PhD Thesis

# Simulation-Based Testing of Highly Configurable Cyber-Physical Systems: Automation, Optimization and Debugging

*Author:*
Aitor Arrieta Marcos

*Supervisors:*
Dr. Goiuria Sagardui Mendieta
Dr. Leire Etxeberria Elorza

Computer and Electronics Department
Faculty of Engineering
Mondragon Unibertsitatea

Arrasate
October 2017

*“The most exciting phrase to hear in science, the one that heralds the most discoveries is not “Eureka!” but “That’s funny...””*
*Isaac Asimov*

Nire gurasoei,
Aitor

# Declaration

Hereby I declare that this document is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

*Aitor Arrieta Marcos*
*Arrasate, October 2017*

# Acknowledgments

After writing this document, I believe that I could write another long chapter thanking one by one everybody that in a way or in another has supported me during the Ph.D. But, I will try to be concise for once!

First of all I would like to thank Goiuria and Leire for their patience and support. They have given me freedom to do whatever I wanted but at the same time they guided me towards our objectives. I want to thank Goiuria for giving me the opportunity to do the Ph.D. when I was a master student. Her support has been constant, giving me valuable feedback in the whole process and encouraging me in the worst moments. I want to thank Leire for her positiveness every time I proposed something I believed it was “novel”. Her feedback has been always valuable and her rigor when reviewing our papers (and this document) led always to find even the smallest typo. Without their support, the quality of this dissertation would have been much lower!

I am also thankful to Mondragon Goi Eskola Politeknikoa as well as to the Embedded Systems group for their financial support.

Secondly, I would like to thank three great researchers (but better persons) that were at different stages of the project. All of them had the patience to teach me new things. First, I would like to thank Justyna Zander for her feedback when developing ASTERYSCO and for providing this thesis with an industrial case study from Daimler. Second, I would like to thank Shuai Wang, who supported me in the hardest stages. From him I learned everything I know about search-based software engineering. His support and feedback have been crucial in this journey and I hope to keep our collaboration for many years. Lastly, I would like to thank Sergio Segura, one of the best researchers a Ph.D. student can have during his/her studies. I want to thank him for his patience and time during my research stay in Sevilla as well as when writing the papers. Sergio is a rigorous researcher that changed my vision of doing research.

My journey towards the Ph.D. wouldn’t have been so entertaining without them: the other Ph.D. students in the lab. We did a great team, supporting each other in the worst moments, but enjoying together the best ones. I just hope to keep sharing with them these best moments. Thanks to Idoia, Iñaki, Maite, Joxe, Aritz, Raul, Enaitz, Alain, Mikel I., Ane, Aitor, Dani, Urtzi, Miren, Mikel M., Oscar, Pablo and Javi. I would like to give special thanks to Urtzi for his support during the last stage of the thesis. In addition, I would like to thank the colleagues from Simula Research Laboratory and the University of Sevilla for all those good moments. I would like to thank Tao and Shaukat for letting me incorporate in their research team. I would also like to thank the rest of the software engineering team of Simula: Dipesh, Shuai,

Stefano, Man, Sagar, Hong, Tao M., Safdar, Muhammad, Phu, Huihui, Sigrid, Leon... Furthermore, I would like to thank Sergio Segura and Antonio Ruiz for inviting me to join the ISA group, as well as the rest of the ISA group, especially Ana Belén and Gabri. Sharing the office with them was fun; I will always remember those great breakfasts in the canteen of the university. Furhtermore, I would like to thank José Antonio Parejo for his availability when I had doubts with statistical tests.

My friends have been great supporters during this whole process. I would like to thank Aritz and Amaia for all those dinners at their home when I was back in Lasarte on Fridays. Also, thanks to Aitor Zelarain for his support. Also, thanks to all the PUM for those great moments all together: Gonzi, Pablo, Patxi, Doggi, Lope, Luis, Dani, Andres, Churras, Angel, Jimma, Sanchi and Fer. To “Family Mordor”, for all those pintxo potes on Thursdays: Maria, Iñaki, Raquel, Irune Y., Miren, Rita, Modesto and especially Irune A., for her support and friendship during all these years. Finally, thanks to “my Norwegian family”, that being an stranger to them, they always treated me as if I was an old friend: Javi, Cristo, Mari Jose, Anna and Victoria.

Por último, me gustaría agradecer a mi familia! En especial a mis aitas, Patxi y Rosi. Porque si soy lo que soy y he llegado hasta aquí es por vuestro esfuerzo y apoyo. Y a mi hermano, Iñigo. Por ser como eres, y porque aunque creas que este documento es una frikada, tu apoyo ha sido incondicional desde Bilbo, China, Mexico o cualquier otro lugar en el que hayas estado durante estos años!

# Abstract

Cyber-Physical Systems (CPSs) integrate digital cyber technologies with physical processes. The variability of these systems is increasing in order to give solution to the different customers demands. As a result, CPSs are becoming configurable or even product lines, which means that they can be set into thousands or millions of configurations. Testing configurable CPSs is a time consuming process, mainly due to the large amount of configurations that need to be tested. The large amount of configurations that need to be tested makes it infeasible to use a prototype of the system. As a result, configurable CPSs are being tested using simulation. However, testing CPSs under simulation is still challenging. First, the simulation time is usually long, since apart of the software, the physical layer needs to be simulated. This physical layer is typically modeled with complex mathematical models, which is computationally very costly. Second, CPSs involve different domains, such as, mechanical and electrical. Engineers of different domains typically employ different tools for modeling their subsystems. As a result, co-simulation is being employed to interconnect different modeling and simulation tools. Despite co-simulation being an advantage in terms of engineers flexibility, the use of different simulation tools makes the simulation time longer. Lastly, when testing CPSs employing simulation, different test levels exist (i.e., Model, Software and Hardware-in-the-Loop), what increases the time for executing test cases.

This thesis aims at advancing the current practice on testing configurable CPSs by proposing methods for automation, optimization and debugging. Regarding automation, first, we propose a tool supported methodology to automatically generate test system instances that permit automatically testing configurations of the configurable CPS (e.g., by employing test oracles). Second, we propose a test case generation approach based on multi-objective search algorithms that generate cost-effective test suites. As for optimization, we propose a test case selection and a test case prioritization approach, both of them based on search algorithms, to cost-effectively test configurable CPSs at different test levels. Regarding debugging, we adapt a technique named Spectrum-Based Fault Localization to the product line engineering context

and propose a fault isolation method. This permits localizing bugs not only in configurable CPSs but also in any product line where feature models are employed to model variability.

# Laburpena

Sistema Ziber-Fisikoek sistema ziber digitalak sistema fisikoekin uztartzen dituzte. Sistema hauen aldakortasuna handitzen ari da erabiltzaileen hainbat behar betetzeko. Ondorioz, sistema ziber-fisikoa aldakorrak edota produktu lerroak ari dira garatzen eta sistema hauek milaka edo milioika konfiguraziotan konfiguratu daitezke. Sistema ziber-fisiko aldakorren test eta balidazioa prozesua garestia da, batez ere probatu beharreko konfigurazio kopuruaren ondorioz. Konfigurazio kopuru altuak sistemaren prototipo bat erabiltzea ezinezkoa egiten du. Horregatik, sistema ziber-fisiko aldagarriak simulazio modeloak erabilita probatzen dira. Hala ere, simulazio bidez sistema ziber-fisikoak probatzea erronka izaten jarraitzen du. Hasteko, simulazio denbora altua izaten da normalki, software-az aparte, sistema fisikoa simulatu behar delako. Sistema fisiko hau normalean modelo matematiko konplexuen bitartez modelatzen da, konputazionalki garestia delarik. Jarraitzeko, sistema ziber-fisikoek ingeniaritzaren domeinu ezberdinak dituzte tartean, adibidez mekanika edo elektronika. Domeinu bakoitzak bere simulazio erremienta erabiltzen du, eta erremienta guzti hauek interkonektatzeko ko-simulazioa erabiltzen da. Nahiz eta ko-simulazioa abantaila bat izan ematen duen flexibilitateagatik, simulagailu ezberdinen erabilerak simulazio denbora handiagotzen du. Azkenik, sistema ziber-fisikoak simulaziopean probatzean, probak maila ezberdinetan egin behar dira (adb., Model, Software eta Hardware-in-the-Loop mailak), eta honek, proba-kasuak exekutatzeko denbora handitzen du.

Tesi honen helburua sistema ziber-fisiko aldakorren test jardunbideak hobetzea da, horretarako automatizazio, optimizazio eta arazketa metodoak proposatzen ditu. Automatizazioari dagokionez, lehenengo, erremienta-bidezko metodologia bat proposatzen da. Metodologia hau test sistema instantziak automatikoki sortzeko gai da, test sistema hauek sistema ziber-fisiko aldagarrien konfigurazioak automatikoki probatzeko gai dira (adb., test orakuluen bitartez). Bigarren, test frogak automatikoki sortzeko planteamendu bat proposatzen da helburu anitzeko bilaketa algoritmoak erabilita. Optimizazioari dagokionez, test frogen aukeraketarako planteamendu bat eta test frogen priorizaziorako beste planteamendu bat proposatzen dira, biak bilaketa al-

goritmoak erabiliz, sistema ziber-fisiko aldakorrak test maila ezberdinetan probatzeko helburuarekin. Arazketari dagokionez, “espektroan oinarritutako falten lokalizazioa” izeneko teknika bat produktu lerroen testuingurura adaptatu da, eta faltak isolatzeko metodo bat proposatzen da. Honek, falta ezberdinak lokalizatzea errezten du ez bakarrik sistema ziber-fisiko aldakorretan, baizik eta edozein produktu lerrotan non “feature model” delako modeloak erabiltzen diren aldakortasuna kudeatzeko.

# Resumen

Los *sistemas cyber-físicos* (CPSs) integran tecnologías digitales con procesos físicos. La variabilidad de estos sistemas está creciendo para responder a la demanda de diferentes clientes. Como consecuencia de ello, los CPSs están volviéndose configurables e incluso líneas de producto, lo que significa que pueden ser configurados en miles y millones de configuraciones. El testeo de *sistemas cyber-físicos* configurables es un proceso costoso, en general debido a la cantidad de configuraciones que han de ser testeadas. El número de configuraciones a testear hace imposible el uso de un prototipo del sistema. Por ello, los sistemas CPSs configurables están siendo testeadas utilizando modelos de simulación. Sin embargo, el testeo de *sistemas cyber-físicos* bajo simulación sigue siendo un reto. Primero, el tiempo de simulación es normalmente largo, ya que, además del software, la capa física del CPS ha de ser testeada. Esta capa física es típicamente modelada con modelos matemáticos complejos, lo cual es computacionalmente caro. Segundo, los *sistemas cyber-físicos* implican el uso de diferentes dominios de la ingeniería, como por ejemplo la mecánica o la electrónica. Por ello, para interconectar diferentes herramientas de modelado y simulación hace falta el uso de la co-simulación. A pesar de que la co-simulación es una ventaja en términos de flexibilidad para los ingenieros, el uso de diferentes simuladores hace que el tiempo de simulación sea más largo. Por último, al testear *sistemas cyber-físicos* haciendo uso de simulación, existen diferentes niveles (p.ej., Model, Software y Hardware-in-the-Loop), lo cual incrementa el tiempo para ejecutar casos de test.

Esta tesis tiene como objetivo avanzar en la práctica actual del testeo de *sistemas cyber-físicos* configurables, proponiendo métodos para la automatización, optimización y depuración. En cuanto a la automatización, primero, se propone una metodología soportada por una herramienta para generar automáticamente instancias de sistemas de test que permiten testear automáticamente configuraciones del sistema CPS configurable (p.ej., haciendo uso de oráculos de test). Segundo, se propone un enfoque para generación de casos de test basado en algoritmos de búsqueda multi-objetivo, los cuales generan un conjunto de casos de test. En cuanto a la optimización, se propone un enfoque para selección y otro para priorización de casos de test, ambos

basados en algoritmos de búsqueda, de cara a testear eficientemente *sistemas cyber-físicos* configurables en diferentes niveles de test. En cuanto a la depuración, se adapta una técnica llamada “Localización de Fallos Basada en Espectro” al contexto de líneas de productos y proponemos un método de aislamiento de fallos. Esto permite localizar bugs no solo en *sistemas cyber-físicos* configurables sino también en cualquier línea de producto donde se utilicen modelos de características para gestionar la variabilidad.

# Contents

**PART I FOUNDATION AND CONTEXT** 1

**1 Introduction** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **3**

1.1 Motivation and Scope of the Research . . . . . . . . . . . . . . . . . 3

1.2 Research Methodology . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.3 Technical Contributions . . . . . . . . . . . . . . . . . . . . . . . . . 7

1.4 Publications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

1.4.1 Journal Articles . . . . . . . . . . . . . . . . . . . . . . . . 9

1.4.2 International Conferences . . . . . . . . . . . . . . . . . . . 9

1.4.3 Workshops and National Conferences . . . . . . . . . . . . . 11

1.5 Related Activities . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

1.5.1 Talks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

1.5.2 Research Stays . . . . . . . . . . . . . . . . . . . . . . . . 12

1.5.3 Service . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

1.5.4 Proposals Writing . . . . . . . . . . . . . . . . . . . . . . 14

1.6 Document Structure . . . . . . . . . . . . . . . . . . . . . . . . . . 14

**2 Technical Background** . . . . . . . . . . . . . . . . . . . . . . . . . . **15**

2.1 Configurable Cyber-Physical Systems . . . . . . . . . . . . . . . . 15

2.2 Feature Models . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.3 Simulation-based Testing of Cyber-Physical Systems . . . . . . . . 19

2.3.1 Test Systems . . . . . . . . . . . . . . . . . . . . . . . . . 20

2.3.2 Reactive Test Cases . . . . . . . . . . . . . . . . . . . . . . 21

2.3.3 Test Levels . . . . . . . . . . . . . . . . . . . . . . . . . . 21

2.4 Configurable Systems Testing . . . . . . . . . . . . . . . . . . . . . 22

2.5 Search-Based Software Testing . . . . . . . . . . . . . . . . . . . . 24

2.5.1 Local Search Algorithms . . . . . . . . . . . . . . . . . . . 24

2.5.2 Global Search Algorithms . . . . . . . . . . . . . . . . . . . 25

2.5.3 Stochastic Algorithms . . . . . . . . . . . . . . . . . . . . . 26

2.5.4 Multi-Objective Search Algorithms . . . . . . . . . . . . . . 26

**3 State of the Art** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **29**

3.1 Testing Product Lines . . . . . . . . . . . . . . . . . . . . . . . . . 29
3.2 Automation for testing Cyber-Physical Systems with Variability . . 30
3.2.1 Variability in Test Systems . . . . . . . . . . . . . . . . . . 30
3.2.2 Test Case generation for Cyber-Physical Systems . . . . . . 32
3.2.3 Other Cyber-Physical Systems Testing Approaches . . . . . 34
3.3 Optimization . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.3.1 Test Case Selection and Test Minimization . . . . . . . . . 35
3.3.2 Test Case Prioritization . . . . . . . . . . . . . . . . . . . . 36
3.4 Debugging and Fault Localization . . . . . . . . . . . . . . . . . . . 37
3.4.1 Slice-based fault localization . . . . . . . . . . . . . . . . . 38
3.4.2 Spectrum-based fault localization . . . . . . . . . . . . . . 38
3.4.3 Program State-Based Fault Localization . . . . . . . . . . . 40
3.4.4 Machine Learning-Based Fault Localization . . . . . . . . . . 41
3.4.5 Data Mining-Based Fault Localization . . . . . . . . . . . . . 41
3.4.6 Model-Based Fault Localization . . . . . . . . . . . . . . . . . 41
3.5 Critical analysis of the State of the Art . . . . . . . . . . . . . . . . . 41
**4 Theoretical Framework . . . . . . . . . . . . . . . . . . . . . . . . 45**
4.1 Research Objectives . . . . . . . . . . . . . . . . . . . . . . . . . 45
4.2 Research Hypotheses . . . . . . . . . . . . . . . . . . . . . . . . 46
4.3 Overview of the Theoretical Framework . . . . . . . . . . . . . . . 46
4.4 Case Studies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
4.4.1 Unmanned Aerial Vehicle . . . . . . . . . . . . . . . . . . . 50
4.4.2 Adaptive Cruise Control . . . . . . . . . . . . . . . . . . . . 54
4.4.3 Industrial Tank . . . . . . . . . . . . . . . . . . . . . . . . . 56
4.4.4 Direct Current (DC) Engine . . . . . . . . . . . . . . . . . . 60
4.4.5 Overview of the key characteristics of the case studies . . . 62
4.5 Case studies employed in each contribution . . . . . . . . . . . . . 63
**PART II AUTOMATION 65**
**5 Automatic Test System Generation . . . . . . . . . . . . . . . . . . . . . 67**
5.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
5.2 Overview of the Approach . . . . . . . . . . . . . . . . . . . . . . 69
5.3 Test System for CPS Validation . . . . . . . . . . . . . . . . . . . . 70
5.3.1 Test System Meta-model . . . . . . . . . . . . . . . . . . . 70
5.3.2 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
5.4 ASTERYSCO: Automatic Test System Generator . . . . . . . . . . 75
5.4.1 Input Files . . . . . . . . . . . . . . . . . . . . . . . . . . . 78
5.4.2 Test System Generation . . . . . . . . . . . . . . . . . . . 83

5.4.3 Test System Configuration . . . 85
5.4.4 Output . . . 87
5.4.5 Limitations . . . 90
5.5 Evaluation . . . 91
5.5.1 Case Study . . . 91
5.5.2 Results . . . 93
5.5.3 Comparison with Manual Test System Generation . . . 93
5.5.4 Discussion . . . 99
5.5.5 Threats to Validity . . . 101
5.6 Related Work . . . 101
5.7 Conclusion and Future Work . . . 103
**6 Automatic Test Case Generation . . . 105**
6.1 Introduction . . . 105
6.2 Multi-Objective Reactive Test Case Generation . . . 106
6.2.1 Cost-Effectiveness Measures . . . 106
6.2.2 Solution Representation . . . 110
6.2.3 Crossover Operator . . . 111
6.2.4 Mutation Operators . . . 111
6.3 Tool Support . . . 114
6.4 Empirical Evaluation . . . 115
6.4.1 Research Questions . . . 115
6.4.2 Experimental Setup . . . 115
6.4.3 Results and Analysis . . . 117
6.4.4 Discussion of the Results . . . 119
6.4.5 Threats to Validity . . . 122
6.5 Related Work . . . 123
6.6 Conclusion and Future Work . . . 125
**PART III OPTIMIZATION 127**
**7 Test Case Selection . . . 129**
7.1 Introduction . . . 129
7.2 Search-Based Test Case Selection . . . 131
7.2.1 Feature Modeling for Cyber-Physical Systems Validation . . . 132
7.2.2 Search-Based Test Case Selection . . . 133
7.2.3 Quality Measures . . . 134
7.2.4 Cost Measures . . . 137
7.2.5 Fitness Functions . . . 138
7.3 Empirical Evaluation . . . 139

7.3.1 Research Questions . . . . . . . . . . . . . . . . . . . . . . 139
7.3.2 Experiment Setup . . . . . . . . . . . . . . . . . . . . . . 140
7.3.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . 142
7.3.4 Discussion . . . . . . . . . . . . . . . . . . . . . . . . 146
7.3.5 Threats to Validity . . . . . . . . . . . . . . . . . . . . . 147
7.4 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . 148
7.5 Conclusion and Future Work . . . . . . . . . . . . . . . . . . . 149
**8 Test Case Prioritization . . . . . . . . . . . . . . . . . . . . . . . . 151**
8.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . 151
8.2 Search-Based Test Case Prioritization Approach . . . . . . . . . . . 152
8.2.1 Basic Concepts . . . . . . . . . . . . . . . . . . . . . . . 153
8.2.2 Variability Management for Test Optimization . . . . . . . 154
8.2.3 Weight-based Search Algorithms . . . . . . . . . . . . . . 154
8.2.4 Cost-Effectiveness Measures . . . . . . . . . . . . . . . . . 155
8.2.5 Fitness Function . . . . . . . . . . . . . . . . . . . . . . 158
8.3 Experimental Setup . . . . . . . . . . . . . . . . . . . . . . . . 159
8.3.1 Case Studies . . . . . . . . . . . . . . . . . . . . . . . . 160
8.3.2 Evaluation Metrics . . . . . . . . . . . . . . . . . . . . . 160
8.3.3 Selected Algorithms . . . . . . . . . . . . . . . . . . . . . 162
8.3.4 Algorithms Configurations . . . . . . . . . . . . . . . . . . 162
8.3.5 Artificial Problems and Experiment Runs . . . . . . . . . . 162
8.3.6 Statistical Tests . . . . . . . . . . . . . . . . . . . . . . . 164
8.4 Results and Analysis . . . . . . . . . . . . . . . . . . . . . . . 165
8.4.1 Fault Detection Time (FDT) . . . . . . . . . . . . . . . . . 165
8.4.2 Average Percentage of Faults Detected (APFD) . . . . . . . . 167
8.4.3 Simulation Time . . . . . . . . . . . . . . . . . . . . . . . 169
8.4.4 Functional Requirements Covering Time (FRCT) . . . . . . . 171
8.4.5 Non-Functional Requirements Covering Time (NFRCT) . . 173
8.5 Discussion of the Results . . . . . . . . . . . . . . . . . . . . . 174
8.5.1 Answer to RQ1 . . . . . . . . . . . . . . . . . . . . . . . 174
8.5.2 Answer to RQ2 . . . . . . . . . . . . . . . . . . . . . . . 174
8.5.3 Answer to RQ3 . . . . . . . . . . . . . . . . . . . . . . . 176
8.6 Threats to Validity . . . . . . . . . . . . . . . . . . . . . . . . 176
8.7 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . 177
8.8 Conclusion and Future Work . . . . . . . . . . . . . . . . . . . 178
**PART IV DEBUGGING 181**
**9 Debugging Product Lines . . . . . . . . . . . . . . . . . . . . . . . 183**

9.1 Introduction . . . 183
9.2 Fault Localization Approach . . . 184
9.2.1 Spectrum-based fault localization in Software Product Lines (SPLs) . . . 185
9.2.2 Fault isolation . . . 188
9.2.3 Methodology . . . 189
9.3 Empirical Evaluation . . . 190
9.3.1 Research questions . . . 191
9.3.2 Experimental design . . . 191
9.3.3 Experimental results . . . 196
9.3.4 Discussion . . . 202
9.4 Threats to validity . . . 204
9.5 Related Work . . . 205
9.6 Conclusion . . . 207
**PART V FINAL REMARKS** **209**
**10 Conclusion . . . 211**
10.1 Summary of the Contributions . . . 211
10.1.1 Hypotheses Validation . . . 212
10.1.2 Limitations of the Proposed Solutions . . . 215
10.2 Lessons Learned . . . 217
10.3 Perspectives and Future Work . . . 219
10.3.1 Industry Transfer . . . 219
10.3.2 Application of the Proposed Methods in Specific Domains . 219
10.3.3 Further Research . . . 220
**Bibliographic References . . . 223**
**Appendices . . . 253**
**Appendix A Statistical Tests Results for Test Case Prioritization . . . . 254**

# List of Figures


1.1 General overview of the research methodology . . . . . . . . . . . . . 6

2.1 Classification of the variability of configurable CPSs . . . . . . . . . . 16
2.2 Example of a product line from the mobile phone industry [BSRC10] . 18
2.3 Example of a test system for configurable Cyber-Physical Systems (CPSs) [ASE15a] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.4 Typical structure of a reactive test case for a cruise control of a car example 21
2.5 Example of crossover and mutation genetic operators . . . . . . . . . . 25

4.1 Overview of each contribution of the dissertation, their dependencies and structure of the thesis . . . . . . . . . . . . . . . . . . . . . . . . . . 49
4.2 Overview of the structure of a CPS, corresponding to an example of an UAV (based on [LS15]). ST refers to the sample time of the computations to obtain data from sensors and compute their control tasks in each level . 51
4.3 Feature model for the configurable AR.Drone software . . . . . . . . . 52
4.4 Feature model for the configurable AR.Drone hardware . . . . . . . . . 53
4.5 Overview of the structure of the case study corresponding to the Adaptive Cruise Control (ACC) . . . . . . . . . . . . . . . . . . . . . . . . . . 55
4.6 Feature model for the ACC example . . . . . . . . . . . . . . . . . . . 56
4.7 Overview of the structure of the case study corresponding to the tank . . 58
4.8 Feature model of the industrial tank system . . . . . . . . . . . . . . . 59
4.9 Overview of the structure of the case study corresponding to the DC Engine 61
4.10 Feature model of the DC Engine system . . . . . . . . . . . . . . . . . 62

5.1 Technological overview of ASTERYSCO and its dependencies . . . . . 70
5.2 Test system meta-model. The test stimuli source stimulates Cyber-Physical System Under Test (CPSUT), the context environment model simulates the context in which the CPSUT resides, the test oracle observes the behavior of the CPSUT and decides the test result and the test execution control sequences the test cases. . . . . . . . . . . . . . . . . . . . . . 71

