analyse, no theorems

Simple & Cheap (to Human)

~Costly (to Human)

N on

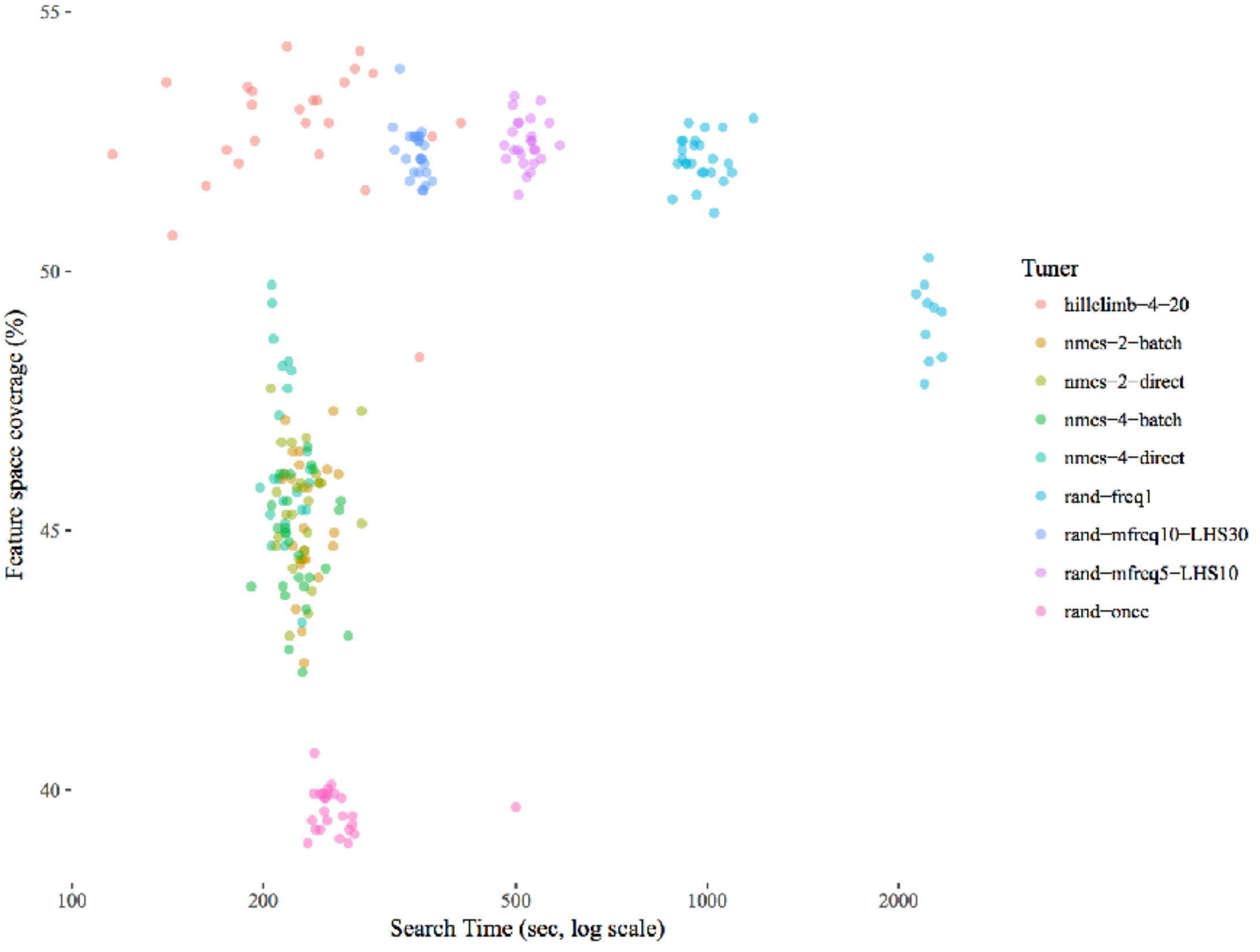
L le

Risk being unfocused

Risk hiding some features

Risk of missing important features

robert.feldt@chalmers.se


55
50
45
40
Feature space coverage (%)
100
200
500
1000
2000
Search Time (sec, log scale)
Tuner
hillclimb−4−20
nmcs−2−batch
nmcs−2−direct
nmcs−4−batch
nmcs−4−direct
rand−freq1
rand−mfreq10−LHS30
rand−mfreq5−LHS10
rand−once

