Automatic Generation of Tests with Constraint Programming

Constraint Programming Project Presentation – EPITA Class of 2026 – Friday, April 11 2025

Charles-Antoine Leger - Rania Saadi - Armand Thibaudon

charles-antoine.leger@epita.fr
rania.saadi@epita.fr
armand.thibaudon@epita.fr

Classic Example

Let’s consider a Python function with **10 parameters** where each of theses parameters has between **3 and 10 possible values**:

```
def complex(
    param1, param2, param3, param4, param5, param6, param7, param8, param9, param10
):
```

- How many **possible factor combinations** ?
- Our goal is to **reduce** this number while maintaining a **high test suite coverage**

Introduction

Generate, automatically, tests for a python **function** based on its **factors** (parameters) and their **factor levels** (possible values)

Main challenges:

- **Combinatorial Explosion**: with a high number of parameters and large domains (high factor levels), the test suite can become too large, leading to long execution times
- **Avoid Redundancy:** many combinations may be redundant or provide little additional coverage
- **Ensure Sufficient Code Coverage:** the generated test suite must provide strong coverage to ensure code safety and reliability

Algorithm

- The algorithm is implemented in **python**
- **Z3** package for handling constraints solving
- **coverage** package to compute coverage of tests
- **MCP** for LLM prompting

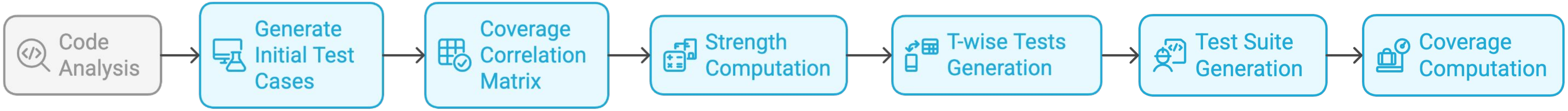

Figure 1: Presented algorithm pipeline

Code Analysis

Our aim:

- To determine **which values of a parameter** meaningfully affect the function's behavior.
- To avoid testing **redundant** or **equivalent** inputs.
- Ideally, to do so **automatically**.

Let’s begin by identifying the **parameters** and their respective **domains**:

- Static analysis
- LLM

```python
def is_positive(n):
    if n > 0:
        return True
    else:
        return False
```

t-wise Testing

What is t-wise testing?

- Also called **Combinatorial Interaction Testing** (CIT).
- A **black-box testing** approach that focuses on interactions between input parameters.
- ‘`t`’ represents the **interaction strength** — e.g., 2-wise tests all pairs, 3-wise tests all triples, etc.

Why is it pertinent?

- **Efficient fault detection**: Many software faults are triggered by specific combinations of inputs. t-wise testing detects these without exhaustive testing.
- **Scalable sampling strategy**: Instead of checking every possible combination (which grows exponentially), it ensures coverage of smaller but meaningful sets of interactions using covering arrays.
- **Proven effectiveness**: Studies (e.g., Kuhn et al.) show high fault detection rates with relatively low test suite sizes.