5.3 Test stimuli meta-model of the test system employed to send the generated test data to the CPSUT . . . 72
5.4 Test oracle meta-model of the proposed test system for the decision of the test result . . . 73
5.5 Context environment metamodel of the test system for the simulation of the environment in which the CPS operates . . . 74
5.6 Test execution control meta-model proposed to control the execution of the test cases, the block to the left belongs to the NTG algorithm, the one in the middle to the IDGen algorithm, and the one in the right to the “End mechanism” . . . 75
5.7 Overview of ASTERYSCO’s activities for the generation of a test system instances in Software Process Engineering Meta-model (SPEM). . . . 76
5.8 Taxonomy of the variability for the chosen test system . . . 78
5.9 Test system feature model without being integrated with the CPSUT . . . 81
5.10 CPSUT input model of the AR.Drone Simulink model . . . 82
5.11 Context environment model for the AR.Drone Simulink model . . . 82
5.12 Overview of the tasks of the test system generator for the generation of the generic test system model. . . . 84
5.13 The automatically generated generic test system model . . . 85
5.14 Overview of the processes corresponding to the test system configuration for a specific system instance. . . . 87
5.15 Configured Simulink Model for the configuration $c_i$ of the proposed illustrative example . . . 90
5.16 Detail of the three requirement monitors for the validation of requirement $r_1$ and its requirement validator . . . 92

6.1 Example of three reactive test cases for the cruise control system testing of a vehicle . . . 108
6.2 Representation of a Test Suite for *N* test cases with 3 stimuli signals . . 110
6.3 Crossover operator example for two test suites of 6 and 10 test cases, having the crossover point at the fifth place . . . 111
6.4 Sub-mutation operators at test suite level . . . 112
6.5 Addition sub-mutation operator for the test case level . . . 113
6.6 Removal sub-mutation operator for the test case level . . . 113
6.7 Exchange sub-mutation operator for the test case level . . . 114
6.8 Change of variable sub-mutation operator for the test case level. . . . 114
6.9 Example of the tool support of the test case generation approach for the Unmanned Aerial Vehicle (UAV) case study . . . 115

6.10 Distribution of the *Hypervolume (HV)* indicator for all the runs in each case study . . . 120
6.11 Distribution of the Requirements Coverage objective for all the runs in each case study . . . 120
6.12 Distribution of the Test Execution Time objective for all the runs in each case study . . . 121
6.13 Distribution of the Similarity objective for all the runs in each case study . 121
6.14 Distribution of the Prioritization-aware Similarity objective for all the runs in each case study . . . 122
6.15 Distribution of the running time employed by each algorithm to generate the test suite . . . 122

7.1 Overall overview of the main contribution . . . 131
7.2 Approach overview for the test case selection process . . . 132
7.3 Test feature models for configurable CPSs . . . 133
7.4 Example of a configuration selection . . . 133
7.5 Average percentage of time reduced by each search algorithm in each level as compared with Random Search (RS) with respect to configuration complexity . . . 144

8.1 Overall overview of the proposed approach for test case prioritization . 153
8.2 Example of how to reduce initialization time of reactive test cases with test case prioritization . . . 156
8.3 Distribution of the FDT for all the artificial problems of each case study 166
8.4 Distribution of the APFD for all the artificial problems of each case study 168
8.5 Distribution of the normalized simulation time (i.e., average simulation time per test case) for all the artificial problems of each of the case studies 170
8.6 Distribution of the FRCT for all the artificial problems of each case study 172
8.7 Distribution of the Non-Functional Requirements Covering Time (NFRCT) for all the artificial problems of each case study . . . 173

9.1 Overview of our approach on Spectrum-Based Fault Localization (SBFL) for highly configurable systems . . . 185
9.2 Overview of our approach for fault isolation in highly configurable systems 189
9.3 Overview of the methodology for SPL fault isolation . . . 190

# List of Tables

2.1 Product suite (2-wise) . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3.1 An example showing the suspiciousness value computed using the Tarantula technique . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39

4.1 Mandatory functional requirements for the CPS involving the configurable UAV . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
4.2 Optional functional requirements for the CPS involving the configurable UAV . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
4.3 System requirements of the ACC case study . . . . . . . . . . . . . . . . 57
4.4 System requirements for the Industrial Tank case study . . . . . . . . . 60
4.5 System requirements for the Industrial Tank case study . . . . . . . . . 62
4.6 Main characteristics of each case study . . . . . . . . . . . . . . . . . . 62
4.7 Case studies used in each of the contributions . . . . . . . . . . . . . . 63

5.1 Time required by ASTERYSCO to generate the generic test system as well as different configurations . . . . . . . . . . . . . . . . . . . . . . 94
5.2 Comparison of ASTERYSCO with manual generation of test systems . 95

6.1 Key characteristics of the case studies . . . . . . . . . . . . . . . . . . 116
6.2 Results for the statistical tests related to the Mann-Whitney U-test and the Vargha and Delaney $\hat{A}_{12}$ measure for RQ1 . . . . . . . . . . . . . . . . 118
6.3 Results for the statistical tests related to the Mann-Whitney U-test and the Vargha and Delaney $\hat{A}_{12}$ measure for RQ2 . . . . . . . . . . . . . . . . 119
6.4 Average percentage of improvement of NSGA-II compared with RS for each objective and each case study . . . . . . . . . . . . . . . . . . . . 123

7.1 Selected objective for each test level . . . . . . . . . . . . . . . . . . . 138
7.2 Characteristics of the selected case studies. FR is the number of functional requirements. NFR is the number of non-functional requirements . . . . 140
7.3 Selected weights for each search objective and each case study . . . . . . 141

7.4 Results for the Mann-Whitney U-Test for time and tank case study . . . 143
7.5 Results for the Mann-Whitney U-Test for time and UAV case study . . . 143
7.6 Results for the Mann-Whitney U-Test for fault detection and tank case study 145
7.7 Results for the Mann-Whitney U-Test for fault detection and UAV case study . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 145
7.8 Results for the Mann-Whitney U-Test for requirements coverage and tank case study. RC refers to functional requirements coverage, PWRC refers to pairwise functional requirements coverage, NFRC refers to non-functional requirements coverage and PWNFRC refers to pairwise non-functional requirements coverage. . . . . . . . . . . . . . . . . . . . . . . . . 146
7.9 Results for the Mann-Whitney U-Test for requirements coverage and UAV case study . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 146

8.1 Main characteristics of each case study. Column blocks is referred to the number of blocks of each Simulink model. FR refers to the total number of functional requirements. NFR refers to the total number of non-functional requirements. Feature and constraints refer to the number of feature and constraints related to the feature model of the case study. 160
8.2 Main characteristics of selected product. NFR refers to the total number of non-functional requirements. The column features refers to the number of features that the selected product configuration has. . . . . . . . . . . 160
8.3 Percentage of improvement by the best algorithm for each case study with respect to the remaining algorithms . . . . . . . . . . . . . . . . . . . . . 167

9.1 An example showing the suspiciousness value computed using the Tarantula technique in the Mobile Phone SPL . . . . . . . . . . . . . . . . . 186
9.2 Subject feature models . . . . . . . . . . . . . . . . . . . . . . . . . . . . 191
9.3 Algebraic form of the suspiciousness techniques under evaluation . . . 194
9.4 Types of faults simulated in each experiment (n = number of features in the SPL) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 195
9.5 EXAMF scores obtained using the pairwise product suite in Experiment 1. Best values on each column are highlighted in boldface . . . . . . . . 196
9.6 EXAMF scores obtained using the 3-wise product suite in Experiment 1. Best values on each column are highlighted in boldface . . . . . . . . . . 197
9.7 EXAMF scores obtained using the pairwise product suite in Experiment 2. Best values on each column are highlighted in boldface . . . . . . . . . 197
9.8 EXAMF scores obtained using the 3-wise product suite in Experiment 2. Best values on each column are highlighted in boldface . . . . . . . . . 198

9.9 EXAMF scores obtained using the pairwise product suite in Experiment 3. Best values on each column are highlighted in boldface . . . . . . . . 198

9.10 EXAMF scores obtained using the 3-wise product suite in Experiment 3. Best values on each column are highlighted in boldface . . . . . . . . . 199

9.11 EXAMF scores obtained using the pairwise product suite in Experiment 4. Best values on each column are highlighted in boldface . . . . . . . . 199

9.12 EXAMF scores obtained using the 3-wise product suite in Experiment 4. Best values on each column are highlighted in boldface . . . . . . . . . 200

9.13 EXAMF scores obtained using the pairwise product suite in Experiment 5. Best values on each column are highlighted in boldface . . . . . . . . 200

9.14 EXAMF scores obtained using the 3-wise product suite in Experiment 5. Best values on each column are highlighted in boldface . . . . . . . . . . 201

9.15 Summary of the Results for the Statistical Analysis for the pairwise suite 202

9.16 Summary of the Results for the Statistical Analysis for the 3-wise suite . 202

A.1 Summary for the Mann-Whiteney U-Test Statistical Test results for the FDT metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 255

A.2 Results for the Spearman's rank correlation, which measures the correlation of the FDT metric with respect to the test suite size. Notice that a negative $\rho$ means an improve in the performance of the algorithm with a larger test suite. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 256

A.3 Summary for the Mann-Whiteney U-Test Statistical Test results for the APFD metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 257

A.4 Results for the Spearman's rank correlation test, which measures the correlation of the APFD metric with respect to the test suite size. Notice that a positive $\rho$ means an improve in the performance of the algorithm with a larger test suite. . . . . . . . . . . . . . . . . . . . . . . . . . 258

A.5 Summary for the Mann-Whiteney U-Test Statistical Test results for the Simulation Time. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 259

A.6 Summary for the Mann-Whiteney U-Test Statistical Test results for the FRCT metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 260
A.7 Results for the Spearman's rank correlation test, which measures the correlation of the FRCT metric with respect to the test suite size. Notice that a negative $\rho$ means an improve in the performance of the algorithm with a larger test suite. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 261
A.8 Summary for the Mann-Whiteney U-Test Statistical Test results for the NFRCT metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 262
A.9 Results for the Spearman's rank correlation test, which measures the correlation of the NFRCT metric with respect to the test suite size. Notice that a negative $\rho$ means an improve in the performance of the algorithm with a larger test suite. . . . . . . . . . . . . . . . . . . . . . . . . . . 263

# Acronyms

**ACC** Adaptive Cruise Control

**APFD** Average Percentage of Faults Detected

**ART** Adaptive Random Testing

**AVM** Alternating Variable Method

**BCVTB** Building Controls Virtual Test Bed

**CIT** Combinatorial Interaction Testing

**CPS** Cyber-Physical System

**CPSUT** Cyber-Physical System Under Test

**CTC** Cross-Tree Constraints

**ECU** Electronic Control Unit

**FDC** Fault Detection Capability

**FDT** Faults Detection Time

**FRC** Functional Requirements Coverage

**FRCT** Functional Requirements Covering Time

**FPGA** Field Programmable Gate Array

**GA** Genetic Algorithm

**GUI** Graphical User Interface

**HD** Hamming Distance

**HiL** Hardware-in-the-Loop

**HV** Hypervolume

**MBT** Model-Based Testing

**MiL** Model-in-the-Loop

**MiLEST** Model-in-the-Loop for Embedded System Test

**MOEA/D** Multi-objective Evolutionary Algorithm Based on Decomposition

**NSGA-II** Non-dominated Sorting Genetic Algorithm II

**NSGA-III** Non-dominated Sorting Genetic Algorithm III

**nFRC** Non-Functional Requirements Coverage

**NFRCT** Non-Functional Requirements Covering Time

**PESA-II** Pareto Envelope-based Selection Algorithm II

**PiL** Processor-in-the-Loop

**PTS** Prioritized Test Suite

**RC** Requirements Coverage

**RCT** Requirements Covering Time

**RQ** Research Question

**RS** Random Search

**RTOS** Real-Time Operating System

**RWGA** Randomly-Weighted Genetic Algorithm

**SBSE** Search-Based Software Engineering

**SBST** Search-Based Software Testing

**SiL** Software-in-the-Loop

**SPEA2** Strength Pareto Evolutionary Algorithm 2

**SPEM** Software Process Engineering Meta-model

**SBFL** Spectrum-Based Fault Localization

**SPEA2** Strength Pareto Evolutionary Algorithm 2

**SPL** Software Product Line

**SUT** System Under Test

**TCS** Test Case Similarity

**TET** Test Execution Time

**TPT** Time Partition Testing

**UAV** Unmanned Aerial Vehicle

**UML** Unified Modeling Language

**UTP** UML Testing Profile

**WBGA** Weight-Based Genetic Algorithm

# Part I

# Foundation and Context

Chapter 1

# Introduction

This chapter introduces the main motivation and scope of the research carried out by the Ph.D. student and which problems have been tackled. The selected research methodology is introduced. The main technical contributions are summarized and the publications for each of the technical contribution are highlighted. In addition, the accomplished research activities, such as research stays, are described.

## 1.1 Motivation and Scope of the Research

Cyber-Physical Systems (CPSs) are defined by Lee and Seshia as “an integration of computation with physical processes whose behavior is defined by both physical and cyber parts of the system” [LS15]. The *physical layer* is composed of physical processes, which are a set of many parallel processes [DLSV11] running in continuous time according to laws of physics [LBB15]. The *cyber layer*, which is composed of computational platforms and networks, monitors and controls the physical processes usually with feedback loops [DLSV11]. *Computational platforms* consist of several sensors, actuators, embedded computers and embedded software. The *network fabric* provides communication mechanisms for the computer platforms.

When CPSs have to be customized to clients demands, variability must be efficiently managed during all the development stages, which considerably increases the complexity of the system development and validation. Variability can be understood as configurability (i.e., variability in product space) or as modifiability (i.e., variability in the time space) [TH02]. Configurable CPS development processes can be similar to those processes employed in product line engineering. In product line engineering, two main layers are considered: the *domain engineering* layer and the *application engineering* layer. The *domain engineering* layer involves different engineering tasks that consider variability of the product (i.e., variability in the product, in requirements, etc.). The purpose of the *domain engineering* layer is to create assets that will be reused in the *application engineering* layer. In the *application engineering* layer, the

variability is resolved to create a specific system variant. The assets developed in the *domain engineering* layer are reused to create a variant, reducing the time for the development of different configurations. The assets are typically traced with features in order to model and manage variability. A feature is defined as any increment in product functionality [Bat05]. Features and their possible interactions are typically depicted in a *feature model*. A *feature model* represents all the possible products of a *configurable system* in terms of features and constraints among them [KCH+90]. The feature modeling notations is the one that is most used in industry to manage variability [BRN+13].

*Highly configurable CPSs* can be configured into thousands or even millions of system variants and it is impracticable to test all possible configurations. Consequently, it is infeasible to use real prototypes of the systems. In addition, cost-effectively testing these systems is important to save time while achieving a high overall test quality. Simulation models that represent aspects such as system behavior, environment, structures and properties of CPSs are capable of raising the level of abstraction at which testing is performed [BNSB16]. Employing simulation models permits software engineers to (1) execute more test cases, (2) develop test methods to select scenarios that should also be executed on the deployed system based on the risk level (e.g., fault revealing capability) and (3) specify test oracles for the automatic fault detection [BNSB16]. Moreover, simulation permits testing scenarios that could be dangerous, expensive or even impossible to reproduce employing a real prototype (e.g., safety related scenarios such as the free fall of a lift). However, although the use of simulation methods permits several advantages, testing *configurable CPSs* is still expensive. The challenges of testing configurable CPSs that have been addressed in this thesis are the following ones:

- One of the reasons of testing a specific system variant in the CPSs domain is being expensive is because manually generating a test system for a configuration is an error-prone, non-systematic and time consuming process. The use of a test system permits a systematic validation of simulation models by automating the execution and evaluation of test cases, reusing the most relevant test cases across the different development stages or employing test optimization algorithms to improve the test execution time while maintaining the overall test quality.

- CPSs have been cataloged as untestable systems and traditional testing techniques, such as Model-Based Testing (MBT) or formal methods, are usually expensive, time consuming or infeasible to apply [BNSB16]. This is due to the fact that it is challenging for the traditional testing techniques to capture complex continuous

dynamics and interactions between the system and its environment (e.g., people walking around when automatic braking systems are in use for automotive systems) [BNSB16, MNBB16]. As a result, novel techniques for generating test cases must be proposed. In addition, simulation-based testing has been envisioned as an efficient means to test CPSs in a systematic and automated manner [BNSB16].

- The generated test suite for a *configurable CPS* is usually large. In addition, note that apart from simulating the software itself, many parallel physical processes corresponding to the physical layer have to be simulated [DLSV11]. These physical processes are often modeled with complex mathematical models, which considerably increase the computational needs and as a result, the simulation time required to test each configuration. Moreover, CPSs have to be throughly tested across the different "in the Loop" test levels (i.e., Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL)). Thus, if test cases are not selected in a cost-effective manner, the cost of testing CPSs will be exponentially high. It is worth mentioning that one of the idiosyncrasies of CPS testing is that the three levels (i.e., MiL, SiL and HiL) have their own characteristics, which require defining corresponding objectives for the test case selection.

- Despite selecting a set of test cases, it is also important to prioritize them. This is because the execution of the entire test suite is not always feasible or because it is important to detect faults as fast as possible to begin the debugging process. However, the test prioritization is a non-trivial problem due to the huge search space. Moreover, each test suite must be executed at a specific "in the Loop" test level, and each of these test levels have their own characteristics, which, as in the case of test case selection, requires defining corresponding test prioritization objectives.

- Debugging *configurable systems* is challenging due to the difficulty to find and isolate the faulty features in the *configurable system*. Moreover, even if a suspicious feature or set of features are detected, it might still be difficult to generate small valid products (i.e., satisfying the constraints of the variability model) where the failure is reproduced and the defective assets can be pinpointed. In this context, the recent advances on *configurable systems* testing contrast with the poor support for debugging, which remains as a manual and time-consuming endeavor.

## 1.2 Research Methodology

The selected research method is an iterative model named design and creation [VK04]. The methodology is constituted by five phases, which are also named process steps.

Each process steps has an output that can be understood as the result of the activity related to the process step. Figure 1.1 depicts the overview of the methodology. The process steps are described bellow:

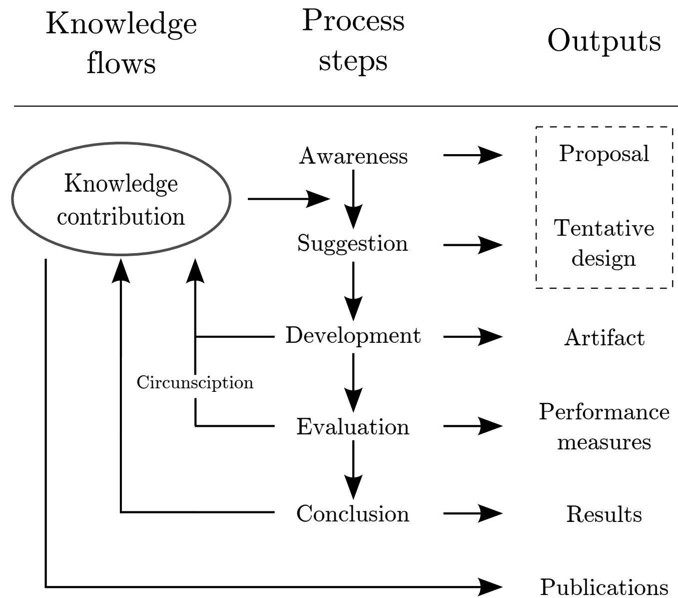



Figure 1.1: General overview of the research methodology

- **Awareness of Problem:** It is the first step, where an interesting problem is detected. The awareness of the problem might come from sources such as new developments in the industries or reading in an allied discipline. The output of this phase is a formal or an informal proposal.

- **Suggestion:** The second step is related to the suggestion; in this phase, a creative step with novel functionalities is envisioned. A tentative design is suggested as an output and likely, the performance of a first version or a prototype of the design could be shown.

- **Development:** The tentative design is further developed and implemented in the third phase, following different techniques for its implementation depending on the artifact to be created. As output, a novel artifact is provided.

- **Evaluation:** The developed artifact is evaluated according to a certain criteria in the evaluation phase. This phase contains analytic sub-phases where hypotheses are tested. The output of this phase will be a set of performance measures.

- **Conclusion:** It is the end of a research cycle or a research effort. It is the last phase of the iterative model and the results from the design and creation model meet the requirements specified in the previous steps. Results are consolidated and the obtained knowledge is detailed and disseminated.

## 1.3 Technical Contributions

The main contributions of this thesis can be summarized as follows:

- A methodology supported by a tool for the automatic generation of test system instances for *configurable CPSs*. Firstly, a test system supporting variability that is specific to CPSs is designed. Secondly, a tool, named ASTERYSCO, is developed which automatically generates the designed test system for configurations of a *configurable CPS* whilst taking a variability model into account.[1] This contribution has resulted in the publication of a set of papers [ASE14a, ASE14c, ASE14b, ASE15b, ASEZ17].

- A test case generation approach for CPSs with tool support developed on top of multi-objective search algorithms that returns a prioritized test suite. First, four corresponding test objectives are defined: (1) test execution time, (2) requirements coverage (3) test case similarity and (4) prioritization-aware similarity. Secondly, with the aim of generating and prioritizing the so-called "reactive test cases", a crossover operator is developed in addition to different mutation operators at two levels (i.e., test suite and test case level).[2] The algorithm is integrated in a tool with the aim of obtaining the characteristics of the *configurable CPS* via *feature models*. The tool is also capable of concretizing the test cases in order to make them executable in Simulink models. The approach is integrated with five pareto-based multi-objective search algorithms and empirically evaluated with four case studies, one of them being an industrial case study. Results showed that Non-dominated Sorting Genetic Algorithm II (NSGA-II) is the best one for solving the proposed problem. This contribution has resulted in the publication of a paper in the IEEE Congress on Evolutionary Computation (CEC 2017) [AWM$^+$17]. Moreover, the extension of the published paper was sent to IEEE Transactions on Industrial Informatics.

- A search-based approach that cost-effectively selects test cases to test system variants of *configurable CPSs* at the MiL, SiL and HiL test levels is proposed.

[1] Information about the developed tool can be found in the following webpage: https://sites.google.com/a/mondragon.edu/asterysco and Chapter 5

[2] Reactive test cases are the types of test cases employed in this thesis, further explained in Chapter 2

Specific test selection objectives are defined for each of the test levels. Based on the selected objectives, three independent fitness functions are selected and integrated within four search algorithms. An empirical evaluation with two case studies and 75 artificial problems is conducted, suggesting that the proposed algorithms can outperform Random Search (RS) in both quality and cost. This contribution was published in the 19th International Conference in Systems and Software Product Line Engineering (SPLC 2016) [AWSE16a].

- A search-based test prioritization approach for *configurable CPSs*, that prioritizes reactive test cases employing search algorithms for the MiL, SiL and HiL test levels is proposed. Specific test prioritization objectives are defined for each of the test levels (e.g., reduction of simulation time). Three independent fitness functions are defined and integrated within four search algorithms (two of them global and two of them local). Four different case studies are employed to evaluate the selected search algorithms. Moreover, the scalability of the search algorithms is assessed using 570 artificial problems. The results showed that local search algorithms performed better than global search algorithms. This contribution was published in the Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016) [AWSE16b]. Furthermore, its extension was sent to the Journal of Systems and Software (JSS).

- A debugging approach for *configurable systems* that employ *feature models* to manage variability. Spectrum-Based Fault Localization (SBFL) techniques are adapted to the Software Product Line (SPL) engineering context, permitting the localization of faulty feature sets. In addition, a fault isolation algorithm that permits the generation of the smallest possible product to reduce debugging efforts is proposed. The approach is evaluated through an empirical evaluation, where ten SBFL techniques are evaluated in nine case studies and under different conditions (i.e., different types of faults). This contribution was sent to the Information and Software Technology (IST) journal.

## 1.4 Publications

Different peer-reviewed publications were published in journals and at conferences during the Ph.D. studies. Notice that some of the conference publication papers are ranked by a raking systems supported by the Spanish Informatics Scientific Society (SCIE (www.scie.es)).[3] The journal publications are scored with their current Journal

[3]http://gii-grin-scie-rating.scie.es/

Citation Report (JCR) score as well as their quartile.

### 1.4.1 Journal Articles

A journal article was published at the Software Quality Journal. Furthermore, by the time this dissertation was submitted, three journal paper were in the second round of review. These three journal included Information and Software Technology journal, Journal of Systems and Software and IEEE Transactions on Industrial Informatics. The journal articles are listed below in chronological order:

- A. Arrieta, G. Sagardui, L. Etxeberria, and J. Zander. "Automatic Generation of Test System Instances for Configurable Cyber-Physical Systems" in Software Quality Journal, 2017, pp. 1041-1083 **JCR:** 1.816. **Q2**.