TSDm is already being 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

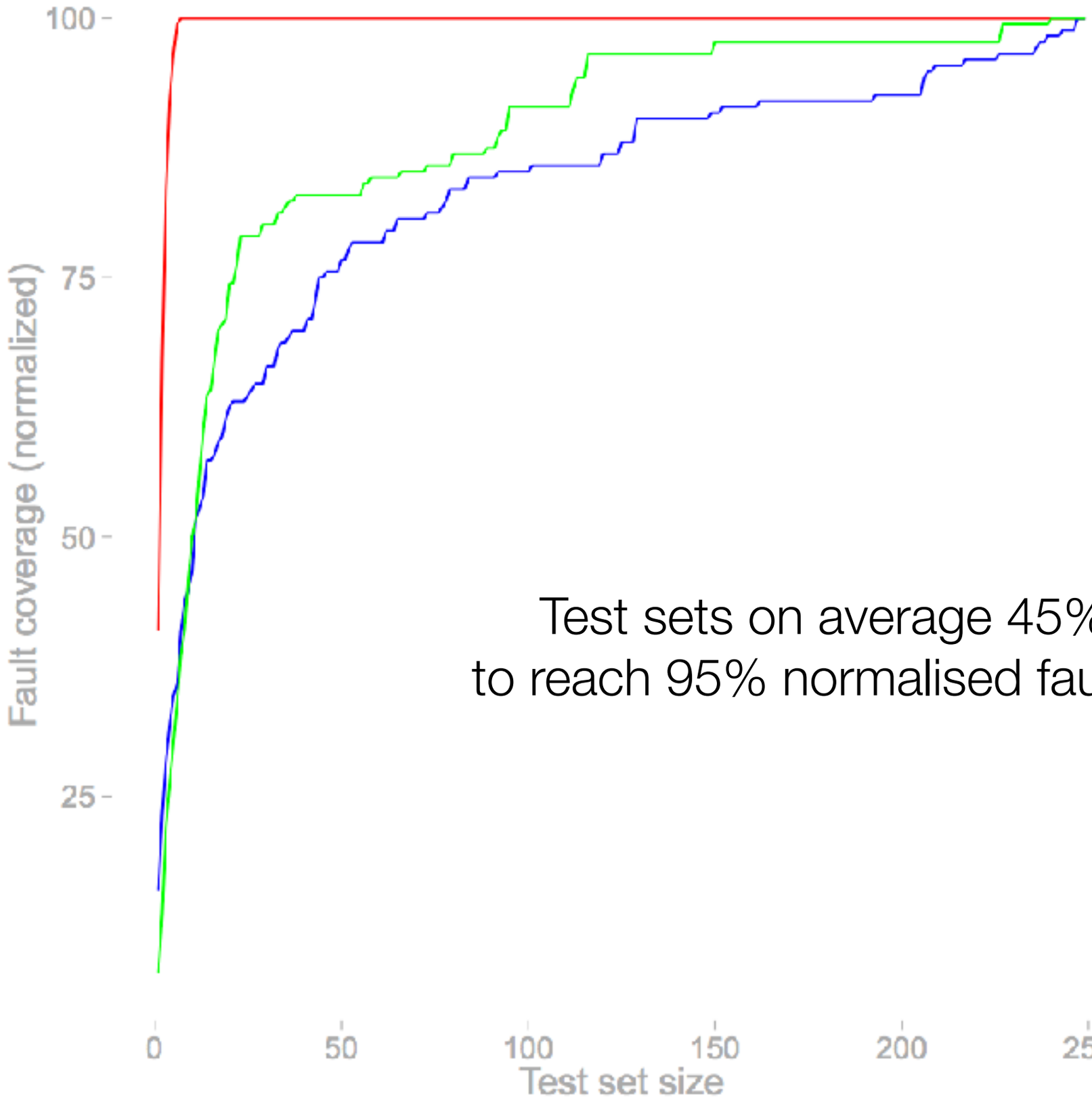
ABSTRACT

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.