Proven Effectiveness of t-wise Testing

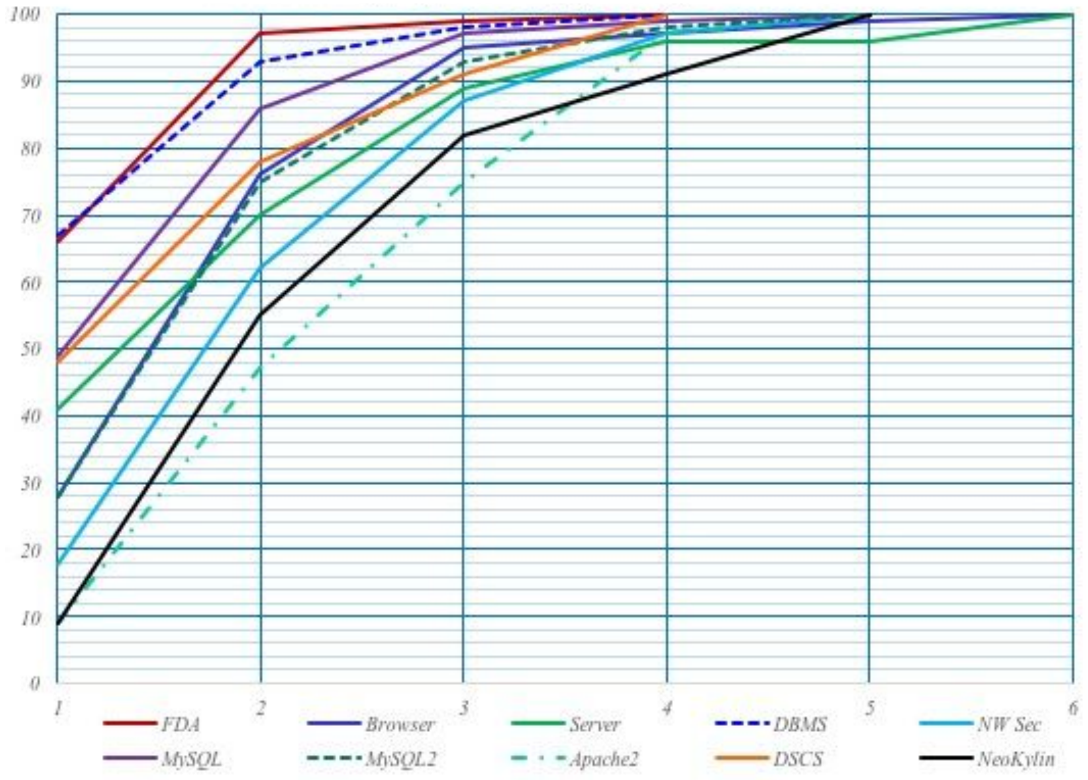

Figure 2: Cumulative proportion of faults[1]

[1]: Combinatorial Methods for Trust and Assurance (https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software)

t-wise Testing Matrix

- Let's imagine a function that takes **three parameters**, each with the following **domain**: [0, 1]
- Our goal is to construct the **corresponding pairwise (2-wise) testing matrix** for this function

X	Y	Z
0	1	0
1	0	1
0	0	0
1	1	0
0	1	1

Figure 3: t-wise combination matrix for $t = 2$

Initial Tests Generation

- From each parameter and its domain, we generate **all possible combinations** to serve as inputs for the function
- Once this is done, we **randomly sample** from these combinations to obtain a representative subset

Correlation Matrix

- To choose **t** for a test case, we rely on the **correlation** coefficient
- It gives an idea on how much two or more parameter are related

$$\mathrm{Corr}(X,Y) = \frac{I_{XY}}{I_X * I_Y}$$

- **Correlation** is computed from the **impact** of parameters on the code coverage

$$I_p = C_p^{\max} - C_p^{\min}$$

- It gives us the correlation matrix

Parameter	P_1	P_2	...	P_n
P_1	—	$\mathrm{Corr}(P_1, P_2)$	...	$\mathrm{Corr}(P_1, P_n)$
P_2	$\mathrm{Corr}(P_2, P_1)$	—	...	$\mathrm{Corr}(P_2, P_n)$
...	...	...	—	...
P_n	$\mathrm{Corr}(P_n, P_1)$	$\mathrm{Corr}(P_n, P_2)$	...	—

Table 1: Conceptual model of parameter correlation matrix

Strength Calculation

Parameter	P_1	P_2	...	P_n
P_1	—	$\text{Corr}(P_1, P_2)$	...	$\text{Corr}(P_1, P_n)$
P_2	$\text{Corr}(P_2, P_1)$	—	...	$\text{Corr}(P_2, P_n)$
...	...	...	—	...
P_n	$\text{Corr}(P_n, P_1)$	$\text{Corr}(P_n, P_2)$	...	—

- To choose a the number of classes of strength (t) for a pair, we calculate the **variance of the correlation (V)** and the **range of the correlation (R)**
- The **number of classes of strength (S)** determines the **number of correlation thresholds (T)**; we chose to have equally spaced thresholds such that:

With **number of parameters N** $N >= 2$ and $i_0 = 1$, we define:

$$S = max(V(N-1), 2)$$

$$T_i = i * (R/S)$$

Results - 1

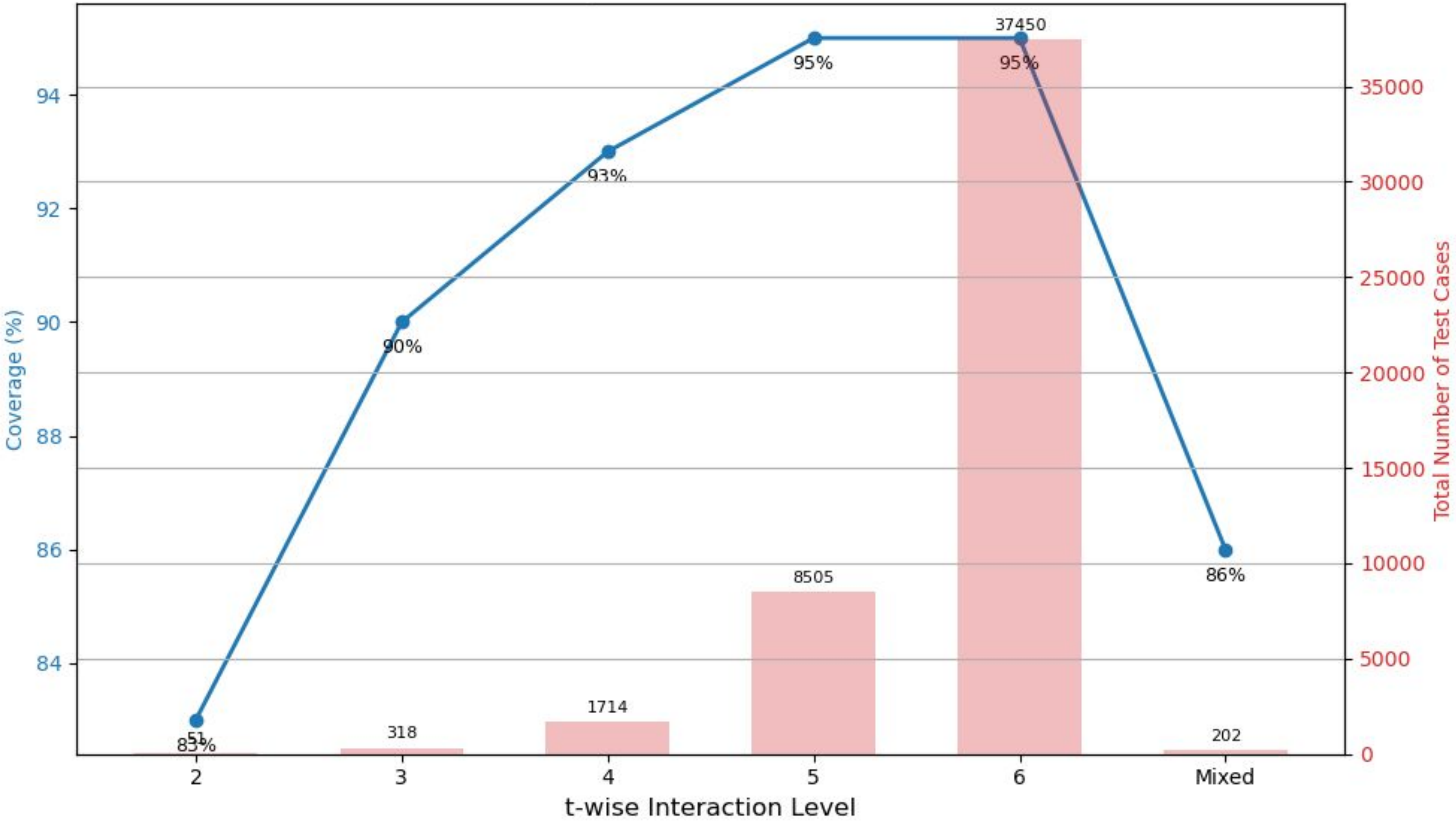

Figure 4: Test Coverage per t-wise Interaction Level - 1

Results - 2

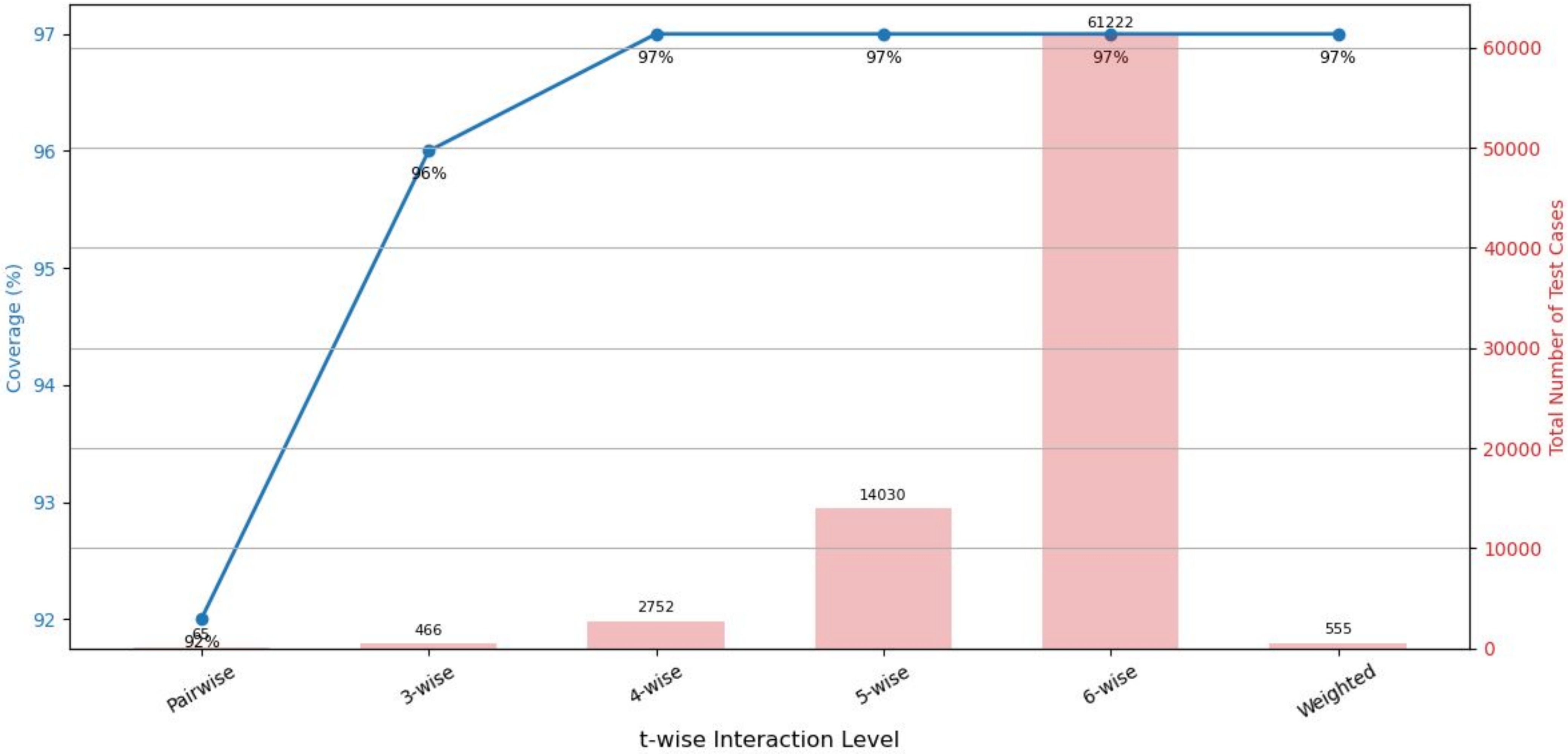

Figure 5: Test Coverage per t-wise Interaction Level - 2

Live Demo

Let's try it out !

References

- Perrouin, G., Papadakis, M., Sen, S., & Klein, J. (2018). Code-aware combinatorial interaction testing. IET Software. https://doi.org/10.1049/iet-sen.2018.5315
- Zhang, X., Huang, K., Harman M., & Jia Y. (2020). Code Coverage Aware Test Generation Using Constraint Solver. arXiv preprint arXiv:2009.02915. https://arxiv.org/pdf/2009.02915
- Johansen D., Anda B.C.D., & Dybå T.. (2016). Constrained Interaction Testing: A Systematic Literature Study. https://core.ac.uk/download/pdf/159193894.pdf
- Kuhn, D.R., Kacker R.N., Lei Y.. (n.d.). Combinatorial Methods for Trust and Assurance. National Institute of Standards and Technology (NIST). https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software/combinatorial-methods-in-testing/interactions-involved-in-software-failures

References

- Combinatorial Interaction Testing 21 May 2021: https://www.youtube.com/watch?v=wuvJu3a0TV4&t=793s
- Introduction to Combinatorial Interaction Testing, Rick Kuhn, 7 June 2011: https://mse.s3d.cmu.edu/facstaff/faculty1/faculty-publications/miranda/kuhnintroductioncombinatorialtesting.pdf
- Achievement of Minimized Combinatorial Test Suite for Configuration-Aware Software Functional Testing Using the Cuckoo Search Algorithm, Bestoun S. Ahmed, Tailb Sh. Abdulsamad: https://arxiv.org/abs/1904.04348