- A. Arrieta, S. Segura, U. Markiegi, G. Sagardui, L. Etxeberria. "Spectrum-Based Fault Localization in Software Product Lines" in Information and Software Technology **JCR:** 2.694. **Q1**. Second round of review.

- A. Arrieta, S. Wang, G. Sagardui, L. Etxeberria. "Search-Based Test Case Prioritization for Simulation-Based Testing of Cyber-Physical System Product Lines" in Journal of Systems and Software **JCR:** 2.444. **Q1**. Second round of review.

- A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, L. Etxeberria. "Employing Multi-Objective Search to Enhance Reactive Test Case Generation and Prioritization for Testing Industrial Cyber-Physical Systems" in IEEE Transactions on Industrial Informatics **JCR:** 6.764. **Q1**. Second round of review.

### 1.4.2 International Conferences

A total of eleven publications were achieved at international conferences, including ETFA, VALID, SPLC, GECCO, CEC and ISSRE. The publications are listed below:

- A. Arrieta, G. Sagardui, and L. Etxeberria. "Towards the Automatic Generation and Management of Plant Models for the Validation of Highly Configurable Cyber-Physical Systems" Proceedings of 2014 IEEE 19th Conference on Emerging Technologies & Factory Automation (ETFA), 2014, pp. 1-8; **Ranking_SCIE: B**

- A. Arrieta, G. Sagardui, and L. Etxeberria. "A Model-Based Testing Methodology for the Systematic Validation of Highly Configurable Cyber-Physical Systems" VALID 2014: The Sixth International Conference on Advances in System Testing and Validation Lifecycle, 2014, 66-72

- A. Arrieta, G. Sagardui, and L. Etxeberria. “A Configurable Test Architecture for the Automatic Validation of Variability-Intensive Cyber-Physical Systems” in VALID 2014: The Sixth International Conference on Advances in System Testing and Validation Lifecycle, 2014, pp.79-83

- A. Arrieta, G. Sagardui, and L. Etxeberria. “Test control algorithms for the validation of cyber-physical systems product lines,” in Software Product Line Conference (SPLC’2015), 2015, pp. 273-282; **Ranking_SCIE: A-**

- A. Arrieta, G. Sagardui, and L. Etxeberria. “Variability in Test Systems: Review and Challenges” in VALID 2015: The Seventh International Conference on Advances in System Testing and Validation Lifecycle, 2015 , pp. 15-22

- A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria. “Test Case Prioritization of Configurable Cyber-Physical Systems with Weight-Based Search Algorithms” in GECCO2016: Genetic and Evolutionary Computation Conference, 2016, pp. 1053-1060; **Ranking_SCIE: A**

- A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria. “Search-Based Test Case Selection of Cyber-Physical System Product Lines for Simulation-Based Validation” in SPLC2016: Software Product Line Conference, 2016, pp. 297-306; **Ranking_SCIE: A-**

- A. Arrieta, S. Wang, U.Markiegi, G. Sagardui, and L. Etxeberria. “Search-Based Test Case Generation for Cyber-Physical Systems” in CEC2017: IEEE Congress on Evolutionary Computation, 2017, pp. 688-697; **Ranking_SCIE: A-**

- U. Markiegi, A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria. “Search-Based Product Line Fault Detection Allocating Test Cases Iteratively” in SPLC2017: Software Product Line Conference, 2017, pp. 123-132; **Ranking_SCIE: A-**

- G. Sagardui, L. Etxeberria, J. Agirre, A. Arrieta, C.F. Nicolás, and J.M. Martín. “A Configurable Validation Environment for Refactored Embedded Software: an Application to the Vertical Transport Domain” in ISSRE 2017 (Industry Track): IEEE International Symposium on Software Reliability Engineering, 2017, **Ranking_SCIE: A**

- L. Etxeberria, F. Larrinaga, U. Markiegi, A. Arrieta, G. Sagardui. "Enabling Co-Simulation of Smart Energy Control Systems for Buildings and Districts" in ETFA2017: IEEE 22nd Conference on Emerging Technologies and Factory Automation, 2017, **Ranking_SCIE: B**

### 1.4.3 Workshops and National Conferences

In addition to international conferences, four national conference paper were published, including JCE and JISBD. Furthermore, a workshop paper was published at ECMSM 2017 in collaboration with an industrial partner. These papers are listed below:

- A. Arrieta, G. Sagardui, and L. Etxeberria. "A Comparative on Variability Modelling and Management Approaches in Simulink for Embedded Systems" in V Jornadas de Computación Empotrada, 2014, pp. 26-33

- A. Arrieta, G. Sagardui, and L. Etxeberria. "Cyber-physical systems product lines: Variability analysis and challenges," in VI Jornadas de Computación Empotrada, 2015.

- A. Arrieta, U.Markiegi, and L. Etxeberria. "Towards Mutation Testing of Configurable Simulink Models: a Product Line Engineering Perspective" in JISBD2017: XXII Jornadas de Ingeniería del Software y Bases de Datos, 2017

- X. Perez, O. Berreteaga, L. Etxeberria, A. Arrieta, and U. Markiegi. "Modeling Systems Variability with Delta Rhapsody" in JISBD2017: XXII Jornadas de Ingeniería del Software y Bases de Datos, 2017

- G. Sagardui, J. Agirre, U. Markiegi, A. Arrieta, C.F. Nicolás, and J.M. Martín. "Multiplex: A Co-Simulation Architecture for Elevators Validation" in ECMSM 2017: IEEE International Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics, 2017, 1-6

## 1.5 Related Activities

In addition to attending the conferences of the aforementioned conference publications, the Ph.D. student has accomplished other activities that helped him in his training as a researcher. These activities have included talks at industrial conferences as well as summer schools, service to the community, three research stays and participation in European projects.

### 1.5.1 Talks

The work developed during this dissertation was also disseminated in the following forums:

- 9th Graz Symposium Virtuellez Fahrzeug, (GSVF 2016)[4]. GSVF is an industrial symposium for the automotive industry. The presentation given by the Ph.D. student, entitled “Efficient virtual testing of variability-intensive automotive cyber-physical systems”, had as an objective to disseminate the work carried out on test case prioritization as well as automatic test system generation to industrial practitioners from the automotive domain.

- First international Summer School on Search-Based Software Engineering, Cádiz, Spain. The talk given by the Ph.D. student, entitled “Test Optimization of Configurable Cyber-physical Systems with Search Algorithms”, aimed to obtain feedback related to the developed search-based algorithms for test prioritization and selection of *configurable CPSs*.

- CPS Summer School, Sibiu, Romania.[5] The talk given by the Ph.D. student, entitled “Test optimization of Cyber-Physical System product lines”, consisted of presenting and disseminating the work carried out in the context of test optimization of *configurable CPSs*.

### 1.5.2 Research Stays

Visiting other institutions to collaborate with relevant scientist is part of the activities researchers carry out during their career. During the Ph.D., three research stays were accomplished by the Ph.D. student. Two of these research stays were in Simula Research Laboratory, Norway, where the Ph.D. student carried out a total of three months of research stay, which is one of the requirements for having access to the international Ph.D. mention. The other one was at the University of Sevilla, Spain. The following activities were carried out during the research stays:

- First research stay (Simula Research Laboratory, mid of April 2015 to mid of June 2015): The objective of the research stay was to acquire feedback from expert researchers in the field of CPS testing. In addition, given the expertise field of the research group of Simula, the Ph.D. student took advantage for learning about the field of search-based software engineering. The results of the collaboration with this research stay were two conference papers [AWSE16b, AWSE16a].

- Second research stay (University of Sevilla, mid of November 2015 to mid of December 2015): The objective of the research stay was to learn about SPL reasoning tools and propose a debugging methodology for the *configurable systems*

[4] http://www.gsvf.at/

[5] http://into-cps.au.dk/summerschool/

context. The results of the collaboration includes the debugging methodology for *configurable systems*, which was submitted to the Information and Software Technology (IST) journal.[6]

- Third research stay (Simula Research Laboratory, mid of April 2016 to mid of May 2016): The objective of the research stay was to extend the previously published GECCO 2016 paper [AWSE16b] to a journal version. In addition, the test case generation algorithm (Chapter 6) was designed, which was later implemented at the home university and published at CEC 2017 [AWM$^{+}$17], and its extension submitted to the IEEE Transactions on Industrial Informatics.

### 1.5.3 Service

Researchers are often involved in peer-reviewing articles or organizing conferences and workshops. As part of his training, the Ph.D. student has been involved in the following activities as a service to the research community:

- Reviewer for IEEE Transactions on Industrial Informatics.
- Program chair of the first IEEE International Workshop on Employing Computational Intelligence Techniques for Testing and Validating Complex CPSs (CITest_CPS 2017), together with Dr. Shuai Wang (Simula Research Laboratory) and Dr. José Francisco Chicano (University of Malaga).[7] The workshop was co-located with the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) conference.
- Program committee member of the 13th Workshop on Advances in Model-Based Testing (A-MOST), co-located with the 10th IEEE International Conference on Software Testing, Verification and Validation (ICST 2017).[8]
- Program committee member of the VALID 2015, VALID 2016 and VALID 2017 conferences.[9]
- Sub-reviewer at different conferences: SPLC 2015, MODELS 2015 and JISBD 2016

[6]By the time this dissertation was submitted, the status of the paper was in the second round of revision

[7]http://paris.utdallas.edu/CITest_CPS17/

[8]http://a-most17.zen-tools.com/

[9]http://www.iaria.org/conferences2017/VALID17.html

### 1.5.4 Proposals Writing

Writing proposals for funding is one of researchers' key activities. The Ph.D. student has been involved in the proposal writing of two European projects that are related to this dissertation. The first one was the "TESTOMAT" project, part of the ITEA-3 call. The second one was the "HiFi-Elements" project, which is inside the Horizon 2020 call inside the Green Vehicle call. Both projects were accepted by the European commission.

## 1.6 Document Structure

The thesis is structured as follows. The first part of the thesis corresponds to the Foundation and Context. Chapter 1 introduces the main motivation of the thesis, the employed research methodology, the contributions, the achieved publications and the activities accomplished by the Ph.D. student. Basic background as well as terminology used during the rest of the document is provided in Chapter 2. Chapter 3 gives an overview of the state of the art and highlights the most relevant studies related to this thesis. The theoretical framework is explained in Chapter 4, including the research objectives, the research hypotheses, an overview of the proposed solutions and the employed case studies.

The second part corresponds to automation. Chapter 5 provides the method we propose for generating test system instances for the automated validation of *configurable CPSs*. Chapter 6 proposes a novel test case generation approach based on multi-objective search algorithms for testing CPSs.

The third part corresponds to the optimization of the test process for *configurable CPSs*. To this end, first, our proposal for selecting test cases for configurable CPSs is proposed in Chapter 7. Second, we propose a test case prioritization method based on search algorithms for prioritizing test cases in Chapter 8.

The fourth part, which is composed of a single Chapter (i.e., Chapter 9), proposes a debugging methodology for the context of configurable systems (e.g., SPLs or *configurable CPSs*). In this chapter, we adapt a technique named SBFL to the product line engineering context and we provide a fault isolation method.

Finally, in the final remarks part in Chapter 10, we summarize the contributions of the thesis, we validate the hypotheses, we discuss the main limitations of the project and we provide a set of lessons learned. Furthermore, we propose and discuss future directions.

Chapter 2

# Technical Background

The goal of this chapter is to familiarize the reader with the areas scoped by the thesis. First, background related to configurable Cyber-Physical Systems (CPSs) is provided in Section 2.1. Second, feature models are explained in Section 2.2. Third, Section 2.3 explains simulation-based testing and related techniques. General practices for testing configurable systems is explained in Section 2.4. Lastly, background related to search-based software testing is provided in Section 2.5.

## 2.1 Configurable Cyber-Physical Systems

As the use of CPSs increases in our society, users demand different needs, which results in a personal customization of these systems. This means that as the variability of the system increases, the trend of them to be configurable increases, appearing in different fields. In Figure 2.1 we provide a taxonomy with the variability points of CPSs considered for simulation-based testing. As for the *physical layer*, we consider variability in several points of the CPS (e.g., an Unmanned Aerial Vehicle (UAV)). The mechanical elements are highly exposed to variability, which has to be taken into account since a change in a mechanical element might change the dynamics of the system. For instance, changing mechanical elements, such as, changes in the shapes, sizes or the material, might have a direct impact on the system weight as well as on its center of gravity. Changing both these factors in turn affects the dynamics of the CPS, which can lead to a completely new behavior. Another variability point in the physical layer consists of the energy supply system. Energy supply in CPSs is a challenge and often the system must optimize its consumption [WSYL11]. However, note that a longer duration battery might be bigger and heavier, which can have a direct influence in the dynamics of the system.

Regarding the *cyber layer*, we consider variability in several points too. For instance, a CPS can deal with different functionalities. Most of these functionalities are implemented in software, and thus, variability must be considered in software. Another

variability point that has to be considered is related to the sensors [VLOdbH+14] (variability in points such as sensitivity, ranges they can measure, response time, parametric values, etc.) and actuators (such as communication, ranges they can work in, response time or robustness properties, etc.). In addition to sensors and actuators, variability in the number of platforms and in the network fabric in charge of communicating the computational platforms is also supported in the approaches presented in this dissertation. Concerning the network fabric, different types of communications can be considered (e.g., Ethernet, CAN bus, 802.11b WLAN, etc.). It is important to highlight that the variability of some parts might require considering variability in other parts of the system. For instance, the variability related to the physical layer might have a direct influence on the dynamics of the system. This issue might also require some changes in the cyber layer, where configurations in the software (such as new functions or changes in certain parametric values), or even in the hardware (such as sensors with better sensitivity or actuators that can cope with heavier weights), might be required. Some functionalities might also require changes in both the software as well as the hardware. Moreover, there are specific hardware elements that require the software to be adapted. For instance each specific sensor needs its driver and each actuator might require a specific controller.

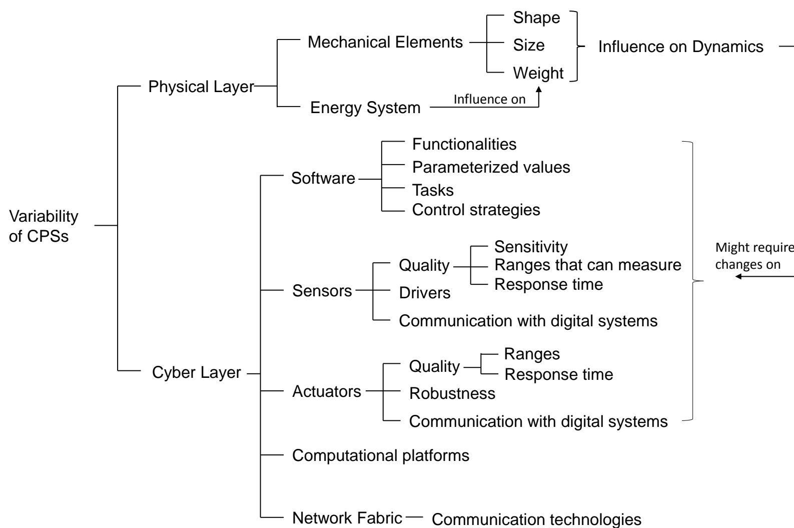



Figure 2.1: Classification of the variability of configurable CPSs

## 2.2 Feature Models

A feature model is a common variability modeling notation for managing variability of configurable systems, such as configurable CPSs or Software Product Lines (SPLs). SPL engineering focuses on the systematic development of related software products from a set of reusable features [CN01]. A *feature* is defined as any increment in product functionality [Bat05]. Features and their possible interactions are commonly depicted in a feature model. A *Feature Model* represents all the possible products of a SPL in terms of features and constraints among them [KCH+90]. In this context, a *product* is a set of features satisfying all the constraints of the *Feature Model*. Figure 2.2 depicts a sample *Feature Model* representing a simplified product line of mobile phones.

Feature models are the de-facto standard for modeling commonality and variability in SPLs [BSRC10, KCH+90]. In fact, according to Berger et al., it is the notation that is most used in industry to manage variability [BRN+13]. Structurally, a feature model is a tree-like structure in which nodes represent features and edges represent constraints among the features. Each feature is related to a set of *assets* that implement the feature's functionality, i.e., code, documentation, test cases, etc. A *product* is a set of features satisfying the constraint of the feature model. Products are implemented by integrating the assets of the features that are part of them.

Child features can be divided into mandatory and optional features. *Mandatory* features must be included in all the products including its parent feature, e.g., all mobile phones in Figure 2.2 must provide support for `Calls`. *Optional* features can be optionally included in those products containing its parent feature, e.g., phones can optionally provide support for `GPS`. Additionally, child features can be grouped into *alternative* and *or* relationships. A set of child features has an *alternative* relationship with their parent feature when only one of them can be selected when its parent feature is part of the product, e.g., phones can only support one type of screen: `Basic`, `Colour` or `High resolution`. Finally, in *or* relations at least one of the child features must be included in the products containing its parent feature, e.g., phones supporting media content must include the features `Camera`, `MP3` or both of them.

In addition to the parental relationships among features, feature models can include cross-tree constraints among features. Typical constraints model dependencies such as "A *requires* B", indicating that the products containing the feature A must also include the feature B, or "A *excludes* B", indicating that the features A and B cannot be part of the same product, i.e., they are incompatible features. In the example, phones including the feature `Camera` must include support for a `High resolution` screen.

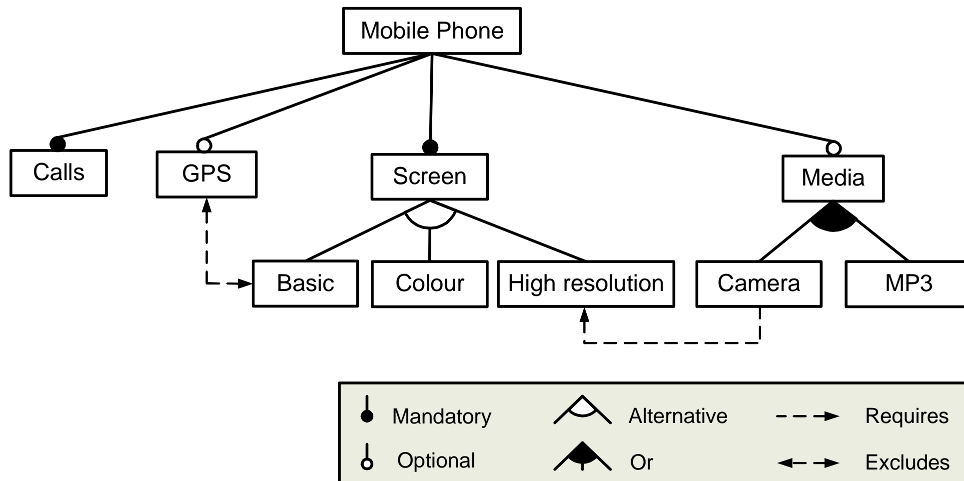



Figure 2.2: Example of a product line from the mobile phone industry [BSRC10]

The analysis of feature models deals with the automated extraction of information from feature models. The analysis is performed in terms of *analysis operations*. Among others, these operations allow finding out whether a feature model is void (i.e. it represents no products), whether it contains errors (e.g., dead features) or what is the number of products represented by the model. Catalogues with up to 30 different analysis operations on feature models have been reported in the literature [BSRC10]. A number of tools support the analysis of feature models including FaMa [fam13a], SPLAR [MBC09a] and FeatureIDE [TKB+14]. In the following, we define some of the basic terms from the product line engineering domain. For the definitions, let $F$ be the set of features in a feature model.

- *Feature set*. Non-empty set of features $S$, $S \subseteq F$, with $|S| \geq 1$, e.g., S= {`Media`, `MP3`}.

- *Configuration*. A configuration is a 2–tuple of the form ($S$,$R$) such that $S, R \subseteq F$ being $S$ the set of features to be selected and $R$ the set of features to be removed such that $S \cap R = \varnothing$ and $S \cup R = F$. If $S \cup R \subset F$ the configuration is called *partial configuration* [BSRC10]. For instance, the following is a partial configuration of the model in Figure 2.2: (S,R) = ({Media,MP3},{GPS}).

- *Product*. A product is equivalent to a configuration where only selected features are specified and omitted features are implicitly removed [BSRC10], e.g., see products in Table 2.1.

- *Product suite*. Set of products under test. Table 2.1 shows the set of products obtained when applying 2-wise testing to the model in Figure 2.2. The product suite

is reduced from 13 products (total number of products in the SPL) to 8 products containing all the possible feature pairs, 41 in total.

- *Core features*. These are the set of features included in all the products of the SPL [BSRC10]. In the example the core features are `Mobile phone`, `Calls` and `Screen`.

| ID | Product |
|---|---|
| P1 | {MobilePhone, Screen, Calls, High resolution} |
| P2 | {MobilePhone, Screen, Calls, Colour, Media, MP3} |
| P3 | {MobilePhone, Screen, Calls, Colour, GPS} |
| P4 | {MobilePhone, Screen, Calls, High resolution, Media, MP3, Camera} |
| P5 | {MobilePhone, Screen, Calls, High resolution, Media, Camera, MP3, GPS} |
| P6 | {MobilePhone, Screen, Calls, Basic, Media, MP3 } |
| P7 | {MobilePhone, Screen, Calls, Basic} |
| P8 | {MobilePhone, Screen, Calls, High resolution, Media, Camera} |

Table 2.1: Product suite (2-wise)

## 2.3 Simulation-based Testing of Cyber-Physical Systems

As developing a CPS prototype is often costly, the use of simulation-based testing for CPSs is increasing. Simulation is one of the most frequently used techniques for testing system models in domains where software interacts with physical processes such as CPSs [MNBB16]. CPS models are heterogeneous due to encompassing software, networks and parallel physical processes. These models therefore provide an accurate representation of the real world and continuous dynamics [MZ16, MNBB16]. Different simulation tools have been employed to test CPSs using simulation including MATLAB/Simulink [MNBB16, MSB+14, ASEZ17], and a combination of System C and Open Dynamics Engine [MBED12]. Moreover, as CPSs are involved in several engineering domains, it is very common to use different simulation environments integrated by a co-simulation engine. Although this integrated CPS model presents an important advance in terms of engineer flexibility, it increases the test and simulation time due to requiring the transfer of data and synchronization between tools. In addition, CPSs simulation often involves the use of complex mathematical models

to represent the continuous dynamics of the physical layer. Consequently, computer resources are allocated by the solvers used by the simulation tools.

### 2.3.1 Test Systems

A *test system* is a set of components that interact with the objective of testing the *System Under Test (SUT)*. The complexity of a *test system* can vary depending on the overall test objectives and type of testing. The organization of the group of components comprising the *test system* is called the test architecture, which specifies the interaction among the different elements of the *test system* and the SUT. A *test system* can include several options, such as test oracles, components that generate input data in the form of test cases, and other testing resources, and it is a necessary artifact in test and validation activities so that verification and validation activities can be systematic. Test cases are part of the *test system* and provide information about the test execution. In Model-Based Testing (MBT), test cases are automatically generated either from the system model, i.e., from the model of the *SUT* or from a test model [ZNSM11]. When the test cases are executed, the test results have to be determined. This is typically performed by other elements of the *test system*, such as test oracles, which are mechanisms that analyze the *SUT* output and are able to decide the test result [ZNSM11]. Figure 2.3 shows an example of a test system for testing configurable CPSs proposed in [ASE15a].

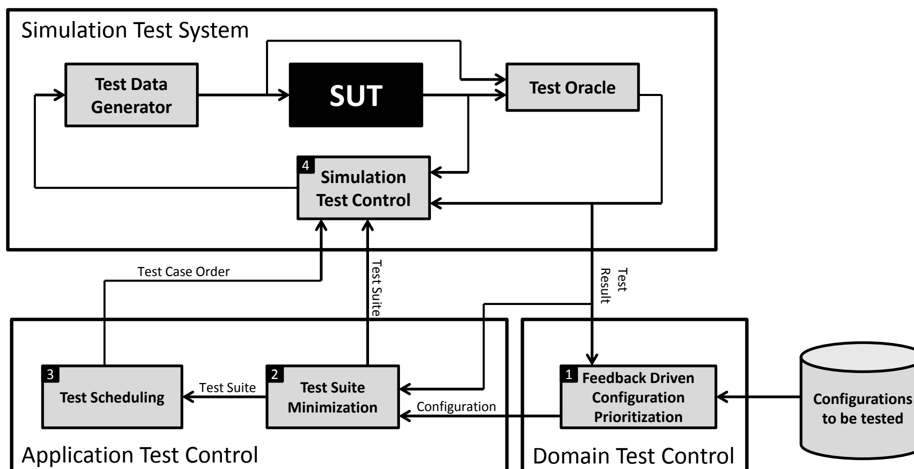



Figure 2.3: Example of a test system for configurable CPSs [ASE15a]

### 2.3.2 Reactive Test Cases

Test reactivity is known as the capacity of the test system to react on the outputs of the System Under Test (SUT), verdicts data, or internal signals of the test oracle [ZN07]. Accordingly, reactive test cases are a set of stimulation signals that excite a system and observe some predefined properties (e.g., SUT outputs, time, etc.) to react on them and change the stimulation signals to other values [AWSE16b]. These systems are typically employed to test embedded systems [MH15] or CPSs [AWSE16b] of different domains such as in the automotive industry [ZN08]. Reactive test cases are usually employed to test functional requirements at system level [ZN08]. Typically, reactive test cases can be modeled in a state chart similar to the one depicted in Figure 2.4. A reactive test case can be structured into three main steps: (1) test case initialization, (2) execution and (3) finalization.