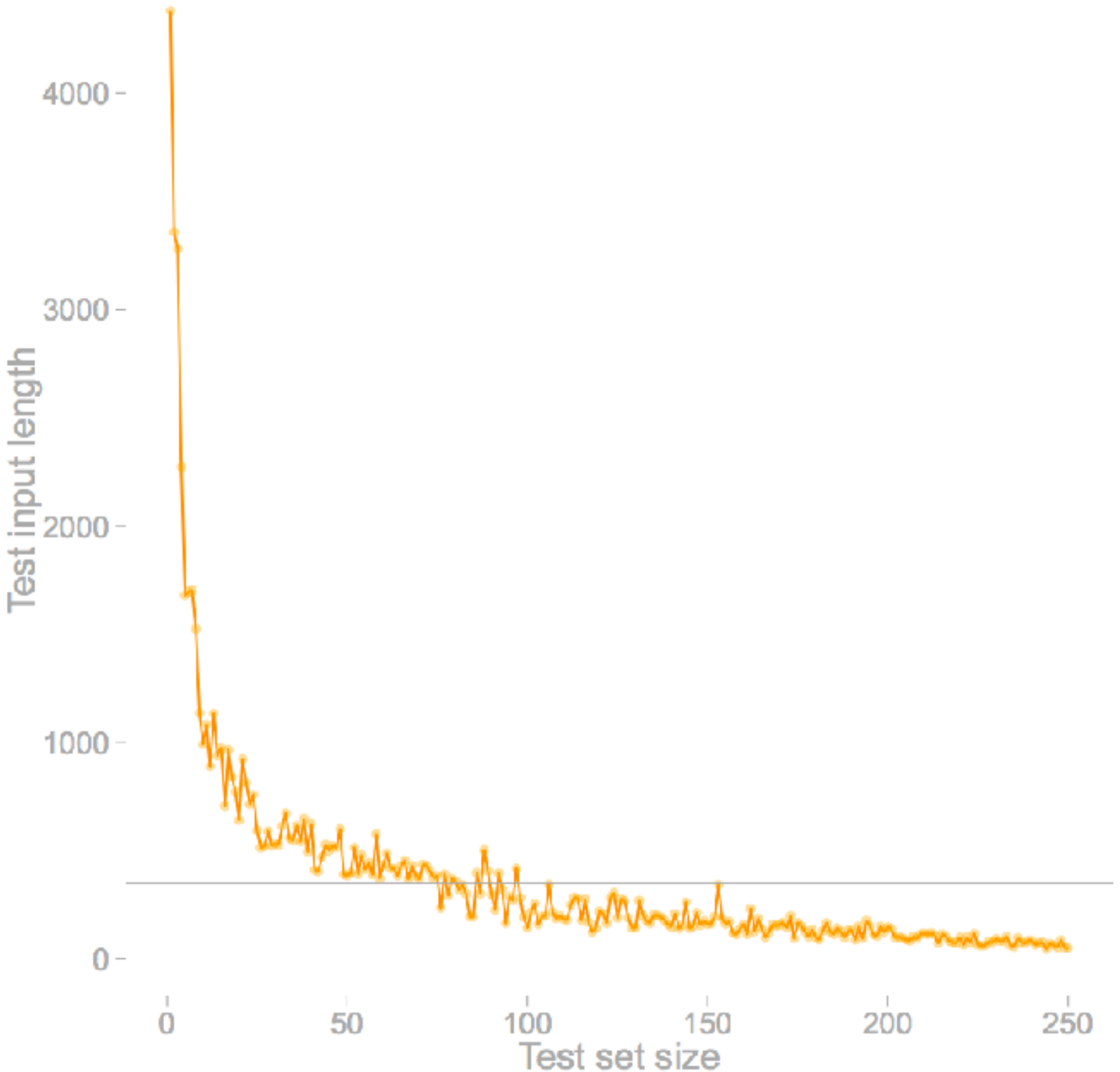
Black-box testing has the advantage of not requiring source code, thereby obviating the need for instrumentation and source code availability. Conversely, one might hypothesize that accessing source code information would allow white-box testing to increase source code coverage and, thereby, to increase 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-

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



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

Word of caution! Length of test case most important!



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)