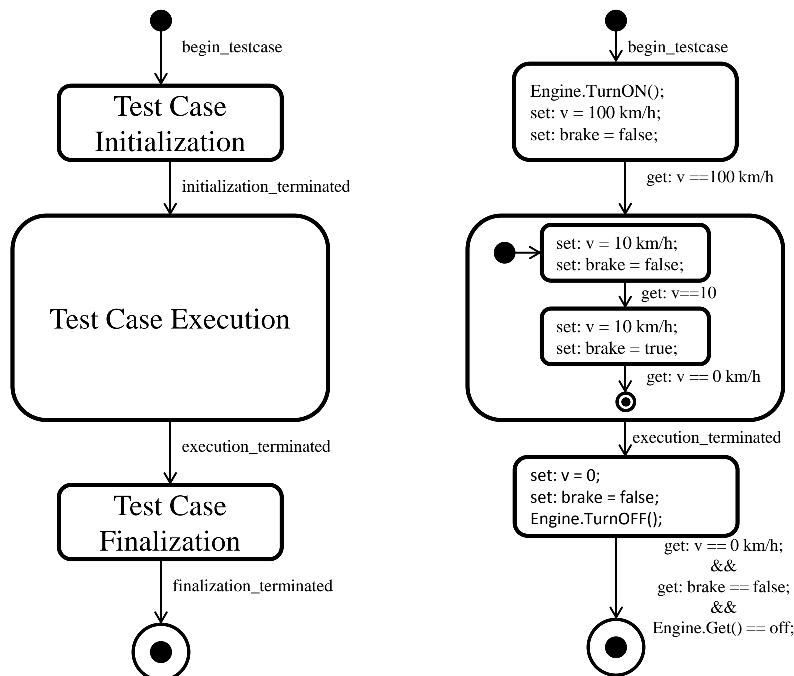



Figure 2.4: Typical structure of a reactive test case for a cruise control of a car example

### 2.3.3 Test Levels

Modeling and simulation tools are typically used at early validation stages to determine an approximate behavior of CPSs. These tools are typically used by test engineers to test different parts of the CPS at early validation stages, when a real prototype is still

not available. Simulation-based validation allows for testing CPSs at different levels: Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), Processor-in-the-Loop (PiL) and Hardware-in-the-Loop (HiL). Different examples of testing CPSs employing different levels are presented in the current literature (e.g., MiL and SiL level [ASE14b, ASE15a] or HiL level [ELM11, KFK14]). These test levels can be used to validate CPSs at the system level with different objectives.

- The first test level corresponds to MiL, which is the basic simulation that engineers use to analyze the embedded software model with the physical layer of the CPS [SH09]. At this test level, the simulation is performed with high precision, as the computations use floating-point arithmetic to obtain the simulation results as a reference for the following testing stages [SH09].

- The second test level corresponds to SiL. The embedded software model is replaced with an executable object code (e.g., a *.dll). Unlike the MiL test level, the SiL test level uses fixed-point computations, which represent the computations done in the final target system [SH09].

- After the SiL test configuration, the PiL uses real object code, which is compiled and deployed on the real target processor. This communicates with the simulation tool in the computer to obtain data from the physical layer [SH09]. At the PiL test level, the main objective is to detect potential inconsistencies introduced by the code compilation tool [SH09].

- The last test level refers to the HiL. In this case, the embedded software source code is integrated with the Electronic Control Unit (ECU), as well as the real-time infrastructure (e.g., drivers, Real-Time Operating System (RTOS)) [SH09]. The physical layer, which was previously executed in the computer, is encapsulated in an embedded device (e.g., a Field Programmable Gate Array (FPGA)) with the objective of accomplishing a real-time simulation. HiL simulations are typically used to validate performance requirements as well as timing constraints [SH09]. Different simulation tools allow for the validation of CPSs at the HiL level (e.g., PTIDES [ELM11]).

## 2.4 Configurable Systems Testing

When testing variability-intensive systems, the number of configurations that the system can be set to is a factor that has to be considered at validation stages. This number is usually too high, and as a result, it is infeasible to test all the configurations.

To this end, in the product line engineering community two levels are differentiated: (1) domain engineering level and (2) application engineering level. While the domain engineering level considers variability of the whole configurable system, the application engineering level focuses on specific product configurations. From the testing perspective, the domain engineering level considers which products must be tested and their order, whereas the application engineering level considers how a specific product configuration is tested (e.g., which test cases must be executed in each of the products). The following steps are typically undertaken when testing a configurable system:

- Generation of relevant configurations to test: As testing all the configurations is infeasible, it is important to select relevant configurations that can ensure certain test coverage for the whole variability-intensive system. The most common approach for selecting these configurations is employing Combinatorial Interaction Testing (CIT) algorithms [KKLH09]. These algorithms usually parse a feature model and generate configurations that cover all possible feature interactions using SAT solvers [PSK+10]. For instance, a pairwise configuration generation criteria will ensure that all the feature pairs of the variability-intensive system are covered. This way, it can be ensured that the system does not fail due to the interaction of two features. This test optimization part corresponds to the domain engineering layer.
- Prioritization of the order in which the configurations are tested: once the relevant configurations are generated, it is also important to prioritize the order in which these are tested. For instance, Sanchez et al. demonstrated that in the context of SPLs, the product complexity helped increase the fault detection rate as compared to other prioritization criteria (e.g., configuration size or dissimilarity of configurations) [SSRC14a]. As in the previous case, this test optimization part corresponds to the domain engineering layer.
- Generation of the test system: Once selected the configuration that must be tested, the test system that includes test cases, test oracles and other sources must be generated. Manually generating the test system can be a time-consuming and error-prone process, and thus, an automatic test system generation is needed to ensure a systematic generation of the test system. This part corresponds to the application engineering layer.
- Test case selection and prioritization: Taking into account the selected configuration it is important to select which are the test cases that must be executed [WAG13, WAY+16]. Studies in the field of SPL engineering have also proposed

several search algorithms for test suite minimization (e.g., [WAG13, WAG15]). In addition, after selecting relevant test cases, test case prioritization helps improve the fault detection rate [WBA+14]. This test optimization level corresponds to the application engineering layer.

## 2.5 Search-Based Software Testing

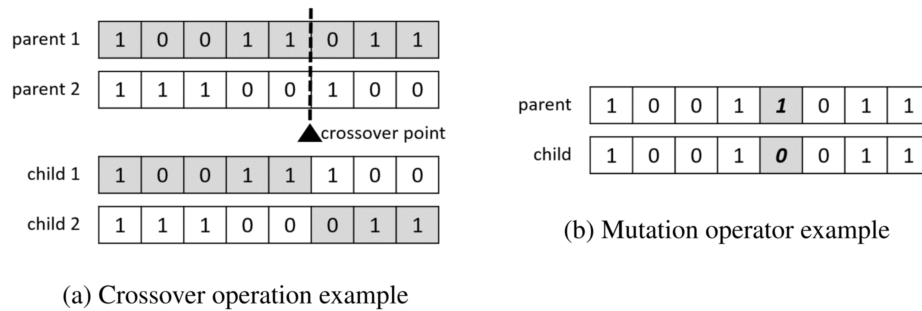
Search-Based Software Engineering (SBSE) aims at converting a software engineering problem in a mathematical optimization problem. SBSE has been applied into many software engineering problems (e.g., requirements engineering [GR04]). Testing is the first software engineering activity where search algorithms were applied [MS76]. Search-Based Software Testing (SBST) aims at reformulating a software testing problem (e.g., test case prioritization) as an optimization problem, which is later solved by search algorithms [WAG13, WBA+14, ABHPW10, AF14, FA13]. Search algorithms aim at searching for optimal solutions by mimicking natural phenomenon such as natural evolution process [Bro12].

To guide the search, a fitness function needs to be defined. A fitness function is the objective function that is used to assess the solutions (also known as individuals or chromosomes). Each solution (i.e., individuals) is composed of genes, which represent units for the solution [Wan15].

### 2.5.1 Local Search Algorithms

Local search algorithms aim at optimizing solutions by performing local changes in an initial solution [MH97]. These changes produce some improvements each time according to a fitness function until a local optimum is found [MH97].

One of the most well-known local search algorithms is greedy. A greedy algorithm makes the locally optimal choice at each time, which allows for finding a local optimum in a fast way. Greedy algorithms have shown to be effective at solving several software engineering problems, such as CIT [CDS08]. Different greedy algorithms exist, and in SBSE the total greedy and the additional greedy are the most widely used ones. Total greedy follows the “next best” search philosophy, which builds solutions by sorting the elements in a descending order (or ascending, if the objective is to minimize the fitness function) [LHH07]. On the contrary, additional greedy combines feedback from previous selections [LHH07]. It iteratively builds solutions, selecting the best element at each iteration. In this thesis, when we refer to greedy algorithm, we refer to the additional greedy algorithm.




(a) Crossover operation example

(b) Mutation operator example

Figure 2.5: Example of crossover and mutation genetic operators

Another local search algorithm is the Alternating Variable Method (AVM). AVM has been shown to be an effective local search algorithm in SBSE [KMS15, MK16]. The main idea of the AVM algorithm is to randomly generate a solution $S = (s_1, s_2, ..., s_N)$, and later, each variable in $S$, (i.e., $s_i$ (where $1 \leq i \leq N$)) is individually optimized with a local search algorithm and according to a specific fitness function [KMS15]. This process is iteratively repeated until either the search budget is consumed or there is no further local search improvements [KMS15].

### 2.5.2 Global Search Algorithms

Global search algorithms are those algorithms capable of dealing with the global optimization of a fitness function.

Genetic Algorithms (GAs) are the most well known global search algorithms. After defining the corresponding objective function (i.e., fitness function), an initial population is randomly generated. Later, until the search budget is consumed, GA applies three operators (1) selection, (2) crossover and (3) mutation. The selection operator selects individuals to be involved in the reproduction. The selection is typically guided by the fitness function, so that individuals with good fitness values can have higher chances to survive [CGF+17]. Different selection strategies can be employed for this operator, such as rank-based, elitism or tournament selection. The crossover operator recombines selected individuals, as shown in Figure 2.5a, where the crossover between two individuals with a single point crossover is performed. The mutation operator changes the genes in each individual with certain probability, typically equal to 1/N, being N the number of genes in the chromosome [CGF+17]. Figure 2.5b shows an example of a mutation.

### 2.5.3 Stochastic Algorithms

Random Search (RS) is a stochastic algorithm commonly used as a baseline algorithm [WAG15]. This algorithm randomly generates solutions from the search space and the best one is chosen. A variant from RS for test case generation is Random Testing, which incrementally builds test suites [CGF+17]. Random testing generates test cases individually. When a generated test case improves coverage (or other objectives), it is included in the test suite, otherwise it is discarded [CGF+17]. Adaptive Random Testing (ART) [CLM04] is another variant of RS for test generation. In this case, a selected set of test cases is generated together with a candidate set of test cases. From the candidate set of test cases the most dissimilar test case is selected as compared with each of the test cases in the selected set. The selected test case is included in the selected set, ensuring this way diversification when generating test cases.

### 2.5.4 Multi-Objective Search Algorithms

Many software engineering problems contain multiple conflicting objectives in order to be optimized. For this reason, in the last few years, the use of Pareto-based multi and many-objective search algorithms has been investigated to solve SBSE problems [Har11, YH07, HLL+16, WBA+14, BANBS16, WAY+16, SPH16, PKT17, CGF+17]. Unlike single-objective search algorithms, Pareto-based search algorithms produce a set of non-dominated solutions (also known as Pareto front), from which the user can select one or more solutions based on their specific needs [WAY+16].

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) [DPAM02] is one of the most well-known multi-objective search algorithms. As common GAs, NSGA-II generates an initial set of random solutions. These solutions later evolve through a series of generations to find better solutions [PKT17]. To this end, new solutions (i.e., offsprings) are created by using the crossover and mutation operators. To generate the offsprings, parents are selected with a selection operator, which uses Pareto optimality to give higher probability to select non-dominated solutions [PKT17]. To preserve solutions forming the non-dominated solutions in the next generation, NSGA-II uses a fast non-dominated sort algorithm (i.e., elitism) [PKT17].

Apart from NSGA-II, other typical multi-objective algorithms include Strength Pareto Evolutionary Algorithm 2 (SPEA2) [ZLT+01] and Pareto Envelope-based Selection Algorithm II (PESA-II) [CJKO01]. However, these algorithms have shown scalability problems with more than three objectives [PKT17]. To solve these scalability problems, in the last few year, many-objective algorithms have been proposed, which are designed for more than three objectives, including Non-dominated Sorting

Genetic Algorithm III (NSGA-III) [DJ14] and Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [ZL07].

Chapter 3

# State of the Art

In this chapter, we give an overview of the state of the art and highlight the most relevant studies related to this thesis. Furthermore, a critical analysis of of the state of the art in testing configurable Cyber-Physical Systems (CPSs) is performed, which aims at finding research opportunities.

## 3.1 Testing Product Lines

Testing product lines has gained high attention from the research community in the last few years. Several literature reviews and mapping studies reveal that most of the work on product line engineering testing focuses on the domain engineering level [LHLE15, NdCMM+11, dCMMCDA14, ER11, HJK+14]. This is, to a large extent, caused by the problem of the huge number of possible products that the product lines can be set to. This leads to the problem of not being possible to test every single product of the product line. As a result, most of the work in the product line engineering testing community focuses on the efficient derivation of products under test by employing Combinatorial Interaction Testing (CIT) techniques [LHFRE15]. To this end, a small subset of products are generated by employing several criteria, such as the pairwise coverage (i.e., the interaction of two features at least once) [PSK+10, POS+12, CDS08, HPP+14, HPHT15, JHF12, OZML11]. According to Lopez-Herrejon et al., domain testing is the field where search-based software engineering is most used [LHLE15].

Similarly, at the domain engineering level, several studies aimed to optimize the testing process by prioritizing the execution of products under test. Sanchez et al. compared several functional test prioritization criteria (such as the product size or product complexity) to increase the fault detection rate [SSRC14a]. The same authors extended their work to the Drupal case study by including non-functional prioritization criteria (e.g., number of changes in the assets) [SSPRC15]. Other works have focused on the prioritization of products employing search algorithms [PSS+16,

HPP+14], or by comparing the similarity of products [AHTM+14, AHTL+16]. Other prioritization strategies include using domain knowledge to extract desirable features and prioritize products including these ones [EBA+11]. Devroy et al. employed statistical testing to prioritize configurations based on their probability to be executed [DPC+14, DPC+15].

Although most of the product line engineering testing studies focus on the domain engineering level, some other studies have proposed optimization of product lines at the application engineering level. As for Search-Based Software Engineering (SBSE), Wang et al. proposed a test suite minimization approach for reducing test cases of Software Product Lines (SPLs) [WAG13, WAG15]. The same authors proposed a multi-objective approach for test case prioritization of SPLs, where three different weight-based search algorithms were compared [WBA+14]. Apart from search-based software engineering, other works have employed model-based techniques for the optimization of product line engineering testing at the application engineering level. Stricker et al. considered a model-based approach that employed data-flow dependencies to select test cases for customer-specific products with the objective of avoiding redundant test activities [SMP10]. Lachmann et al. proposed a delta-oriented approach that prioritizes test cases to cover new integrated feature assets [LLL+15]. Other works have combined feature models with other elements such as "component family models" to systematically select test cases [WAGL16, WGAL13].

## 3.2 Automation for testing Cyber-Physical Systems with Variability

Automation is important in any aspect of testing. For testing configurable CPSs automation gains importance due to several aspects, such as the high amount of configurations that need to be tested. This section provides a state of the art relevant to testing configurable CPSs from two aspects. The first aspect refers to test systems that are capable of performing the automated validation of configurable systems by considering variability (Section 3.2.1). The second aspect refers to automatic test case generation for CPSs (Section 3.2.2). Furthermore, in Section 3.2.3 we highlight recent approaches for testing CPSs, for instance, uncertainty-wise testing of CPSs.

### 3.2.1 Variability in Test Systems

Modeling variability in the elements of the test systems allow the execution of tests under different conditions. Furthermore, requirements of a configurable system varies from a configuration to another, and the test system has to be adapted in order to test

different system variants, thus, variability in the components of the test system can help to achieve this goal. Different works in the current state of the art have proposed variability handling test systems.

The approach presented in [KWW09] shows an evolutionary test system, primarily based on the MESSINA tool, that tests functional and non-functional properties of embedded systems. An evolutionary algorithm is an optimization technique based on the principles of the Darwinian theory of evolution, where a set of candidate solutions called individuals are selected. The fitness of these individuals are evaluated by the evolutionary algorithm by executing a problem-specific fitness function. The proposed approach by Kruse et al. in [KWW09] supports Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), Processor-in-the-Loop (PiL) or Hardware-in-the-Loop (HiL) test levels, and allows the reuse of test cases across them. In the case of MiL and SiL test levels, MESSINA supports different tools, e.g., MATLAB/Simulink, ASCET models, etc. In the case of HiL, MESSINA is connected to modularHiL, a universal HiL test system developed by Berner & Mattner. The main variability points of this approach can be found in managing different test levels for test execution.

Model-in-the-Loop for Embedded System Test (MiLEST) is a toolbox for MATLAB/ Simulink developed by Zander-Nowicka [ZN08]. This test system is designed towards the validation of automotive real-time embedded systems at the MiL test level. The hierarchy of MiLEST is divided into four abstraction levels: Test Harness level, Test Requirement level, Test Case level and Feature level. Although the approach in [ZN08] proposes mechanisms for modeling variants, the test system itself is not designed for the validation of variability-handling systems. The proposed modeling technique is uniform variability. This variability modeling technique allocates all the components in the modeling framework, i.e., Simulink, and the variability is bound with different mechanisms, e.g., switch and constants.

A product line of validation environments with variability to test different applications in different domains and technologies is proposed in [MGP08]. The study presents a validation environment able to test different System Under Tests (SUTs) from different domains, used programming languages, etc. The proposed validation system works as follows [MGP08]: The test engineer executes a test through the Graphical User Interface (GUI), the GUI sends the test command to the engine, and this transforms the test command into the programming language that the SUT understands. For this step, the engine communicates with the database to obtain the correspondences between the source and target languages. When the transformation is finished, the command is sent to the SUT through the SUT interface, and awaits the response to begin the process again. The variability points of this system includes the

user interface, test control, code generator, information system or gateway.

The study presented in [PPP09] defines an extended architecture for UML Testing Profile (UTP) to deal with variability in the test models. The proposed extension includes mechanisms to describe the behaviour of test cases and other elements needed to support variability. The main variability in the proposed UTP extension is included in the Test Context, Test Cases, Test Components and Data Pool, Data Partition and Data Selector:

- TestContext: It is a class that organizes the test artifacts and contains test cases [PPP09]. It can be stereotyped with "Variation Point", which means that the test cases corresponding to the TestContext have variation points [PPP09].
- TestCase: A test case is represented with UML sequence diagram in [PPP09]. A test case can also be stereotyped as "Variation Point" for testing a functionality with variability [PPP09].
- TestComponent: Test components interact with the SUT with the aim of realizing the test behaviour [PPP09]. In the proposed extension, a test component can be stereotyped with "Variation Point" or "Variant", which means that the test component can encapsulate the communication with the SUT, for the entire variation point or only for one of its variants.
- SUT: It can be stereotyped as "Variant", which means that it realizes the functionality for its variant.
- DataPool, DataPartition and DataSelector: The DataPool contains the test data while the DataPartition contains the equivalence classes and data sets [PPP09]. The dataPool can be stereotyped as "Variation Point", which means that it contains specific data for a Variation point. The DataPartition and the DataSelector are stereotyped as "Variant", which means that the DataPartition contains the data associated with one of its variants and the dataSelector selects the data in the DataPartition for a specific variant.

### 3.2.2 Test Case generation for Cyber-Physical Systems

The interest in testing CPSs has increased in the last few years by the research community. Specifically, most of the testing approaches for CPSs focus on test case generation. Several approaches proposed different test case generation approaches for CPSs based on Model-Based Testing (MBT). The idea of MBT is to develop a model of the system, which is later processed by a test generator, which generates test cases

based on the developed model to fulfill certain objectives (e.g., state coverage). Most of them are based on conformance testing. Mohaqeqi et al. proposed an approach for conformance testing for CPSs that captures conformance relations. The idea is to define some criteria on the test suite (relative to the sampling points and the system dynamics around those sampling points), which guaranteed soundness in test suites [MM16]. The same authors compared two conformance testing approaches (hioco approach and hconf) for CPSs [MMT14]. They concluded that hioco is richer and more expressive for modeling than hconf, although practical issues are detected when checking conformance. On the contrary, hconf is a more practical approach to checking conformance of CPSs, but it ignores important aspects in modeling (e.g., explicit discrete interaction and non-determinism). Abbas et al. also proposed an approach based on notions of conformance testing for CPSs, but from the control theory perspective [AHF+14, Abb15]. Aerts et al. reviewed the most common modeling notions for CPSs focusing on hybrid system models and provided a brief overview of conformance relations and conformance testing techniques for CPSs [ARM16]. Woehrle et al. proposed a methodology that uses measurements of physical quantities of CPSs for testing the conformance of a running CPS [WLT13]. To this end, the behavior of the system is described by means of a formal description, which allows for defects detection [WLT13].

Apart from conformance testing or MBT methods, other works rely on search-based techniques guided by a fitness function to generate test cases for CPSs. Their fitness functions are obtained by using simulation models of the Cyber-Physical System Under Tests (CPSUTs). Matinnejad et al. proposed a novel search-based algorithm for the automatic generation of test cases for Simulink models of CPSs [MNBB16]. Ben Abdesalem et al. focused on generating test cases for autonomous vehicles by employing multi-objective search algorithms [BANBS16]. Vos et al. proposed an evolutionary testing framework to test automotive systems, where objective functions were obtained using SiL and HiL simulations [VLW+13]. A drawback of these approaches is that the execution time of the algorithms is high because simulating the system in each iteration is too costly. Formal methods were also proposed for generating test cases for CPSs, where formal specifications using differential dynamic logic (i.e., a logic for the specification of hybrid systems) were used [ZHY13].

Some works focused on the generation of reactive test cases. Zander captured the reactivity of the system with test oracles and later employed a model-based approach to generate one test case per requirement [ZN08]. Mjeda captured the reactivity of the system with Simulink models and later used them to generate reactive test cases for safety-critical systems [Mje13]. Lehmann proposed a tool named Time Partition

Testing for the elaboration of model-based reactive test cases and it automatically generated executable test cases for different simulation tools (e.g., Simulink) [Leh00].

MATLAB/Simulink is one of the prevalent modeling and simulation tools for CPSs [MNBB16]. The generation of test cases for testing Simulink models has been widely applied. A wide range of studies have proposed mutation-based test case generation for testing Simulink models [BHM+10, BT+15, RSB+13, LTMHT14, ZC05, ZC08]. Others have focused on structural coverage [YRW+15, HWRS08]. Requirements coverage was employed for testing Simulink models from the aerospace industry [RWSH08]; in this case test cases were naively generated, and their empirical evaluation demonstrated that test cases generated for satisfying the requirements coverage outperformed test cases generated to satisfy the MC/DC coverage in terms of fault detection. Liu et al. generated test cases employing search-based techniques to improve fault localization of Simulink models [LLNB17].

### 3.2.3 Other Cyber-Physical Systems Testing Approaches

Apart from the generation of test cases, other approaches considered testing CPSs. Some works focus on testing CPSs under uncertain behaviors to deal with the unpredictability of the physical world, most of them inside the scope of the U-Test European project [AY15, AYZ16, ZSA+16, ALW+17, TB17].[1] The main goal of this project was to improve dependability of CPSs by testing them under uncertainty in a cost-effective manner. Apart from those works related with the U-Test project, other works have focused on testing CPSs. Spichkova et al. proposed an introduction to human-centered considerations for modeling and testing CPSs, which would allow for an agile iterative refinement process of different abstraction levels when errors are detected [SZF15]. Wan et al. proposed a general test platform for low-priced intelligent vehicles with wireless sensor networks navigation, which are one kind of CPSs, to test and verify properties of these systems [WSYL11]. Abbas et al. proposed a framework for automatic specification-guided testing of Stochastic CPSs, where they aim at detecting system operating conditions that cause the system to show the worst expected specification robustness [AHFU14]. To this end, they used Markov chain Monte Carlo algorithms.

## 3.3 Optimization

Test optimization plays a crucial role when testing configurable systems. The objective of test optimization is to cost-effectively test a system, i.e., reduce the cost of testing

[1] http://www.u-test.eu/

a system while the overall test quality is maintained. We tackle test optimization by selecting and prioritizing test cases; since these approaches are typically non-trivial, search-based algorithms are employed. This section highlights relevant work on test case selection and minimization in Section 3.3.1, and test case prioritization in Section 3.3.2.

### 3.3.1 Test Case Selection and Test Minimization

Test case selection and minimization has been widely studied for regression testing. In this context, while test case selection focuses on selecting a set of test cases from the test suite that tests a specific system version, test minimization aims to eliminate redundant test cases from the existing test suite in order to reduce cost (i.e., reduce the test execution time) [WAG15]. The main difference between both techniques is that while test minimization eliminates the redundant test cases permanently for the systems, test selection selects relevant test cases temporarily for testing the modified version of the system [YH12, Har11]. Nevertheless, from the perspective of test optimization, there is no significant difference between both techniques [Har11].

Engström et al identified 27 studies encompassing a total of 28 techniques for regression test selection [ERS10], including dataflow based [HRS+00], modification and changed based [FRC81, HS88, SR05], and coverage based [GHS92, HS88]. Furthermore, in the last few years, search-based approaches have gained important attention in the field of test case selection and minimization. Yoo and Harman used a Greedy algorithm and the Non-dominated Sorting Genetic Algorithm II (NSGA-II) to select test cases for testing programs [YH07]. They formulated the test case selection problem as a multi-objective problem that was instantiated with two version: (1) a bi-objective formulation that combined coverage and cost and (2) a formulation with three objectives, where, in addition to the coverage and cost, historical information related to faults was included. While their evaluation showed that NSGA-II performed best, their results indicated that the greedy approach produces good approximations to the pareto-front. Hemmati and Briand investigated different similarity measures (e.g., Hamming Distance) to select test cases in a MBT context [HB10]. They concluded that the Jaccard Index was the most cost-effective similarity measure when employing this technique. Pradhan et al. proposed a multi-objective test selection method that can be used when a limited time budget exists [PWAY16]. Their fitness function included one cost function (i.e., time difference) and thee effectiveness measures (i.e., mean priority, mean probability and mean consequence). Their empirical evaluation with eight multi-objective algorithms (including both, pareto-based and weight-based search algorithms) concluded that Strength Pareto Evolutionary Algorithm 2 (SPEA2)

was the best algorithm for solving the test selection problem in the presence of time budget. Lachmann et al. compared several black box test case selection criteria (e.g., fault revealing history, requirements coverage, etc.), and analyzed the effectiveness of the combination of several criteria [LFN+17].

### 3.3.2 Test Case Prioritization

A recent systematic literature review on test case prioritization [KIJT17] collected 69 studies on regression testing. They provided a taxonomy, where 14 test case prioritization approaches were identified. Among these approaches, search-based test case prioritization was the most common one. They also found the main advantages and limitations of each of the approaches.

Test case prioritization has been widely applied in the software engineering field [HMZ12, YH12]. The problem was formally defined by Rothermel et al., where six techniques (four of them coverage-based and two of them estimated ability to reveal faults) were compared [RUCH99, RUCH01]. Since then, several studies have been performed by the software engineering community in the test case prioritization context. According to a systematic mapping study performed by Catal and Mishra, most of the papers used only coverage-based prioritization methods [CM13]. In addition, many studies have used the Greedy algorithm for the test case prioritization problem; the use of metaheuristic and evolutionary algorithms for prioritizing test cases was first introduced by Li et al. in 2007 [LHH07]. Since then, many techniques have been empirically evaluated for the test case prioritization problem, such as, comparison between different evolutionary multi-objective algorithms [EYHB15], comparison between white-box and black-box test prioritization techniques [HPH+16], or comparison between static and dynamic test prioritization techniques [LMP16].

Although most of the studies on test case prioritization have proposed the use of coverage-based methods, in the last few years, importance has been given to test execution time. Malishevsky et al. combined test coverage information with test execution time for testing software systems [MRE02]. A genetic algorithm to prioritize regression test suites that will always run within a given time budget and will have the highest possible potential for defect detection based on coverage information was proposed by Walcott et al. [WSKR06]. A similar study, but employing Integer Lineal Programming techniques was proposed by Zhang et al. [ZHG+09]. Marijan et al. proposed the use of test execution time, along with historical failure data, to prioritize test cases for continuous regression testing [MGS13]. Srikanth et al. proposed a prioritization scheme based on the setup time of different configurations of a system [SCQ09]. Knauss et al. employed a combination of test case failures

and source code changes to prioritize system level test cases [KSM$^+$15]. Elbaum et al. used the concept of time windows to track the execution time of test suites combined with occurrence of failures to prioritize test cases in a continuous integration development environment [ERP14]. Hemmati et al. found that the use of previously failed test cases were a good source for prioritizing test cases [HFM15]. Notice that all these studies focused on testing software systems. On a much smaller scale, apart from purely software systems, test case prioritization has also been applied into other areas. For instance, Zhai et al. used test case prioritization for regression testing of location-based services (i.e., services with positional data) [ZJC14]. Despite its importance, few studies have tackled the problem of test case prioritization for CPSs. A recent study proposed a time-aware method using constraint programming to schedule test cases to run on multiple CPSs constrained by the tests' access to shared resources (e.g., measurement or networking devices) [MGS$^+$17].

Other approaches for test case prioritization include requirements-based (e.g., [SWO05, KM09]), model-based (e.g., [KTH05, KKT07, KKT08]), mutation score-based based (e.g., [RUCH99, RUCH01]) and machine learning-based (e.g., [BX16]). Furthermore, in the context of product line engineering, different test case prioritization approaches have been proposed (e.g., [SSRC14a, SSPRC15, PSS$^+$16, AHTL$^+$16]). Further information regarding the test case prioritization in this domain is detailed in Section 3.1.

## 3.4 Debugging and Fault Localization

Fault localization is the activity of identifying the locations of faults in a program [WGL$^+$16]. This activity is one of the most tedious and time-consuming when debugging a program, but still it is critical [WGL$^+$16]. Wong et al. identified 331 published papers since 1970 to 2014 in fault localization and 54 master and Ph.D. theses. On the one hand, traditional fault localization techniques include intuitive fault localization techniques, including program logging, assertions, breakpoints, and profiling. On the other hand, to overcome the size and scale of nowadays software, advanced fault localization techniques have been proposed, including slice-based (Section 3.4.1), spectrum-based (Section 3.4.2), program state-based (Section 3.4.3), machine learning-based (Section 3.4.4), data mining-based (Section 3.4.5) and model-based (Section 3.4.6) [WGL$^+$16].

### 3.4.1 Slice-based fault localization

Slice-based fault localization was proposed by Weiser in 1979 [Wei79], and since then, many studies on this topic have been published [WGL+16]. Program slicing consists of abstracting a program into a reduced form by deleting irrelevant parts such that the resulting slice will be equivalent to the original program with respect to certain specifications [WGL+16]. This permits reducing the search domain while developers localize bugs in their code [Wei84], which is based on the idea that when a test case fails due to an incorrect variable value at a statement, the defect shall be found in the static slice associated with that variable-statement pair [WGL+16]. This allows debuggers to confine their search to the slice instead of looking at the entire program [WGL+16]. An issue with slice-based fault localization techniques is that handling pointer variables can make data-flow analysis inefficient [WGL+16].

### 3.4.2 Spectrum-based fault localization

Spectrum-Based Fault Localization (SBFL) is a technique to assist on the location of program bugs [AZGvG09, LLT15]. According to Wong et al., SBFL is the most widely investigated technique for locating faults in software [WGL+16], encompassing the 35% of papers. SBFL uses the results of test cases and their corresponding code coverage information to estimate the risk of each program component (e.g., statements) of being faulty. A *program spectrum* refers to a collection of data that provides a specific view on the dynamic behavior of a software program such as statement or branch coverage [AZGvG09, RBDL97]. Various forms of program spectra have been proposed [HRS+00]. For example, *block-hit* is a commonly used program spectra, where the program code is divided into statement blocks [LLT15]. When SBFL with block-hit spectra is used, the result of the technique is an ordered list of code blocks sorted by their likelihood to cause the failure, so-called *suspiciousness score*.

Table 3.1 illustrates an example of SBFL with block hit spectra in a C program. Horizontally, the table shows the five code blocks in which the program has been divided, i.e., the components. Note that the code has a bug in block $b_3$. Vertically, the table shows four test cases of the program. For each test case (i.e., $T_1$, $T_2$, $T_3$ and $T_4$), a cell is marked with "•" if the program block of the row has been exercised by the test case of the column, creating what is known as the *coverage matrix* [AZVG07]. Additionally, the final row depicts the so-called *error vector*, which contains the outcome of each test case, either successful ("S") or failed ("F"). Based on this information, the suspiciousness score of each block can be calculated using more than 30 different techniques proposed in the literature [XCKX13]. One of the most

well-known techniques to calculate the suspiciousness score is named *Tarantula*, which, for a program component (in our example a statement block), is computed as follows [LLT15].

Table 3.1: An example showing the suspiciousness value computed using the Tarantula technique

| ID | Program block | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $N_{CF}$ | $N_{CS}$ | $N_S$ | $N_F$ | Suspiciousness | Ranking |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $b_1$ | int count n;<br>Ele *proc;<br>List *src_queue, *dest_queue;<br>if (prio >= MAXPRIO) { /*MAXPRIO=3*/ | • | • | • | • | 1 | 3 | 3 | 1 | 0.5 | 2 |
| $b_2$ | return;<br>} | • | | | | 0 | 1 | 3 | 1 | 0 | 3 |
| $b_3$ | src_queue = prio_queue[prio];<br>dest_queue = prio_queue[prio + 1];<br>count = src_queue->mem_count;<br>if (count > 1) {<br>/* **BUG: It should be if (count >= 1)** */ | | • | • | • | 1 | 2 | 3 | 1 | 0.6 | 1 |
| $b_4$ | n= (int) (count*ratio + 1);<br>proc = find_nth(src_queue,n);<br>if (proc) { | | • | • | | 0 | 2 | 3 | 1 | 0 | 3 |
| $b_5$ | src_queue = del_ele(src_queue,proc);<br>proc->priority = prio;<br>dest_queue = append_ele(dest_queue,proc);<br>}<br>} | | • | • | | 0 | 2 | 3 | 1 | 0 | 3 |
| Execution results | | S | S | S | F | | | | | | |

$$Suspiciousness(Tarantula) = \frac{\frac{N_{CF}}{N_F}}{\frac{N_{CF}}{N_F} + \frac{N_{CS}}{N_S}} \tag{3.1}$$

where $N_{CF}$ is the number of failing test cases that cover the block, $N_F$ is the total number of failing test cases, $N_{CS}$ is the number of successful test cases that cover the block, and $N_S$ is the total number of successful test cases. The suspiciousness score of each block is in the range [0,1], i.e., the higher the suspiciousness score of the block, the higher the probability of having a fault. The values of $N_{CF}$, $N_{CS}$, $N_S$, $N_F$ and the Tarantula suspiciousness value of each code block are given in Table 3.1. The last column indicates the position of the statement in the suspiciousness-based ranking where top-ranked blocks are more likely to be faulty. In the example, the faulty block ($b_3$) is ranked first.

Suspiciousness techniques may often provide the same value for different components, being these tied for the same position in the ranking, e.g., blocks $b_2$, $b_4$, and $b_5$ in Table 3.1. Under this scenario, different approaches are applicable such as measuring the effectiveness in the best and worst scenarios, using an additional technique to break the tie, or using some simple heuristics such as alphabetical ordering [WGL$^+$16].

Among the works on SBFL, several empirical studies have been carried out to assess the performance between different SBFL techniques. Pearson et al. compared the performance of five SBFL techniques and two mutation-based fault localization techniques for both artificial and real faults from five open source projects (JFreeChart, Google Closure compiler, Apache Commons Lang, Apache Commons Math and Joda-Time) [PCJ+17]. They found that Dstar outperformed the rest of techniques. They also found that although in artificial faults Tarantula does not perform better than other techniques (e.g., Ochiai), in real faults there is no statistically significant difference between Tarantula and the rest of techniques. Abreu et al. compared Ochiai with Tarantula in the Siemens set, finding that Ochiai performed better [AZVG07]. Ochiai is also found to be the best technique in the study performed by Le et al., where the Siemens set, together with NanoXML, XML-security and Space were employed as program subjects [LTL13]. Wong et al. compared 38 techniques on different real-world programs (e.g., Siemens set, grep, make, gzip, etc.), finding that their proposed Dstar technique outperformed the rest [WDGL14]. Jones and Harrold compared five SBFL and slice-based technique in the Siemens set, finding that Tarantula was the best technique at finding faults [JH05]. The use of SBFL assumes the use of a test oracle, since SBFL needs test results. However, as a test oracle is not always available, Xie et al. adapted SBFL to the metamorphic testing context by proposing an approach named metamorphic slice [XWCX13]. They compared their approach with three SBFL techniques (Tarantula, Ochiai and Jaccard) in nine programs and found that their approach is as effective as traditional SBFL.

### 3.4.3 Program State-Based Fault Localization

The idea behind program state-based fault localization is to monitor the variables of the program at a particular point during program execution. This monitoring can later be used to locate program bugs. A popular program state-based fault location technique is delta debugging [ZH02]. The core idea of delta debuging consists of simplifying large test cases that produce a fault by removing irrelevant details [ZH02]. The Delta Debugging algorithm has been extended in other studies, proposing a Hierarchical Delta Debugging approach where unlike in Delta Debugging, the input structure is taken into account [MS06]; this enables reducing the number of test cases and producing smaller outputs. Similar techniques to the Delta Debugging have been proposed for minimizing the constraints on the input parameters to isolate the cause of faults of web applications [AKD+10], or for isolating C compiler bugs [RCC+12].

### 3.4.4 Machine Learning-Based Fault Localization

Machine learning-based fault localization techniques aim at localizing faults by trying to learn the location of a fault processing input data (e.g., statement coverage and test execution results of each test case) [WGL+16]. Different machine learning algorithms have been investigated for locating bugs in software, such as back-propagation neural networks [GLJZ12, HN+88, AAPV09], radial basis function networks [WDG+12] or decision tree algorithms [BLL07].

### 3.4.5 Data Mining-Based Fault Localization

Data mining technique are proposed to overcome the problem related to the huge volume of data that make other fault localization techniques heavy for usage in practice [WGL+16]. As in the case of machine learning-based fault localization techniques, data mining techniques also produce a model using information extracted from data. This permits data mining techniques to uncover hidden patterns in samples of data that, due to the large volume of information, cannot be discovered by manual analysis [WGL+16]. To this end, several approaches have considered the use of data mining for fault localization in software [NAW+08, CDFR08, CDFR11, ZZ14].

### 3.4.6 Model-Based Fault Localization

The idea of model-based fault localization consists of automatically generating a model from the source code being debugged, which is later used to identify model elements when existing deviations between the observed program execution and the expected results [WGL+16]. Different model-based fault localization techniques exist, including dependency-based [MSW00, MS02, WSM02], abstraction-based [MS07], value based [KW04, MSWW02] and model checking-based [BNR03, CGS04, GSB10, GCKS06, KB11].

## 3.5 Critical analysis of the State of the Art

This section critically analyses the current state of the art in configurable CPSs testing, which aims at providing potential research opportunities.

In the scope of product line engineering testing, most of the approaches focus on the domain engineering layer, either generating relevant products (e.g., [PSK+10, POS+12, HPP+14, HLL+16]) or prioritizing the order in which the generated products have to be tested (e.g., [SSRC14a, SSPRC15, DPC+14, DPC+15, AHTM+14, AHTL+16]). Despite some works focusing on the application engineering level (e.g.,

[SMP10, WAGL16, WBA+14, WAG13]), none of them consider configurable CPSs, which compared to common SPLs face several differences for testing. The most striking differences are (1) the need for employing simulation models to test them, (2) higher test execution time due to simulation of the physical layer and longer test cases, (3) different types of faults (e.g., apart from software faults, faults in sensors, actuators or communication systems) and (4) the use of different test levels (i.e., MiL, SiL and HiL).

In the scope of automated testing, test systems provide a great possibility to perform the automated validation in a systematic manner. In this field, to the best of our knowledge, there are no test system approaches that consider CPSs particularities. As for variability of test systems, to the best of our knowledge, there are no approaches that systematically generate specific test systems instances for the automated validation of CPSs configurations in Simulink models, which is the de-facto CPSs simulation tool [MNBB16].

Regarding automated test case generation, many test case generation approaches for CPSs rely either on MBT (e.g., [MM16, MMT14, ARM16, AHF+14, Abb15, WLT13]) or formal methods (e.g., [ZHY13]). However, Briand et al. claimed problems when generating test cases for CPSs using MBT or formal approaches for generating CPSs test cases [BNSB16]. On the one hand, in the CPSs context, MBT faces scalability as well as practicality issues [BNSB16]. Furthermore, when test engineers have to consider properties of systems involving physical devices with continuous dynamics and its environment (e.g., people), the scalability challenge is further exacerbated [BNSB16]. On the other hand, formal methods involve complex mathematics and are rare in practice [MNBB16].

To overcome this problem, several approaches considered search-based software engineering for testing CPSs, where they proposed obtaining fitness values by simulating the CPSUT. While this is an interesting approach, since worst case scenarios can be pinpointed, it is an issue as well, basically due to the high amount of time required to generate effective test cases (e.g., Ben Abdessalem et al. employed a time budget of 120 minutes for test case generation [BANBS16]).

As for reactive test cases generation, which provide a good possibility for testing CPSs because they can lead with system's unpredictability, the approaches considering the generation of reactive test cases (e.g., [Leh00, ZN08, Mje13]) focused on testing requirements, without considering other critical measures when testing configurable CPSs, such as the test execution time. Moreover, in all of them the process of generating test cases is semi-automatic as they all need to specify some reactivity behaviors.

Several approaches have proposed test case selection and prioritization approaches, especially search-based, either for general purpose software (e.g., [PWAY16, LHH07]) or SPLs [WAG13, WBA+14]. However, most of them are for software systems, which have several differences when compared to CPSs. For instance, the test cases for testing software systems are typically in the order of some milliseconds [AIB10], whereas the test cases for testing CPSs are in the order of some seconds or minutes. Another difference is that systematic simulation-based testing of CPSs involve different "-in-the-Loop" test levels, and thus, when selecting and prioritizing test cases, the objectives of each test level have to be considered and integrated in the fitness functions. An idiosyncrasy of reactive test cases is that their test execution time varies depending on the previously prioritized test case.

Debugging is an important aspect when testing any kind of system. Several approaches have considered fault localization, either static (e.g., [BTWV15]) or dynamic (e.g., SBFLs [AZVG07, AZGvG09]). However, most of them focus on locating bugs in general purpose software, but in the context of configurable systems (e.g., SPLs or configurable CPSs), the debugging aspect has centered little attention. Debugging is important either for SPLs or configurable CPSs, since localizing the faulty features would reduce the time for test and validation. Furthermore, as compared to general purpose software, configurable systems typically use feature models for managing variability. Applying a fault localization technique at the feature level could help reduce the time for fault localization not only in software but also in other sources (e.g., sensors or actuators, and interaction between features).

Chapter 4

# Theoretical Framework

In this chapter we give a theoretical overview of the dissertation. Specifically, we define five research objectives (Section 4.1) together with the hypotheses (Section 4.2). Furthermore, we give an overall overview of the theoretical framework proposed for testing configurable Cyber-Physical Systems (CPSs) (Section 4.3). In addition, we explain the employed case studies that were used to validate the effectiveness of the solutions proposed in the theoretical framework (Section 4.4). Lastly, we explain which case studies where employed for the validation of each of the contributions in Section 4.5.

## 4.1 Research Objectives

The goal of this thesis is to **develop and evaluate a set of tools and methods that permit the systematic testing of *configurable CPSs*.** This objective can be divided into the following sub-objectives:

- *Objective 1:* Develop a tool-supported methodology which permits the automatic generation of test system instances for specific configurations of *configurable CPSs*.
- *Objective 2:* Develop and evaluate a tool for the automatic generation of test cases that permits testing *configurable CPSs* on a cost-effective manner.
- *Objective 3:* Develop and evaluate different test case selection algorithms which are capable of cost-effectively testing specific system variants of *configurable CPSs* at different test levels (i.e., Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL)).
- *Objective 4:* Develop and evaluate different test case prioritization algorithms that will help reduce the testing costs of *configurable CPSs* at different test levels (i.e., MiL, SiL and HiL).

- *Objective 5:* Develop and evaluate a methodology that helps debuggers to localize and isolate faults in *configurable systems*.

## 4.2 Research Hypotheses

- *Hypothesis 1:* The use of variability models (e.g., *feature models*) helps the systematic generation of test system instances for testing *configurable CPSs*, which is faster than a manual generation. This hypothesis corresponds to research objective 1.
- *Hypothesis 2:* The use of multi-objective search algorithms permits the generation of cost-effective test suites in CPSs testing. This hypothesis corresponds to research objective 2.
- *Hypothesis 3:* The use of search algorithms permits optimization of the test process of *configurable CPSs* employing simulation by selecting and prioritizing test cases. This hypothesis corresponds to research objectives 3 and 4.
- *Hypothesis 4:* The adaption of Spectrum-Based Fault Localization (SBFL) techniques to the product line engineering context permits the localization and isolation of faulty features. This hypothesis corresponds to research objective 5.

## 4.3 Overview of the Theoretical Framework

Figure 4.1 depicts the overall overview of the developed methods for testing configurable CPSs and the dependencies between them. The scope of this dissertation is to advance the practice of testing configurable CPSs. To this end, we proposed and developed a set of methods that were individually evaluated. Despite these methods being individually evaluated, they can be used either collectively or individually. Furthermore, since configurable CPSs share similarities with product lines, the proposed methods have been integrated with product line engineering methods (e.g., variability modeling) and tools (e.g., FeatureIDE [TKB+14]). To this end, the different steps proposed in this dissertation to cost-effectively test configurable CPSs include the following:

- ***Step 1: Variability modeling:*** The first step consists of the elaboration of a variability model (e.g., feature model). In our case, the tool FeatureIDE [TKB+14] is employed for this activity. It is important to highlight that not only the variability of the system is modeled, but also the variability of the test system (e.g., requirements, test stimuli, etc.).

- ***Step 2: Product suite generation:*** The second step consists of the generation of the product suite. The variability model developed in the previous step is later processed by a product suite generation algorithm. This product suite generation algorithm, which is typically a Combinatorial Interaction Testing (CIT) algorithm, will generate a set of relevant products following certain criterion. A typical criterion in configurable systems is the pairwise coverage [PSK+10, POS+12, HPP+14].

- ***Step 3: Product suite prioritization:*** With the sake of improving the fault detection rate, the third step aims at prioritizing the order in which the generated products have to be tested. Different product prioritization approaches have been proposed in the literature, which have as the main objective to increase the fault detection rate of the configurable systems (e.g., [SSRC14a, SSPRC15, PSS+16, AHTL+16, DPC+15]).

- ***Step 4: Test case generation:*** In parallel to product suite generation and product suite prioritization, test cases that have to be executed can be generated. This is depicted in Figure 4.1 as the fourth step. To this end, this dissertation proposes a test case generation approach based on multi-objective search algorithms, which is further explained in Chapter 6.

- ***Step 5: Test system generation:*** Once the product suite and the test cases for testing them have been generated, the test execution can be started. In order this test execution to be automatic, a test system needs to be generated. The fifth step consists of generating a test system in Simulink that permits automatically testing a product of the configurable CPS. This test system includes the required test cases as well as test oracles. A fully automated approach is proposed in Chapter 5.

- ***Step 6: Test case selection:*** When testing a specific product, there might be redundant test cases, test cases without any fault detection ability or test cases that do not test requirements specific to the selected product, and thus, they might be omitted. As a sixth step, this dissertation proposes a test case selection method to cost-effectively test products of configurable CPSs. This is further explained in Chapter 7.

- ***Step 7: Test case prioritization:*** To further optimize the testing process of configurable CPSs, a test case prioritization method is proposed in Chapter 8. Given a specific product, our test case prioritization approach aims to reduce the overall simulation time, the fault detection time and the requirements testing time.

- ***Step 8: Test execution:*** Test execution is the activity of automatically executing and evaluating test cases for a specific product. In our case, this is performed in an

eighth step employing a Simulink model. Notice that the test execution executes the selected test cases in the sixth step, in the order given by the prioritization approach in the seventh step and employing the test system generated in the fifth step. When the test execution has been finished, the whole process is repeated from the fifth step, until either the testing time budget has been finished or until all the generated products in the second step have been tested.