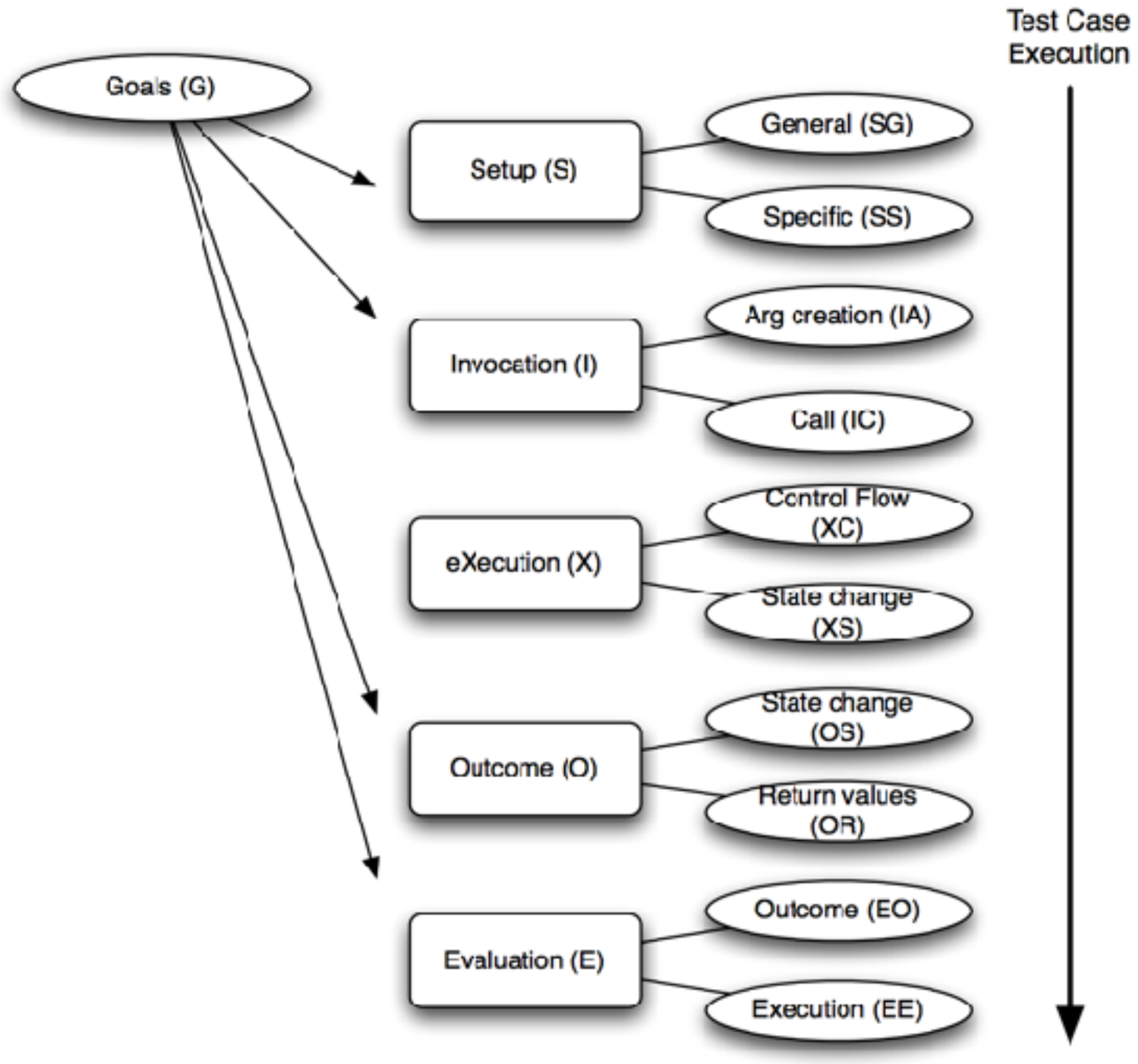
The “Compression trick”

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) = 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.

Many sources of test case information



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

Test Set Diameter:

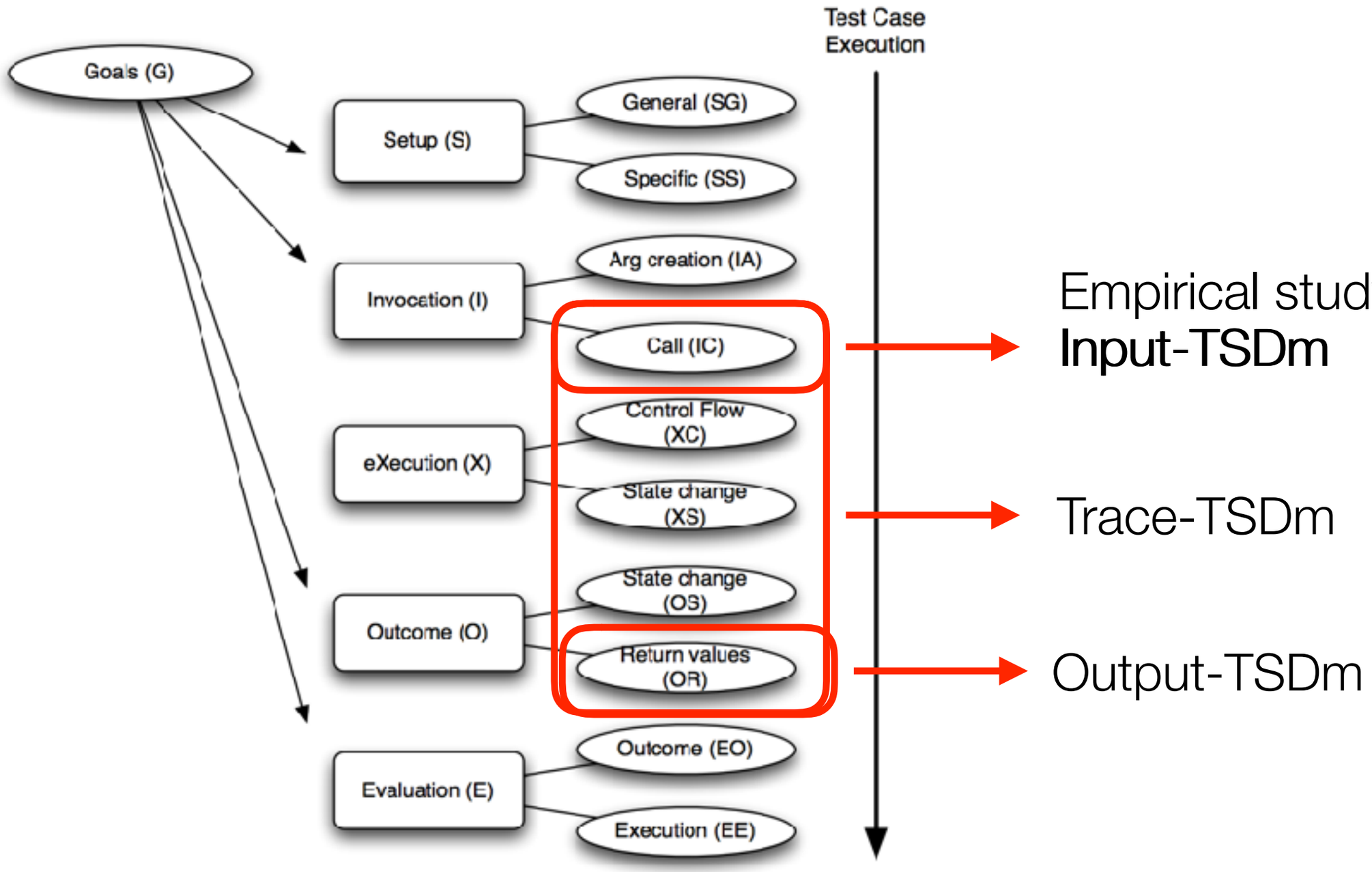
Quantifying the Diversity of Sets of Test Cases

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




BLEKINGE TEKNISKA HÖGSKOLA · BTH ·
DEPARTMENT OF SOFTWARE ENGINEERING
UCL
KAIST

TSDm = NCDm(subset of VAT info)



Empirical study here:
Input-TSDm

Trace-TSDm

Output-TSDm

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?

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

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 in report) Boundary Value testing.
  - Focusing on available information also has added value in industry collaborations.

Searching for (Test) Diversity

Robert Feldt, Simon Poulding

BLEKINGE TEKNISKA HÖGSKOLA · BTH ·

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

Method	ChoiceModel	Runs	Coverage	std	Time	Preferred
hillclimb − 4 − 20	RecDepth5	25	52.7	1.3	235.9	80.5
rand − mfreq5 − LHS10	RecDepth5	25	52.5	0.5	519.4	65.7
rand − mfreq10 − LHS30	RecDepth5	25	52.3	0.5	348.7	66.8
rand − freq1	RecDepth5	25	52.2	0.5	980.1	61.9
rand − freq1	Default	10	49.1	0.8	2237.1	51.1
nmcs − 4 − direct	Default	25	46.4	1.6	217.6	62.4
nmcs − 2 − direct	Default	25	45.4	1.2	231.3	61.9
nmcs − 2 − batch	Default	25	45.2	1.2	234.3	61.5
nmcs − 4 − batch	Default	25	44.7	1.2	228.6	61.7
rand − once	Default	25	39.6	0.4	265.2	64.0

Table 1. Descriptive statistics on the performance of the 10 investigated methods on the 2-dimensional feature space of string length and number of digits for the ExprGen generator. The 'Runs' columns shows the number of runs per method, 'Coverage' shows the mean FSHC while 'std' is its standard deviation. Finally, 'Time' is the mean search time in seconds and 'Preferred' is the ratio of samples that is within the preference hypercube.