- ***Step 9: Debugging:*** When the testing has been finished, the debugging phase starts. The objective of this phase is to localize and isolate faults of the configurable CPS. When employing the methods proposed in this dissertation, the debugging process would be conducted in the ninth step. This debugging process is explained in detail in Chapter 9.

We reiterate that the proposed methods can be used either collectively or individually. For instance, in some cases test engineers could manually generate test cases and thus, it would not be required to employ our method. Another example could be the test system. For instance, in our work with an industrial partner we proposed a different test system to that proposed in Chapter 5 [SEA+17].

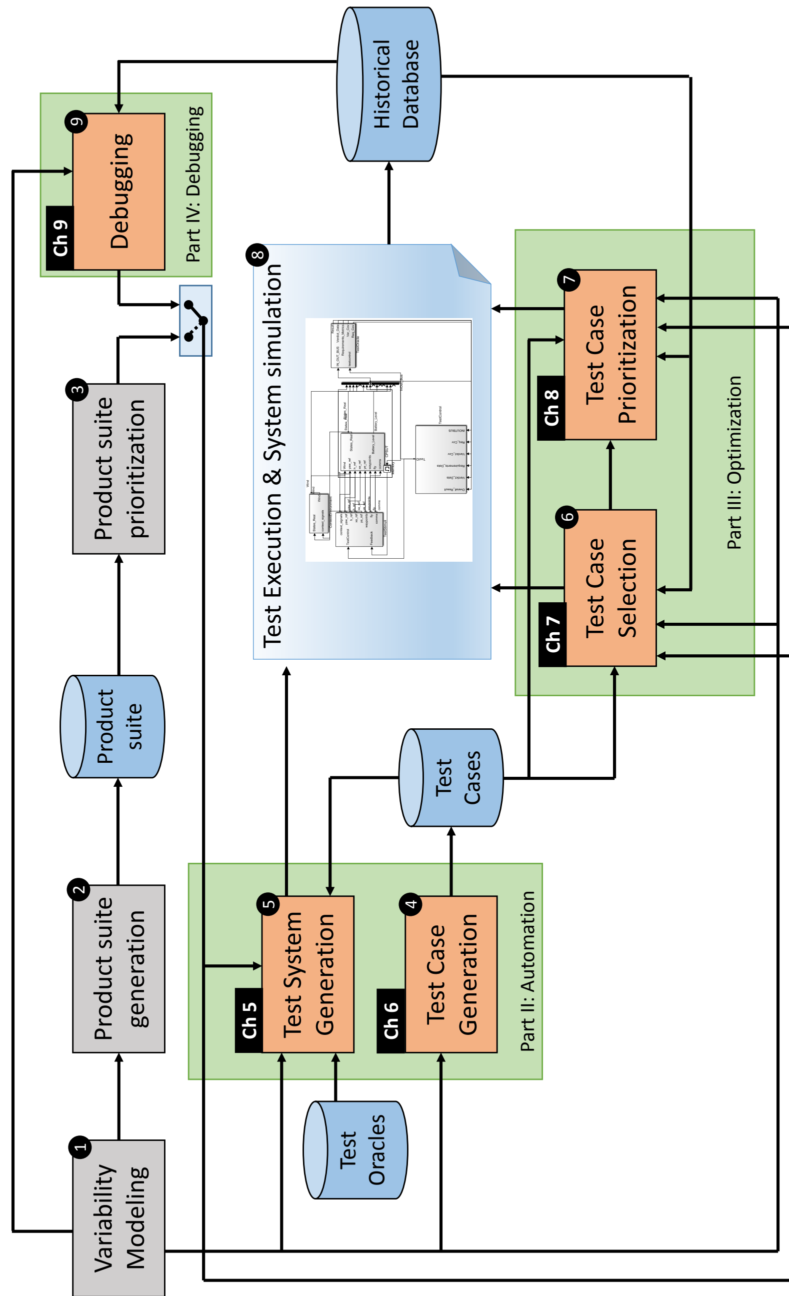



Figure 4.1: Overview of each contribution of the dissertation, their dependencies and structure of the thesis

## 4.4 Case Studies

With the objective of validating the proposed methods, four case studies were developed in MATLAB/Simulink. These case studies were from different domains (i.e., aerospace, automotive, industrial and electrical) and complexities. Some of them were developed from the beginning, whereas others were adapted from either open source models or industrial case studies. Apart from these case studies, for the experiments related to the last contribution (Chapter 9), examples from the SPLOT repository were employed [MBC09b].

### 4.4.1 Unmanned Aerial Vehicle

The AR.Drone is an Unmanned Aerial Vehicle (UAV) developed by Parrot for the market of video games and home entertainment [BCVP11]. Mosterman et al. modeled the dynamics of the AR.Drone using experimental input-output data, the structure of the vehicle, equations of motion and system identification techniques [MSB+14]. The mathematical models of the AR.Drone dynamics were previously validated with the real vehicle in [MSB+14], where an accurate position estimation was demonstrated.

We reused the AR.Drone model developed by [MSB+14] and added different variability points in terms of extra functionalities as well as variability points in hardware and software units. In addition, as we added new functionalities, functional requirements were re-written.

#### System Architecture

The cyber layer of the UAV is composed of three different platforms, as depicted in Figure 4.2. Each platform controls the UAV at a specific level. Platform 1 corresponds to the layer performing the high level control of the system. This platform is in charge of deciding the path that the UAV must follow. For this task, the platform integrates three sensors (obstacle sensor, battery sensor and GPS). It is also the platform in charge of communicating to the ground station. Based on the user's request, the embedded system obtains positional data via the GPS as well as information from the outside environment (e.g., if there is any obstacle) and the path to follow is plotted. It also integrates the "FlyLed" that is turned on while the UAV is flying. Platform 2 makes a low level control to ensure that the UAV keeps flying. It obtains data corresponding to the dynamics of the system with the gyroscope sensor and computes a set of control algorithms to regulate the speed of each rotor so that the system keeps stable. The speed of each rotor is stabilized by sending a set of speed commands to Platform 3,

which is in charge of controlling the speed of each rotor. This platform performs the lowest level control.

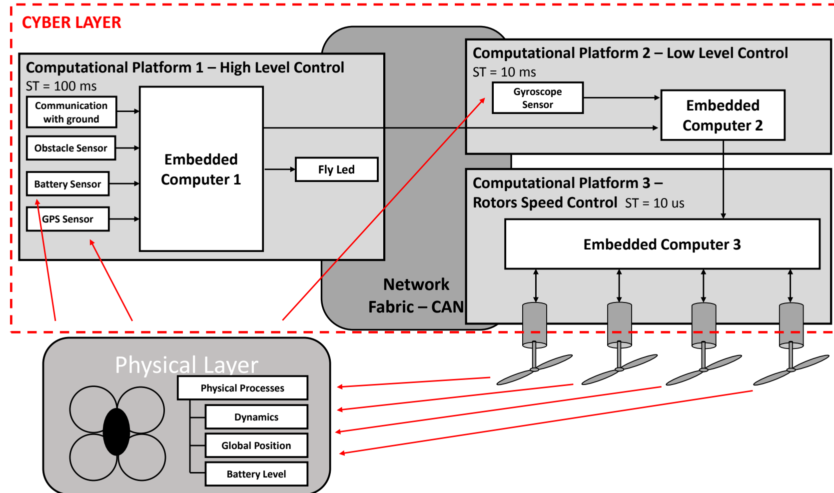



Figure 4.2: Overview of the structure of a CPS, corresponding to an example of an UAV (based on [LS15]). ST refers to the sample time of the computations to obtain data from sensors and compute their control tasks in each level

### Variability of the AR.Drone UAV

The identified variability has been classified in the two main layers that a CPS can have, i.e., the cyber and physical layers. Figures 4.3 and 4.4 depicts the feature models used to manage the variability of the AR.Drone model.[1] As for the software, there are two mandatory features: (1) track system and (2) control. The former refers to how the UAV plans its path. As compared to the model developed by [MSB+14], we included an extra path planning strategy, where the UAV can follow a person from different perspectives, i.e., from the left, right, front and back (as specified by the user). The latter refers to the control strategy. The control strategy proposed in [MSB+14] consisted of a set of proportional controllers for the position, height, forward and lateral velocity and heading angle. In our case, we give the customer the option to choose between a proportional or a proportional-integral control strategy.

In addition, we included some optional features in the software. We added some safety functions: collision avoidance equipment, wind avoidance algorithms, an emergency system and a strategy to independently return home. We also added some functionalities regarding the battery management. We developed two battery models:

[1]For presentation purposes, two independent feature models have been developed.

a battery lasting around 12 minutes, and another one lasting around 25 minutes. In addition, to prevent the drone from losing battery power, the vehicle could either launch the landing program, or it could find the closest battery station to recharge the battery. As for the Real-Time Operating System (RTOS), in our modified system, we gave the option of using RTOS or not. If the customer chooses to use a RTOS, embedded linux or FreeRTOS [Fre14] can be used. By default, the AR.Drone uses an embedded linux operating system [BCVP11].

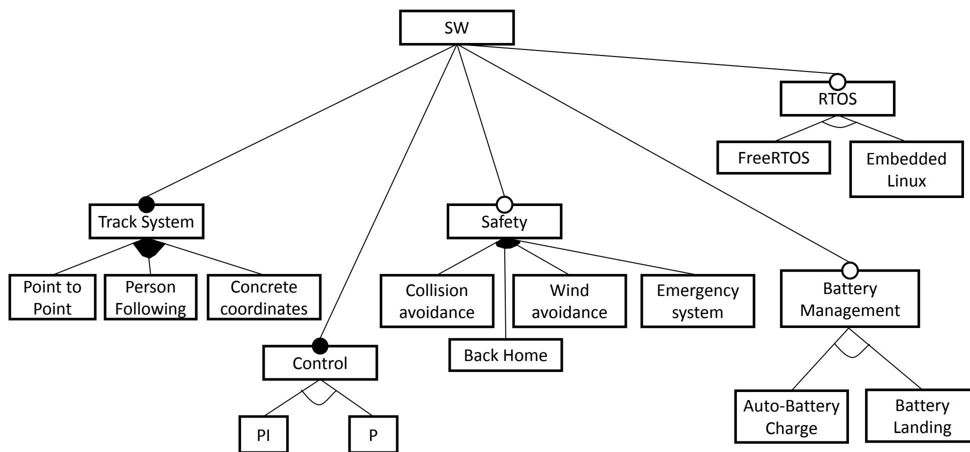



Figure 4.3: Feature model for the configurable AR.Drone software

From the hardware perspective, we also proposed some variability points, allowing the user to choose different components. Concerning the sensors, four types of sensors could be used in the drone: Gyroscope and GPS, which are mandatory for position and vehicle state estimations, Obstacle Sensor, which is only employed if the collision avoidance strategy is chosen, and battery sensor, which is mandatory if the user has chosen the automatic management of the battery. For each type of sensor, the user can choose among three different models. With regard to actuators, we proposed other rotors for higher speeds; thus, the user can choose between low or high speed rotors. In addition, a LED that indicates that the vehicle is flying was added as an optional feature.

### System Requirements

A total of 20 requirements were defined. There are certain requirements that are mandatory (Table 4.1) whereas others are optional requirements (Table 4.2). Mandatory requirements are those related to mandatory features. For instance, $r_4$ is related to the time that the UAV must fly, which is directly covered by the battery. In the

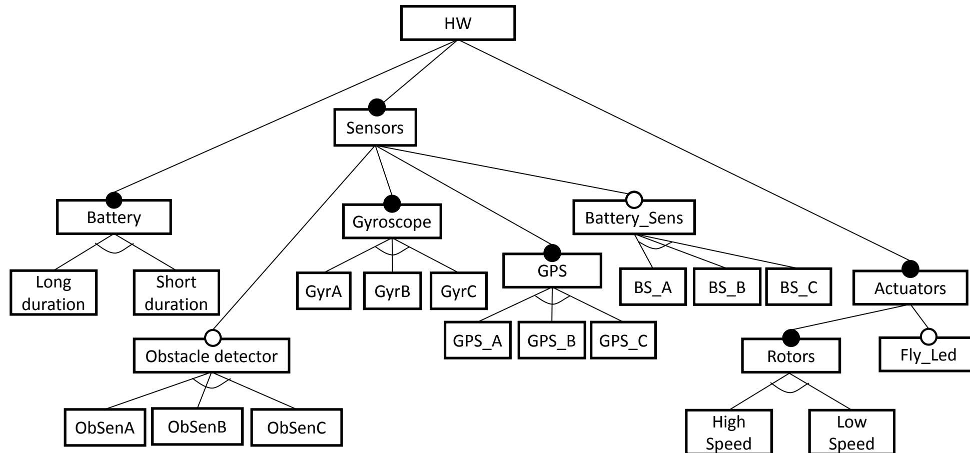



Figure 4.4: Feature model for the configurable AR.Drone hardware

same way, optional requirements are those related to optional features. As an example, $r_6$ is related to the wind avoidance, which is an optional functionality; as a result, if the chosen configuration does not include this functionality, $r_6$ will not be inside the requirements of the chosen configuration.

Table 4.1: Mandatory functional requirements for the CPS involving the configurable UAV

| Reqs | Sub-reqs | Features Involved | Requirement |
|---|---|---|---|
| $r_1$ | $r_{1a}$ | Rotor A | The maximum vertical speed for the drone shall not exceed 1 m/s |
| | $r_{1b}$ | Rotor B | The maximum vertical speed for the drone shall not exceed 3 m/s |
| $r_2$ | $r_{2a}$ | Rotor A | The maximum horizontal speed for the drone shall not exceed 3 m/s |
| | $r_{2b}$ | Rotor B | The maximum horizontal speed for the drone shall not exceed 5 m/s |
| $r_4$ | $r_{4a}$ | Battery A, Battery Control | The flight program shall last at least 12 minutes |
| | $r_{4b}$ | Battery B, Battery Control | The flight program shall last at least 25 minutes |
| $r_{10}$ | $r_{10}$ | Gyroscope, Controllers | When the speed is around 0 m/s, the pitch and roll euler angles of the drone shall not exceed 20 degrees (deg)/100 meters height |
| $r_{11}$ | $r_{11}$ | Gyroscope, Controllers | The yaw angle shall be the one selected by the drone driver with a maximum error of 15 degrees (deg)/100 meters high |
| $r_{16}$ | $r_{16}$ | Communication | The drone shall not fly if the command for flying is not activated |
| $r_{17}$ | $r_{17}$ | gps | The drone shall not move horizontaly if it is placed in the floor, i.e., h ==0 |
| $r_{18}$ | $r_{18}$ | gps | The drone shall not move rotationally if it is placed in the floor, i.e., h ==0 |
| $r_{19}$ | $r_{19}$ | Communication | The drone shall land if the communication with the base station has been lost |

In addition, some requirements handled variability (i.e., $r_1$, $r_2$, $r_3$, $r_4$ and $r_7$). For instance, $r_1$ refers to the maximum vertical speed of the UAV. However, two kinds of rotors can be chosen by the user: high speed rotors and low-speed rotors. Thus, the maximum allowed vertical speed will not be the same for high or low-speed rotors. Therefore, $r_1 = \{r_{1a}, r_{1b}\}$, where $r_{1a}$ specifies the maximum vertical speed for system variants using low-speed rotors, whereas $r_{1b}$ specifies the maximum vertical speed when high speed rotors are chosen for the configuration.

Table 4.2: Optional functional requirements for the CPS involving the configurable UAV

| Reqs | Sub-reqs | Features Involved | Requirement |
|---|---|---|---|
| $r_3$ | $r_{3a}$ | Battery A, Battery Control | The drone shall begin the landing if the battery is less than 20% |
| | $r_{3b}$ | Battery B, Battery Control | The drone shall begin the landing if the battery is less than 15% |
| $r_5$ | $r_5$ | Emergency system | The drone shall run the landing program if there is an emergency |
| $r_6$ | $r_6$ | Wind avoidance | The drone shall run the landing program if there is too much wind |
| $r_7$ | $r_{7a}$ | BatteryA, Battery Control, autobattery, GPS | The drone shall displace to the closest battery station if there is less than 30% |
| | $r_{7b}$ | Battery B, Battery Control, autobattery, GPS | The drone shall displace to the closest battery station if there is less than 25% |
| $r_8$ | $r_8$ | AutoHome, GPS | The drone shall come back home if asked |
| $r_9$ | $r_9$ | Collision Avoidance | If there is an obstacle, the drone shall detect it in less than 0.5 seconds and be able to skip it independently |
| $r_{12}$ | $r_{12}$ | Point to point, GPS | The drone shall follow all the points with a maximum error of 40 cms |
| $r_{13}$ | $r_{13}$ | ConcreteCoordinates, GPS | The drone shall achieve the selected coordinates with a maximum error of 40 cms |
| $r_{14}$ | $r_{14}$ | PersonFollowing, GPS | The drone shall follow a person to 3 meters of distance from the horizontal coordinates, with a height indicated by the user |
| $r_{15}$ | $r_{15}$ | PersonFollowing | The drone shall land if the contact with the person to follow has been lost due to the distance |
| $r_{20}$ | $r_{20}$ | Flying Led | The drone shall turn on the led while it is on the air |

### 4.4.2 Adaptive Cruise Control

The second case study involves an Adaptive Cruise Control (ACC), which is an adaption of the Cruise Control model developed by Daimler AG. The Cruise Control example only involved the speed control units of the system, together with the physical vehicle model. In this case, we adapted this model and included the capabilities for the adaptiveness (i.e., autonomously be able to adapt speed based on the context information such as speed limit signals and position and speed of rest of vehicles). Figure 4.5 depicts the overall system architecture. While the physical layer as well as

computational platforms 2 and 3 were original from the Daimler's model, platform 1 was developed during this Ph.D. project.

### System Architecture

The system architecture (Figure 4.5) of the ACC case study is composed of three main computational platforms connected by a CAN network (which is the standard automotive communication protocol [KFK14]) in addition to the physical vehicle model. The first computational platform controls the adaptive functionalities, which involves a radar sensor to capture the vehicles that are close to the car. When a car gets closer, an acoustic alarm can get turned on to warn the driver. The camera unit has as an objective to captures the traffic signals to adapt the speed of the car to the speed limits of the road. Finally, the emergency break unit captures information of the environment to avoid a collision of the vehicle either with other vehicles or pedestrians. The three units are also composed by their corresponding embedded computer to perform their corresponding signal processing as well as higher level computations.

The second and third platforms were developed by Daimler AG, and it involves the speed control of the car. While platform 2 performs the control of the car speed, i.e., the high level control, platform 3 performs a lower level control involving the speed of the car engine. The physical layer of the case study involves the vehicle dynamics, which is a 1400 kg car, and the model includes, among other subsystems, the engine of the car as well as the different gears.

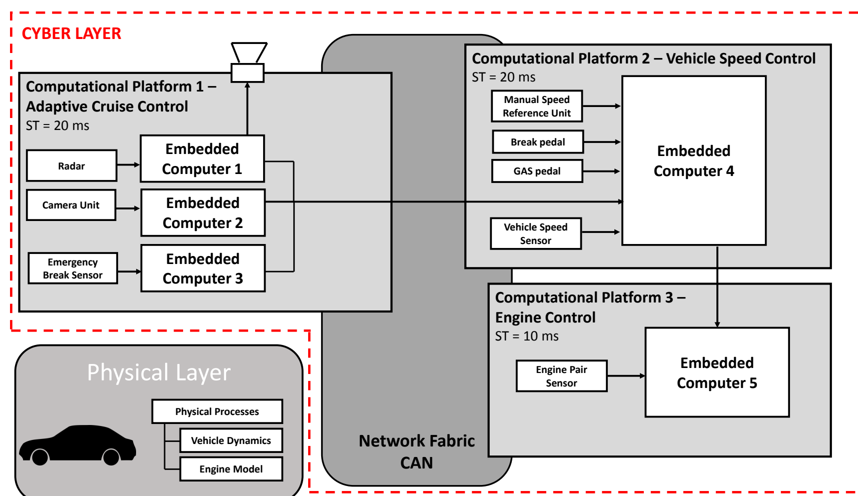



Figure 4.5: Overview of the structure of the case study corresponding to the ACC

### Variability of the ACC system

Figure 4.6 shows the feature model capturing the variability of the developed ACC case study. All these variability points were modeled in a Simulink model, and by means of a parser, we were able to automatically configure the model for a specific system variant. The mandatory features are those that are necessary to perform the cruise control of the vehicle (e.g., `Speed Sensor`). The different optional functionalities include the `adaptive cruise control`, the `automated speed limit` and the `emergency break`. An additional actuator is included, involving the `acoustic alarm`, which warns to the driver when it is too close to a vehicle. The sensors `radar` and `camera` are constrained to the adaptive and the sign detection functionalities. Apart from the variability shown in the feature model in Figure 4.6, the case study includes several parameters that can be used for calibration. These parameters include for instance, parameter of the speed controllers, which are PID controllers.

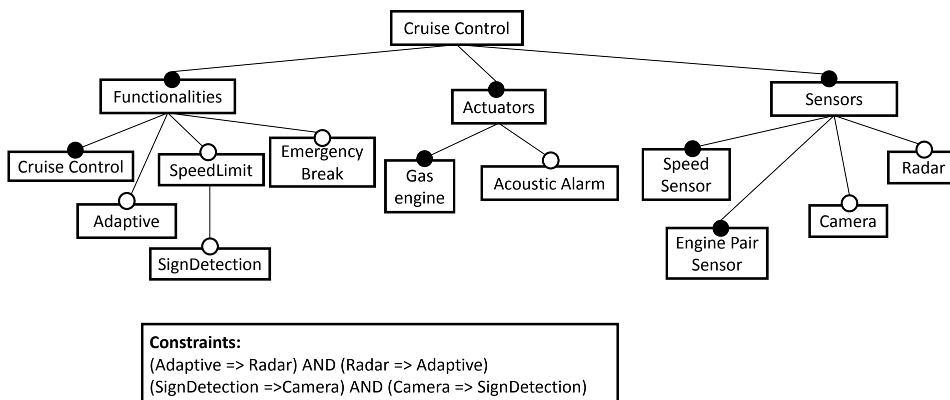



Figure 4.6: Feature model for the ACC example

### System Requirements

Together with experts from the automotive domain, twelve functional requirements were defined for the ACC case study (Table 4.3). Notice that each requirement has a feature associated, and if this feature is not selected in a specific product, the requirement is also unselected from the configuration.

### 4.4.3 Industrial Tank

The industrial tank presented in this section was developed during this thesis. It was a simple toy case study that was used to perform some initial validations of the developed

Table 4.3: System requirements of the ACC case study

| Reqs | Features involved | Requirement |
|---|---|---|
| $r_1$ | Cruise Control | If the cruise control is deactivated and the cruise control lever is pulled, the last chosen speed set point should be adopted as set speed |
| $r_2$ | Cruise Control | If no speed was set since the last start of the motor and the cruise control lever is pulled, the current vehicle speed is used as speed set point. If the current vehicle speed is below 20km/h, the speed is not adopted as speed set point and the cruise control is not activated |
| $r_3$ | Cruise Control | If the cruise control is deactivated and the cruise control lever is moved up or down, the current vehicle speed is used as speed set point. |
| $r_4$ | Cruise Control | If the driver pushes the gas pedal and by the position of the gas pedal more acceleration is demanded than by the cruise control, the acceleration setting as demanded by the driver is adopted. |
| $r_5$ | Cruise Control | By pushing the brake pedal, the cruise control is deactivated until it is activated again. |
| $r_6$ | Speed limit | By pressing the gas pedal beyond 90% the speed limit is temporarily deactivated. |
| $r_7$ | Speed limit | When the pressure on the gas pedal decreases below 90%, the speed limit is automatically activated again. |
| $r_8$ | Sign detection | If a road sign indicating a speed limit of F km/h according to the local traffic laws (i.e., maximum permissible speed) is recognized when driving with activated speed limit function, the speed limit is set to value F |
| $r_9$ | Adaptive Cruise Control | If the distance to the vehicle ahead falls below the specified speed-dependent safety distance, the vehicle brakes automatically. The maximum deceleration is 5m/s2. |
| $r_{10}$ | Adaptive Cruise Control, acoustical … | If the maximum deceleration of 5 m/s2 is insufficient to prevent a collision with the vehicle ahead, the vehicle warns the driver by two acoustical signals and by this demands to intervene. |
| $r_{11}$ | Adaptive Cruise Control | If the distance to the preceding vehicle increases again above the speed-dependent safety distance, the vehicle accelerates with a maximum of 2m/s2 until the set speed is reached. |
| $r_{12}$ | Acoustic alarm | The ACC warns the driver with an acoustic alarm if the actual distance to the vehicle ahead is less than (current speed/3.6)*t. |

methods. The idea of this example was to mimic similar functionalities that typical industrial CPSs involve. To this end, first, different functionalities were designed and discussed with experts in the field. After this, requirements were specified and some variability points included. Finally, the overall overview of the system was designed before implementing the model of the system in Simulink.

### System Architecture

The cyber layer of the tank is composed of two independent platforms connected among them with an EtherCAT communication system, as shown in Figure 4.7. Platform 1 corresponds to the layer in which the signal processing related to the system's sensors is performed. Specifically, this platform processes the data related to the level sensor, which measures the liquid level in the tank, the temperature sensor, which measures the temperature of the liquid, and the pH sensor, which measures the acidity of the liquid. This data is sent to the second platform, which its main

purpose is to perform the control at the actuators level. The embedded computer in the second platform also obtains the reference point related to the liquid level with a manual reference, apart from the sensor post-processed data. Based on all this data, the embedded controller decides whether the fill and drain gates should be opened or closed, as well as the different alarm systems turned on or off. Notice that the Sample Time (ST) of both platforms is 100 ms.

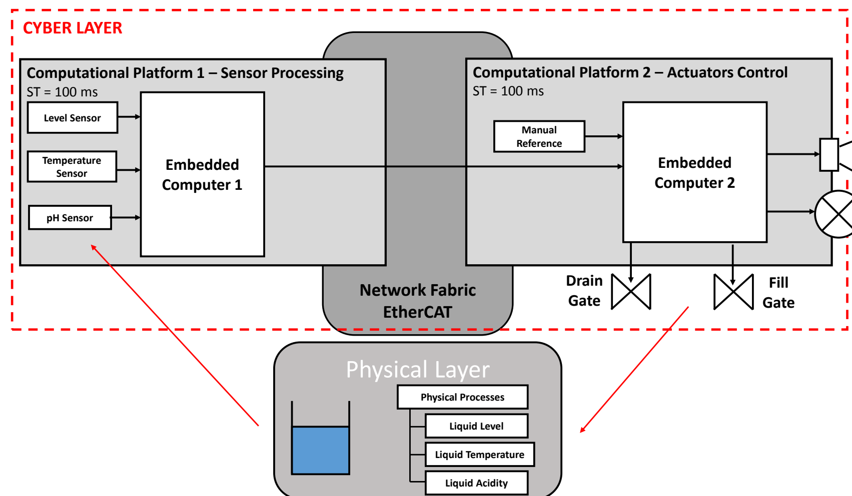



Figure 4.7: Overview of the structure of the case study corresponding to the tank

### Variability of the Industrial Tank System

The variability of the configurable CPS involving the industrial tank is depicted in Figure 4.8. The liquid stored in the tank can be either water or a chemical product. The feature model also involves some constraints (e.g., the chemical product would require a pH sensor that measures the acidity). The variability of the system is summarized bellow:

- **Variability in Sensors:** The system can include three sensors. While the sensor for measuring the level of the liquid is mandatory, a sensor for measuring the temperature will be optional and the inclusion of a sensor for measuring the pH will depend on the selected product (i.e., it will be included in case the stored liquid is a chemical product). In addition, each of the sensor types might use sensors from different vendors. Since this example is a toy case study, for the sake of simplicity, only two vendors for each type of sensor were selected.

- **Variability in Actuators:** The actuators in charge of draining and filling the tank are mandatory actuators. However, in addition to these two actuators, an alarm system is included to warn about system's dangerous situations. This alarm system is optional and can be either an acoustic alarm or a visual one (or both).

- **Variability in the Physical System:** The physical system has two main variability points. The first one refers to the previously mentioned liquid to be stored, which can be either a chemical liquid or water. The second variability point corresponds to the shape of the tank, which can be either cylindrical or conical. Notice that the dynamics of the system will be different depending on the selected tank shape.

- **Variability in Software**: The software of the proposed system must also deal with variability. Specifically, variability in the functionality must be considered (e.g., depending on the liquid to be manipulated, the maximum level will change). Furthermore, depending on the selected sensor vendor, the corresponding driver must be configured to avoid corrupted data to be sent to the second platform of the system, which is in charge of controlling the actuators.

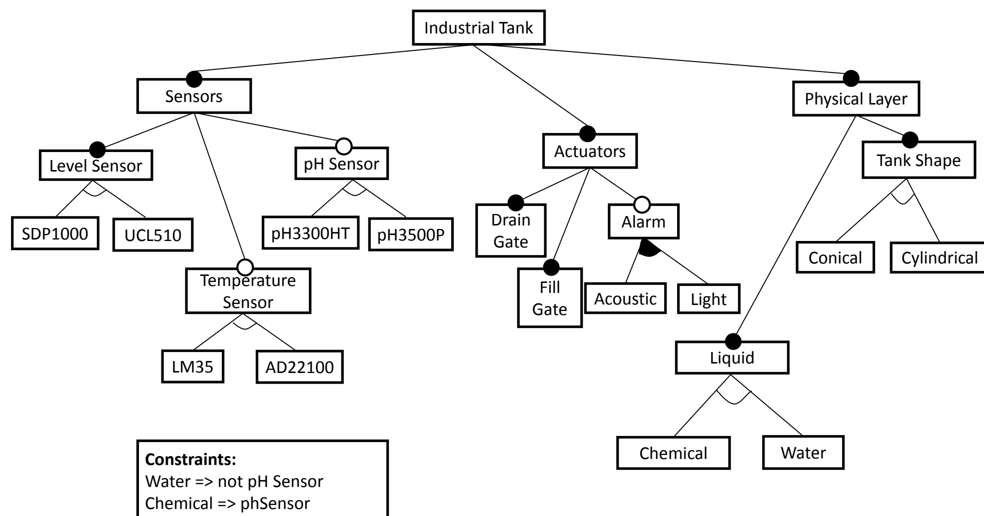



Figure 4.8: Feature model of the industrial tank system

### System Requirements

Six requirements were specified for the toy case study involving the industrial tank. Notice that, as in the case of the aforementioned case studies, the requirements are subject to variability, and thus, based on the selected features, the requirements might differ from product to product. Table 4.4 describes the selected requirements for the

industrial tank case study. As it can be appreciated, some requirements are composed of sub-requirements (e.g., $r_3$). The sub-requirements will be selected based on the selected features. For instance, for the third requirement, $r_{3a}$ will be selected when the liquid to be stored is water, whereas $r_{3b}$ will be selected when the liquid is chemical.

Table 4.4: System requirements for the Industrial Tank case study

| Reqs | Sub-reqs | Features involved | Requirement |
|---|---|---|---|
| $r_1$ | $r_1$ | Drain Gate, Fill Gate, Level Sensor | The level of the liquid shall not exceed the reference value provided by the operators in more than a 5% |
| $r_2$ | $r_2$ | Drain Gate, Fill Gate, Level Sensor | The maximum allowed time by the system to achieve the established level shall not exceed 50000 seconds |
| $r_3$ | $r_{3a}$ | Water, Fill Gate, Level Sensor | The level of the liquid shall not exceed the 90% of the capacity of the tank when the stored liquid is water |
| | $r_{3b}$ | Chemical, Fill Gate, Level Sensor | The level of the liquid shall not exceed the 75% of the capacity of the tank when the stored liquid is a chemical product |
| $r_4$ | $r_4$ | pH Sensor, Drain Gate | The liquid of the tank must be drained when the if its pH is less than 3.5 |
| $r_5$ | $r_{5a}$ | Drain Gate, Temperature Sensor, Water | The liquid of the tank must be drained if the temperature is less than 15°C |
| | $r_{5b}$ | Drain Gate, Temperature Sensor, Chemical | The liquid of the tank must be drained if the temperature is less than 15°C or more than 50°C |
| $r_6$ | $r_{6a}$ | Alarm,Water, Level sensor | The alarm system must be turned on if the level of the liquid is more than 85%. This alarm will be turned of manually by an operator. |
| | $r_{6b}$ | Alarm, Water, Level sensor | The alarm system must be turned on if the level of the liquid is more than 70%. This alarm will be turned of manually by an operator. |

### 4.4.4 Direct Current (DC) Engine

In this case, the case study was developed during the development of the Ph.D studies with the scope of validating the different methods (e.g., test case generation). Although the case study is simple, it contains similar functionalities that industrial CPSs (or parts of them) contain (e.g., safety-critical functionalities).

#### System Architecture

The system contains three main computational platforms, each of them containing a specific embedded computer. The communication between the three computational platforms are performed by a Time Triggered Ethernet communication system, which is a typical protocol for safety-critical systems. The first computational platform contains two emergency buttons. If any of them is pushed the system must be set to the safe state (i.e., the engine must be stopped). These functionalities are typical in factory automation systems involving robots, etc. The second computational platform

contains a temperature sensor that measures the engine's temperature. In case the engine temperature exceeds certain level, the system must be set to the safe state. These functionalities are typical in several domains such as railway. Finally, the third computational platform involves the speed control of the system, which takes into account a manual reference set by the user as well as the different functionalities implemented in platforms 1 and 2. This is the lowest level control of the system, which is performed with a feedback loop.

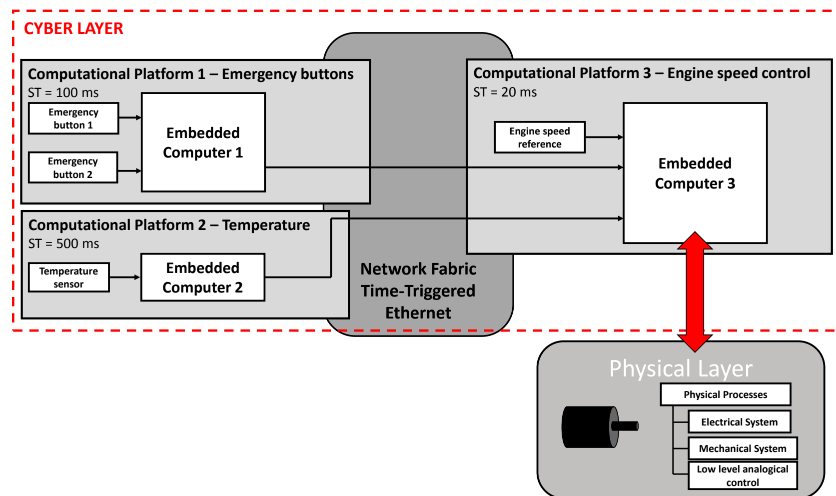



Figure 4.9: Overview of the structure of the case study corresponding to the DC Engine

## Variability of the DC Engine system

The variability of the system is captured in the feature model in Figure 4.10. Specifically, the feature model consists of eight features including two mandatory features and two optional features. Furthermore, three engine types can be chosen, which is also typical in CPSs depending on the context in which it operates (e.g., in an elevator system, depending on the allowed number of people and weight in the cabin, a more powerful or less powerful engine will be selected).

## System Requirements

The requirements of this case study are specified in Table 4.5. When assessing the scalability of the approaches presented in this dissertation, we defined a large number of so-called "artificial requirements".

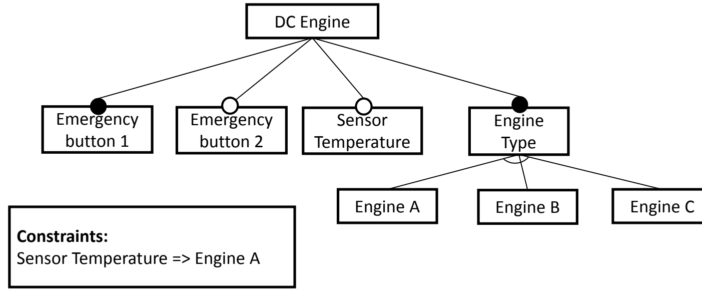



Figure 4.10: Feature model of the DC Engine system

Table 4.5: System requirements for the Industrial Tank case study

| Reqs | Features involved | Requirement |
|---|---|---|
| $r_1$ | Core features | The speed of the engine shall be stabilized within a maximum time window of 5 seconds. |
| $r_2$ | Emergency button 1, Emergency button 2. | The engine should stop within a maximum of 0.5 seconds when any of the emergency buttons is pushed. |
| $r_3$ | Temperature sensor | The engine should stop within a maximum of 1 second if the temperature of the engine exceeds 140 $^{o}$C |

### 4.4.5 Overview of the key characteristics of the case studies

The four case studies correspond to different domains. Furthermore, to ensure the sufficient degree of heterogeneity, the complexity of each case study is different. Table 4.6 summarizes the main characteristics of each case study. *Column* blocks is referred to the number of blocks of each Simulink model. Column *SS* refers to the number of stimuli signals of the Simulink model of the case study. *Depth* refers to the number of hierarchical level of the system model in Simulink. *FR* refers to the total number of functional requirements. Last, *Feature* and *constraints* columns refer to the number of feature and constraints related to the feature model of the case study. It is important to highlight, that for some evaluations, different versions of the case studies could have been employed, and thus, these characteristics could vary.

Table 4.6: Main characteristics of each case study

| | Blocks | SS | Depth | FR | Features | Constraints |
|---|---|---|---|---|---|---|
| UAV | 843 | 10 | 4 | 20 | 46 | 5 |
| ACC | 415 | 7 | 5 | 12 | 15 | 2 |
| TANK | 112 | 3 | 4 | 6 | 24 | 2 |
| DC ENGINE | 257 | 4 | 3 | 3 | 8 | 1 |

## 4.5 Case studies employed in each contribution

As mentioned before, each of the contributions was independently assessed by means of an empirical evaluation. For each evaluation, one or more case studies were employed. Table 4.7 shows which case study was employed in each of the contributions. Specifically, for the test system generation, the UAV case study was used. We selected the UAV case study because it was the largest case study in terms of number of requirements as well as number of Simulink blocks. For the test case generation as well as test case prioritization approach, the four case studies were used. As for the test case selection, the UAV case study as well as the tank case studies were used; we foresee to extend the evaluation by integrating the other two case studies. Finally, for the debugging approach, the UAV case study in addition to models from the SPLOT repository [MBC09a] were used. We employed models from this repository to ensure a sufficient degree of heterogeneity within the selected case studies.

Table 4.7: Case studies used in each of the contributions

| Contribution | UAV | ACC | Tank | DC Engine | SPLOT models |
|---|---|---|---|---|---|
| Test system generation | X | | | | |
| Test case generation | X | X | X | X | |
| Test case selection | X | | X | | |
| Test case prioritization | X | X | X | X | |
| Debugging | X | | | | X |

# Part II

# Automation

Chapter 5

# Automatic Test System Generation

Cyber-Physical Systems (CPSs) are ubiquitous systems that integrate digital technologies with physical processes. These systems are becoming configurable to respond to the different needs that users demand. As a consequence, their variability is increasing, and they can be configured in many system variants. To ensure a systematic test execution of CPSs, a test system must be elaborated encapsulating several sources such as test cases or test oracles. Manually building a test system for each configuration is a non-systematic, time-consuming and error-prone process. To overcome these problems, we designed a test system for testing CPSs and we analyzed the variability that it needed to test different configurations. Based on this analysis, we propose a methodology supported by a tool named ASTERYSCO that automatically generates simulation-based test system instances to test individual configurations of CPSs.

To evaluate the proposed methodology, we selected different configurations of a configurable Unmanned Aerial Vehicle, and measured the time required to generate their test systems. On average, around 119 seconds were needed by our tool to generate the test system for 38 configurations. In addition, we compared the process of generating test system instances between the method we propose and a manual approach.

## 5.1 Introduction

When CPSs have to be customized to clients demands, variability must be efficiently managed during all the development stages, which considerably increases the complexity of the system development and validation.[1] Configurable CPS development processes can be similar to those processes employed in product line engineering. In product line engineering, two main layers are considered: the domain engineering

[1]With variability we refer to configurability, i.e., variability in product space [TH02].

layer and the application engineering layer. The domain engineering layer involves different engineering tasks that consider variability of the product (i.e., variability in the product, in requirements, etc.). The purpose of the domain engineering layer is to create assets that will be reused in the application engineering layer. In the application engineering layer, the variability is resolved to create a specific system variant. The assets developed in the domain engineering layer are reused to create a variant, reducing the time for the development of different configurations.

However, although the use of simulation methods permits several advantages, testing CPSs is still expensive. One of the reasons of testing a specific system variant in the CPSs domain is expensive is because manually generating a test system for a configuration is an error-prone, non-systematic and time consuming process. The use of a test system permits a systematic validation of simulation models by automating the execution and evaluation of test cases, reusing the most relevant test cases across the different development stages or employing test optimization algorithms to improve the test execution time while maintaining the overall test quality.

We highlight two main contributions in this chapter: (1) a novel test system for testing CPSs employing simulation models and (2) a method for the automatic generation of test system instances for each configuration of configurable CPSs. For the first contribution, we have extended the test system proposed in [ZN08] for configurable CPSs by supporting different methods for testing particularities of CPSs (i.e., concurrency, timing behavior and unpredictability). The test system for each configuration is automatically generated by the proposed tool, which is considered the second contribution of this chapter. The tool obtains information of the variability parsing a feature model. After performing a set of model transformation the test system for testing a specific configuration is obtained. The tool allows full automation when generating the test system, what permits reducing the test system generation time as well as the error proneness.

The chapter is structured as follows: Section 5.2 gives a general overview of the proposed methodology for the automatic generation of test systems in the context of configurable CPSs. The meta-model of the designed test system for the validation of CPSs is explained in detail in Section 5.3. Section 5.4 presents our approach for the systematic generation of test system instances for configurable CPSs. A complete evaluation of the proposed approach is presented in Section 5.5. Section 5.6 positions our work with other previous studies in the current state of the art. Finally, conclusion and future work are outlined in Section 5.7.

In this Chaper, the case study presented in Section 4.4.1 is used to better explain some technical concepts.

The reader is encouraged to visit the following webpage for further information of the developed tool as well as for a demonstrative video:
https://sites.google.com/a/mondragon.edu/asterysco/

## 5.2 Overview of the Approach

The overview of the approach considered in this study to automatically generate test system instances is highlighted in Figure 5.1. It represents several points that test engineers must consider when using the tool we propose. The first point corresponds to the variability management of both the configurable CPS and the test system. In our case, FeatureIDE [TKB+14] is used for this task. It is important to systematically document the requirements of the system, highlighting the test cases employed to test them. In our approach, this is addressed in the second point, where we employ the tool Doors to capture all the requirements of the configurable CPS and tracing them with test cases. Once the domain requirements of the system are specified, in a third step, system engineers develop a 150% model of the configurable CPS in Simulink. 150% models address all the variability that the system can have, which is later pruned by a system configurator. In the fourth point, engineers develop the infrastructure including test assets that will be reused at the application engineering layer for testing the configurable CPS. This infrastructure considers executable test cases, different requirement monitors and a context environment employed for simulating the environment in which the CPS operates. The fifth step corresponds to the selection of configurations to test.

Generation of test systems is performed in the sixth step in the proposed approach. This part is explained in detail in Section 5.4 of this chapter. Manually generating a test system for each configuration is infeasible and thus, an automated solution is needed. For this we propose ASTERYSCO (Automatic Simulation-based TEst system geneRator for cYberphyscal Systems COnfigurations), a tool that automatically generates test systems to test the different system variants that a highly configurable CPS can be set to. ASTERYSCO (1) processes information of the feature model as well as information of each specific configuration to test, (2) obtains and configures the test cases, requirement monitors and the context environment functions needed to test the configuration and (3) generates the test system for the chosen configuration in Simulink. The generated test system is in accordance with a previously designed test system. The metamodel of the designed test system for testing CPSs is presented in Section 5.3. Our approach is designed for early validation of configurable CPSs at system level as testing starts very early when simulation models are available

[MNBB16]. In a seventh step, the test cases for each configuration are executed in Simulink employing the test system that is automatically generated with our tool. The eighth step in Figure 5.1 consists of the analysis of the results of the tests by system engineers so that they can fix faults in the configurable CPS.

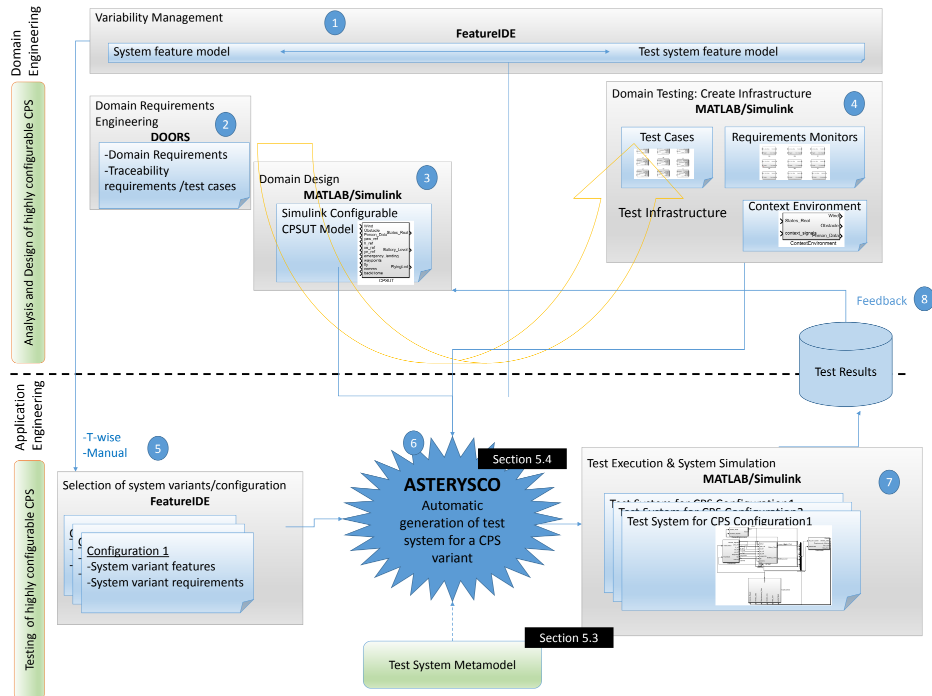



Figure 5.1: Technological overview of ASTERYSCO and its dependencies

## 5.3 Test System for CPS Validation

This section explains the test system for the early validation of configurable CPSs. We took the test system proposed in [ZN08] as a base. Nevertheless, this system was oriented for embedded systems from the automotive domain as well as for non-configurable systems. Thus, we extended that test system for testing configurable CPSs.

### 5.3.1 Test System Meta-model

The highest level of the meta-model for our test system, which is illustrated in Figure 5.2, is composed of five different elements:

- Cyber-Physical System Under Test (CPSUT): The system to be tested. In our case, the CPSUT is a model composed of the cyber and the physical layer.
- Test Stimuli: The source in charge of generating the necessary data to stimulate the CPSUT and to control part of the behavior of the context environment.
- Test Oracle: Observes the behavior of the CPSUT with respect to its inputs and decides whether the results fulfill functional requirements or not.
- Test Control: The source that control the execution of test cases and decides when the test must be finished.
- Context Environment Model: Simulates the physical world in which the CPS resides and has to interact with.

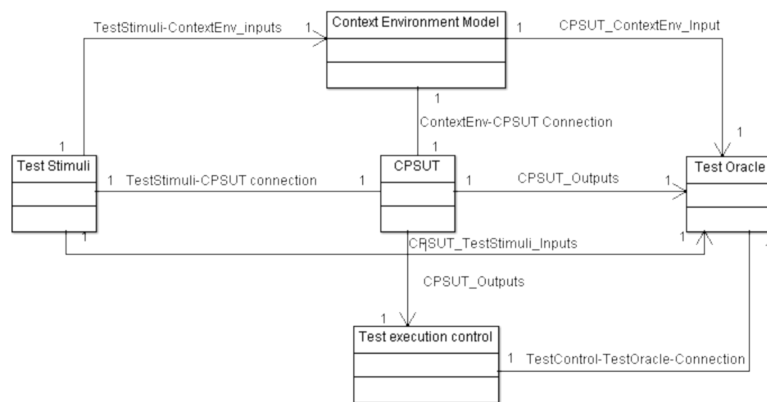



Figure 5.2: Test system meta-model. The test stimuli source stimulates CPSUT, the context environment model simulates the context in which the CPSUT resides, the test oracle observes the behavior of the CPSUT and decides the test result and the test execution control sequences the test cases.

### Test Stimuli

Figure 5.3 depicts the overview of the test stimuli block. This block encapsulates the different test cases for validating the system requirements. A test case refers to a set of signals that stimulate the inputs of the system. Each functional requirement is validated with a set of test cases (unlike in [ZN08], where each requirement is validated with a single test case). An algorithm named Test Trigger Control sequences the test case execution for each requirement. We also provide support for reactive test cases, which are test cases that observe the state of the CPS and take decisions

accordingly (Section 2.3.2). Reactive test cases are very common in CPSs as they support the unpredictability of the physical world by observing the state of different parts of the CPS. “FeedbackSignals” contain signals related to the state of different parts of the CPS.

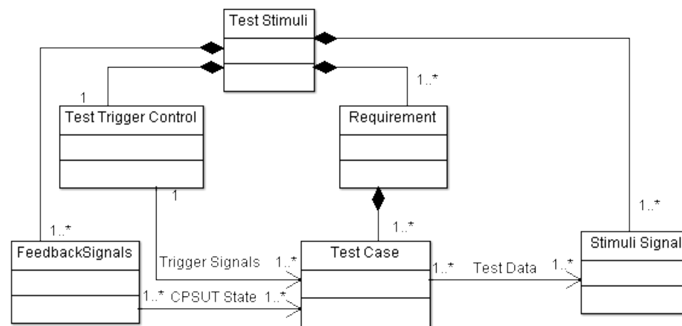



Figure 5.3: Test stimuli meta-model of the test system employed to send the generated test data to the CPSUT

## Test Oracle

The main structure of the test oracle is depicted in Figure 5.4. The test oracle observes the behavior of the CPSUT and decides whether the system fulfills functional requirements. The test oracle examines each requirement independently using one or more requirement monitors. For starting the validation of a specific requirement, the block named “RequirementTriggerControl” encapsulates an algorithm that triggers each requirement. A requirement monitor is a piece of software that determines the requirement status (pass, fail, inconclusive) taking into account a stream of significant input events [Rob10]. We employ one or more requirement monitors to assess the concurrency of the CPS. A requirement monitor is composed of a precondition and a post-condition. The precondition specifies when a requirement monitor can evaluate a requirement, whereas the post-condition evaluates whether the requirement is valid or not. Each post-condition consists of a property verifier and a verdict generator. On the one hand, the property verifier observes that the state of the system is correct. On the other hand, the verdict generator evaluates whether the properties for the requirement meets the specified requirements. When the verdict generator evaluates the validity, it generates a verdict called Requirement-Monitor Verdict (RMVerdict), which is processed by the requirement validator. The data generated by the requirement validator is sent to the arbiter. The arbiter processes and analyses the verdicts of each requirement, and generates a verdict that represents the result of the executed tests.

The arbiter is composed of the coverage calculator and the validation algorithm, which are employed to give the overall test result. First, the coverage calculator calculates the amount of test coverage the tests have achieved (i.e., in our context, the number of requirements that have been covered). This data is transferred to the validation algorithm to give the result of the overall test.

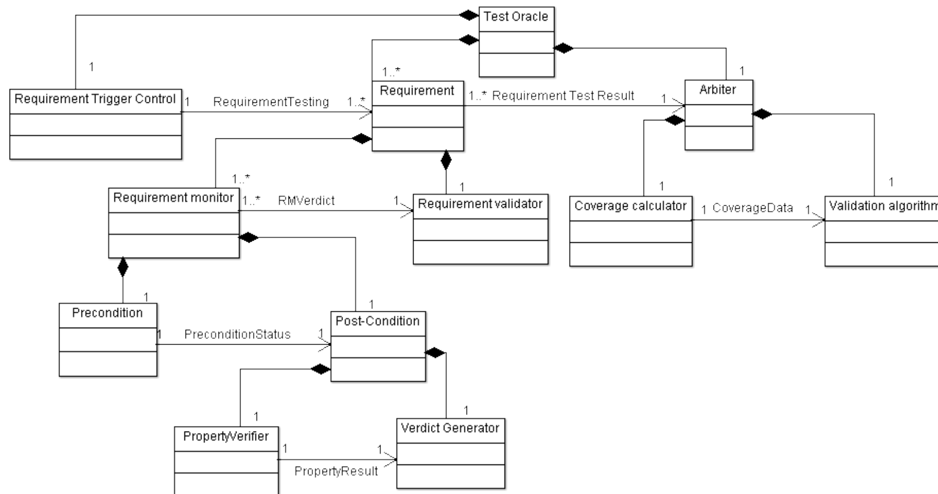



Figure 5.4: Test oracle meta-model of the proposed test system for the decision of the test result

### Context Environment

A CPS works inside an environment. Two kind of environments can be differentiated: (1) the context environment, and (2) the irrelevant environment. The former refers to the environment that directly affects the CPS, thus, engineers must take this environment into account when validating CPSs. For example, in the case study of the Unmanned Aerial Vehicle (UAV) presented in Section 4.4.1, the context environment includes a function that models the wind, another function that models the objects as well as other kinds of hazards that the drone can be exposed to and last, the position of the person that must follow (if that feature is selected). The irrelevant environment refers to the environment that does not affect the CPS, even though it is operating within it.

The context environment is included in our test system to deal with the unpredictability of the physical world. Figure 5.5 depicts the metamodel of this part of the test system. The context environment models different scenarios, called context environment functions, in which the CPS can be involved. There can be one or more

context environment functions. As mentioned in the previous paragraph, we model the wind, the obstacles that the CPS has to face and in the case that the UAV is configured to follow a person, the position of that person. The context environment communicates with the TestStimuli source (Stimuli Outputs). This is important as the test stimuli sends orders to the context environment (e.g., generate heavy wind). In addition, the context environment also obtains data related to the CPS via the CPSUT outputs (e.g., in the provided example, the altitude of the drone is obtained, so that as the drone gains height, the wind is heavier). After processing this data, the context environment sends data that can affect the CPSUT (such as wind or obstacle) via the "CPSUTInput."

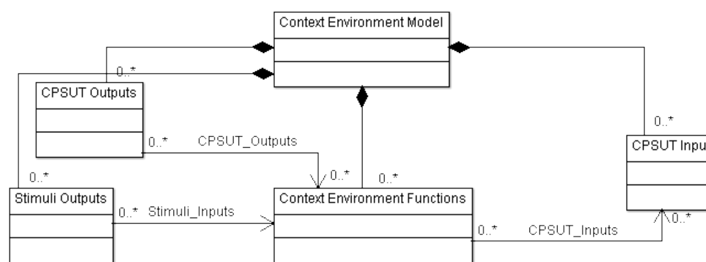



Figure 5.5: Context environment metamodel of the test system for the simulation of the environment in which the CPS operates

### Test Control

The test control algorithm we defined is quite different to Zander's test system. In [ZN08] the test cases were executed following a time-, logical condition-, and verdict-triggered strategy. In contrast, to support the cost-effectiveness that the configurable CPSs demand when testing, we have divided it into three main parts: the Next Test Generator (NTG) algorithm, the Test ID Generator (IDGen) algorithm and the Test Ending Mechanism. The NTG algorithm decides **when** to execute a new test case, i.e., its main control is to detect when the test that is being executed has finished testing the CPSUT. When a new test case has to be executed, the NTG algorithm sends a boolean signal to the IDGen algorithm. The IDGen algorithm decides **which** test case to execute. Finally, a subsystem named "end mechanism" decides when to finish the test. Figure 5.6 depicts the architecture and the interactions of the different sources in charge of controlling the test at simulation level. For more information about different test control strategies for the cost-effective validation of CPSs refer to our previous work [ASE15a].

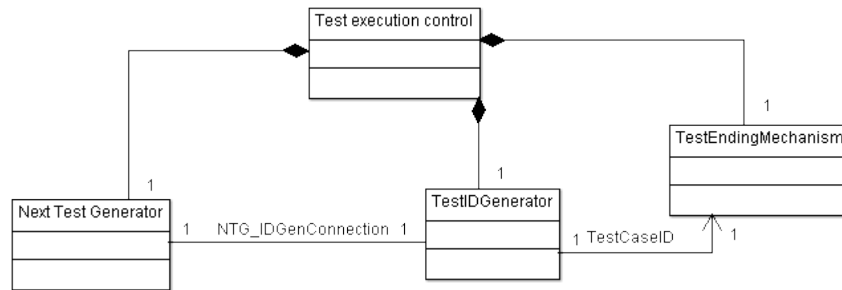



Figure 5.6: Test execution control meta-model proposed to control the execution of the test cases, the block to the left belongs to the NTG algorithm, the one in the middle to the IDGen algorithm, and the one in the right to the "End mechanism"

### 5.3.2 Summary

Although some parts have been inspired by the previous work of [ZN08], we extended different parts of the test system for testing particularities of configurable CPSs. CPSs are concurrent systems, which means that several tasks are performed simultaneously. To support concurrency, the test oracle we propose monitors each requirement with one or more requirement monitor. Each requirement monitor is capable of observing a specific task of the CPS. The unpredictability of the physical world is another particularity of CPSs. Our test system supports the testing of this issue by (1) including a new block named Context Environment and (2) employing reactive test cases. The former models the physical world in which the CPS resides simulating different scenarios. The latter refers to the test stimuli block, which encapsulates different test cases capable of observing different states of the system and taking decisions based on them. Finally, one particularity of configurable CPSs is the need for cost-effectively testing. This is because many configurations have to be tested and simulating each configuration is expensive. Our test system supports cost-effectiveness by employing an efficient test control strategy that can combines test case selection and test case prioritization (both approaches are explained in Chapters 7 and 8 respectively).

## 5.4 ASTERYSCO: Automatic Test System Generator

This section explains the approach to automatically generate test system instances for each system variant. Figure 5.7 depicts the overview of the processes performed by the developed tool, named ASTERYSCO, for the automatic generation of the test system in a Software Process Engineering Meta-model (SPEM). The system takes a set of external files as input. The output result of the process is a test system ready

to test the specified configuration of the CPS. The arrows with the solid line indicate the control flow, whereas the arrows with the dashed lines indicate the object flow (i.e., inputs and outputs of each process). In this case, the first activity is the test system generation, and the second activity corresponds to the configuration of the test system for a specific system variant. In the first process the inputs are the feature model, the CPSUT and the context environment; the output of the first process is a Generic Test System, which is an input to the second process. The rest of the inputs for the test system configuration process are the test cases, the configuration file and the requirements monitors. The output of this process is the test system instance for the selected configuration.

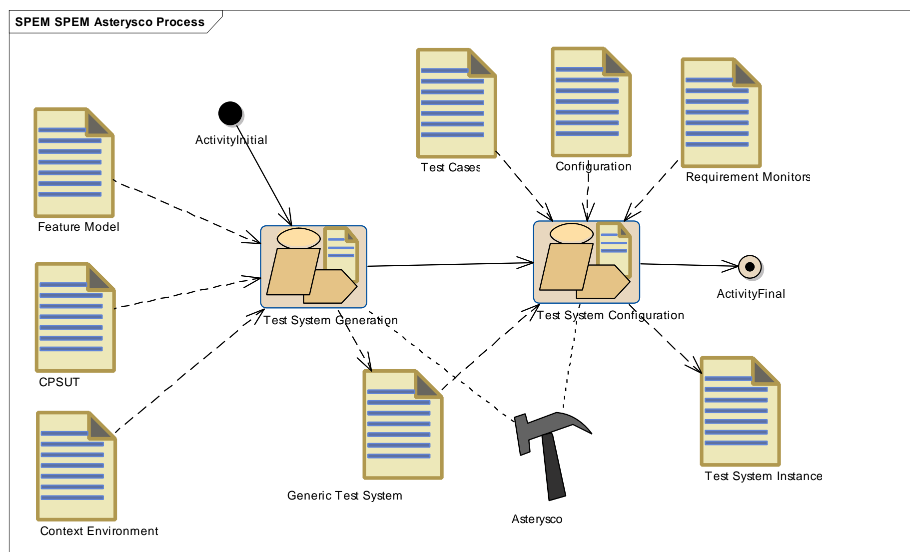



Figure 5.7: Overview of ASTERYSCO's activities for the generation of a test system instances in SPEM.

The test system we choose is the one explained in Section 5.3. The generated test system is a Simulink model. Simulink was chosen because it is a prevalent modeling language for CPSs [MNBB16]. It allows simulation of heterogeneous models, encompassing software, network and physical units. Moreover, it allows different variability mechanisms [DLPW08, PMB+12]. In addition, Simulink supports CPSs modeling by employing different toolboxes for computer vision and signal processing, concurrency modeling of computing platforms, graphical state machine modeling, system-level physics modeling, communication and computation modeling and in-the-loop test levels [Mat16]. Feature modeling, which is widely used in industry [BRN+13], was employed for variability management.

For generating different test system instances it is important to analyze the variability points of the test system. Depending on the chosen system variant or test objective, the test system has to be adapted and automatically configured. To achieve this goal, the different elements of the test system have to deal with variability. Figure 5.8 depicts a taxonomy summarizing the different variability points of the designed test system.

The variability of the Test Stimuli strongly depends on the selected product configuration: variability can be found in the number and type of requirements, and as consequence in the test cases in charge of testing these requirements. Depending on the number of test cases needed to test a specific system variant, the test control source must also be adapted. The inputs of the CPSUT can vary from system variant to system variant, and therefore, the stimuli signals have to be updated. In addition, the test cases are reactive, which means that they act taking into account different signals and variables of the CPSUT. Depending on the system variants, these signals can also vary.

The test oracle also depends on the selected system configuration. Variability in this system can be found in the number and type of requirements. As a consequence, the signals of the test trigger control must be adapted. Furthermore, each requirement might also have one or more requirement monitor, which have to be configured for the selected system variant, thus, variability can also appear in the signals of the precondition or in the signals of the property verifier. The final verdict is given by the arbiter, which has to be adapted to the number of requirements in the system. Variability can also be found in this arbitration algorithm, as two algorithms were developed with different test objectives (i.e., validation based on the percentage of requirements covered or validation based on the obtained verdicts). In addition, the test oracle has to be adapted to the input and output signals present in the CPSUT system variant.

With regard to the context environment, depending on the selected system configuration, there are some functions that are inside the context environment and others inside the irrelevant environment. The functions that are inside the irrelevant environment of the selected configuration have to be removed from the test system when configuring it with the objective of saving simulation time.

Two main steps are taken to generate a test system for a specific configuration: (1) generation of the generic test system and (2) instantiation of the generic test system for a selected configuration. In the first step, the test system generator reads the variability management file, and a generic test system is automatically generated. This test system generator uses test system specific algorithms (e.g., arbitration algorithms,

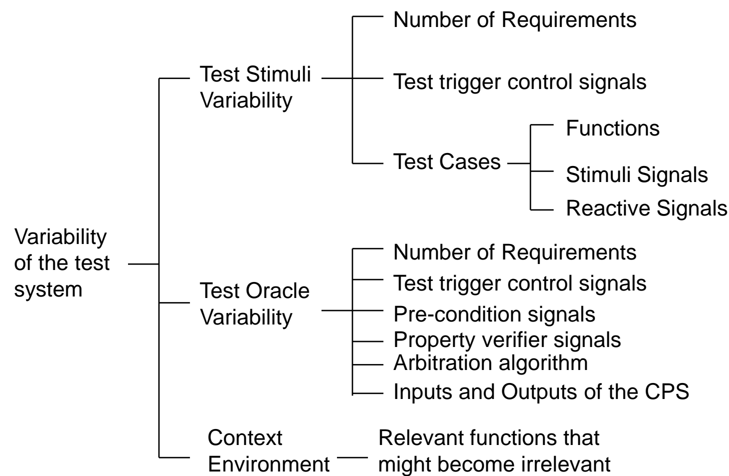



Figure 5.8: Taxonomy of the variability for the chosen test system

test control algorithms, etc.) that have been developed and stored in a Simulink library, and are the same for any system. In addition, it also takes the model of the CPSUT as well as the model related to the context environment in which the CPSUT operates. Once the generic test system is generated, this process is not needed to be performed again (unless there are modifications such as requirement changes, etc.).

In the second step, the generic test system is reused by the test system configurator, which automatically rebuilds it to obtain the test system compliant with the system variant. In this step, the test system configurator replaces the elements of the test system with the elements necessary to test the configuration. Finally, it removes the elements of the test system that are not needed to test the selected system variant.

### 5.4.1 Input Files

As shown in Figure 5.7, a set of input files are needed for the generation of the test system. These inputs include:

- A variability management file which is manually developed by test and system engineers.
- A model of the CPSUT, which is developed by system engineers.
- A model related to the context in which the CPS operates, which is usually built by test engineers.

- A configuration file, which is related to a specific system variant that must be tested. This can be either automatically generated by combinatorial algorithms or manually, for testing specific user needs.
- A set of test cases, which can be either automatically generated from a behavioural System Under Test (SUT) model or manually specified by test engineers.
- A set of requirement monitors, that are employed by the test oracle to verify that each requirement meets the specifications provided by the test engineers.

### Variability management

For generating the test system instance, one of the most important files is the variability management file, which is in charge of specifying the variability of both the CPSUT, as well as the test system. The tool could clearly be used in a wide range of industrial applications. According to the survey conducted by [BRN+13], feature modeling is the most used notation in industry to manage variability, where 74.3% of respondents reported using feature models for this task. Moreover, industrial practitioners are not usually familiar with modeling notations [WAGL16]. However, researchers in the field of testing have demonstrated that feature models can be appropriate for industrial test engineers (e.g., test engineers from Cisco [WAGL16]). In addition, we use Simulink models of the system, and thus the variability we consider is limited to the system model level. For the cases we have considered, feature models have supported our needs. Further, an industrial feature modeling tool named pure::variants [PS14] has demonstrated that feature models can deal with variability of Simulink models. The Feature Modelling tool chosen was FeatureIDE [TKB+14]. The reason for choosing FeatureIDE was that it is a robust tool, which employs an intuitive user interface. Moreover, this tool is open source, which allows for the adaptation of the tool to our needs. It also supports both the automatic selection of system variants employing search algorithms as well as manual selection of products employing a user interface. Other variability notations can also be appropriate for the variability management of configurable CPSs (e.g., SimPL [BRN+13], Clafer [BDA+15] or Zen-CC [LYAZ16b]). We believe that ASTERYSCO can be adapted to support these tools.

On the one hand, system engineers develop the feature model related to the configurable CPS, taking into account the variability of both the Cyber and the Physical layers. On the other hand, test engineers build the feature model related to the test system, taking into account the variability of the test system. For the chosen test system, the variability is summarized in Figure 5.8. Note that when the test

system for a specific configuration has to be generated, the tool must take into account several test related issues. The requirements of the selected configuration as well as the associated test trigger control signals have to be taken into account (for both the test stimuli and test oracle). The test cases associated to the configuration must also be allocated inside the test stimuli source, where the test stimuli signals (i.e., the ones that are employed to stimulate the CPSUT) corresponding to the configuration must be instantiated. This has an impact in the inputs and outputs that are evaluated by the test oracle, where the ones related to the specific configuration must be selected, and the not selected ones must be removed from the simulation model. The precondition and property verifier signals must also be allocated for the selected configuration (these ones in our test system are related to specific requirements). In addition, the arbitration algorithm used for the automatic evaluation must be specified (in our case this is specified beforehand). Finally, the different functions related to the context environment have to be taken into account, as not all the functions have an influence in all the configurations (e.g., if in a configuration of the UAV example (Section 4.4.1) the collision avoidance feature is not chosen, the function related to the obstacle is removed from the context environment).

When both feature models are developed (i.e., the one related to the system and the one related to the test system), test engineers integrate them, and trace the elements related to the CPSUT with the elements related to the test system. This traceability is done with the constraints of the feature modeling technique, i.e., with "requires" and "excludes." For this integration it is important to correctly interpret system requirements, and how each system requirement is associated with the features related to the system. It is important to highlight that in this stage there must be an efficient communication between the system engineer and the test engineer as some misunderstandings might appear. For instance, the name given to a specific system feature can be inappropriate so that the test engineer can interpret its association with a specific system requirement. The traceability is important when configuring a specific system variant, in this way the elements of the test system are automatically selected when a system variant is chosen.

Figure 5.9 depicts the test feature model used to manage the variability of the test system for the proposed case study.[2] The test system is divided into three main parts: Requirements, Stimuli Signals and Context Environment. In each part, child features are allocated in terms of optional (e.g., $r_{20}$, yaw_ref or Obstacle) or mandatory (e.g., $r_1$, context_signals or Wind). After developing the test feature model for managing the variability of the test system, the following step would be to integrate it with the

[2]For presentation reasons, we could not allocate every single feature of the test system.

CPSUT's feature model.

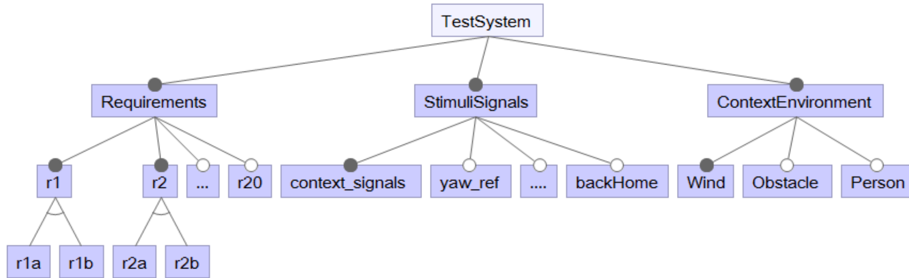



Figure 5.9: Test system feature model without being integrated with the CPSUT

The integration with the CPSUT is mainly important for the automatic configuration of the test system. When the feature model that manages the variability of the CPSUT and the test feature model are integrated, the features of both need to be traced with cross-tree constraints. Thus, when a specific system variant of the CPSUT is chosen, the elements related to the test feature model are automatically selected. As an example, the requirement for the flying LED is $r_{20}$: "The drone shall turn on the flight LED while it is in the air." Therefore, to automatically select $r_{20}$ when the flying LED is part of the system variant, the cross-tree constraint between the CPSUT feature model and the test system is the following: $Fly_Led \Rightarrow r_{20}$. In addition, when the flying LED is not part of the system variant, $r_{20}$ must be automatically unselected. For this case, the cross-tree constraints would be the following: $\neg Fly_Led \Rightarrow \neg r_{20}$.

### CPSUT

Another input when generating the test system is the CPSUT model, which is a 150% model of the CPSUT. 150% models integrate all the variability, i.e., the variability related to the whole product line; when a specific system variant needs to be selected, the variability of the 150% model is bound. The CPSUT is part of the input, as the test system needs to be integrated with it. Figure 5.10 shows the input model of the CPSUT related to the case study provided in Section 4.4.1. Note that the output "States_Real" is a bus with information related to the UAV.

### Context Environment

In Section 5.3.1 we described the context environment. The context environment model is taken as an input. This model simulates the behavior of the environment that directly affects the CPSUT. For the example provided in Section 4.4.1, the wind, obstacles and information related to the person that the UAV must follow (if the

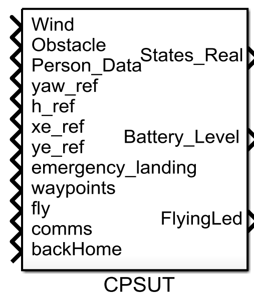



Figure 5.10: CPSUT input model of the AR.Drone Simulink model

functionality is chosen) is modeled. Figure 5.11 depicts the context environment model for our case study.

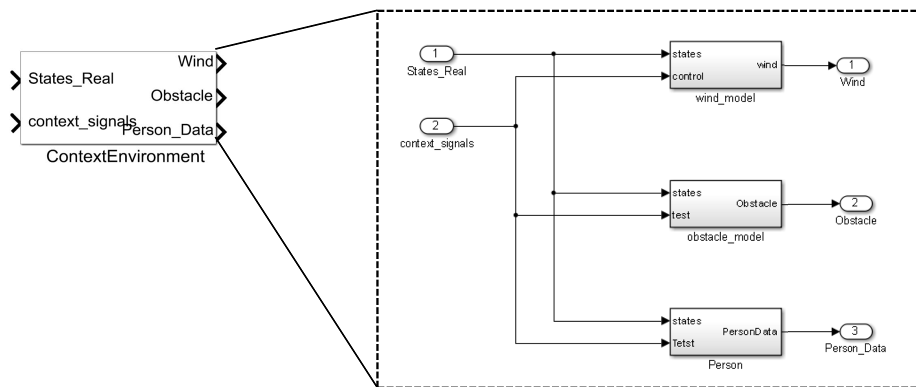



Figure 5.11: Context environment model for the AR.Drone Simulink model

### Configuration

The configuration file expresses the elements of both the CPSUT and the test system, for the selected system variant. System variants can be generated either automatically or manually. When generating system variants automatically, a very typical technique used in configurable systems testing is Combinatorial Interaction Testing (CIT). The configurations can also be specified manually. FeatureIDE provides a user interface to perform this task. This can be useful when a customer wants a specific configuration, and before the configuration is built, it must be tested and validated by a simulation tool. The Test Engineer chooses a specific CPSUT configuration, and the elements of the Test System are automatically selected; this is possible as we have previously traced the components of the configurable CPS with the elements belonging to the test system. When all the features from the feature model are selected, the configuration file

(which has a *.config extension) related to the chosen configuration is automatically generated by FeatureIDE.

As an example, we manually customized the C.AR.Drone. For this case, we chose a simple configuration, $c_1$, with the following features: $F_{c_1} = \{TrackSystem, Point_To_Point, PI, ConcreteCoordinates, BatteriesManagement, BatteryLanding, Short_Duration, GyrA, GPS_A, BS_A, LowSpeed\}$. The elements of the test system needed to test $c_1$ are automatically selected:

- Requirements: $R_{c_1} = \{r_1, r_{1a}, r_2, r_{2a}, r_3, r_{3a}, r_4, r_{4a}, r_{10}, r_{11}, r_{12}, r_{13}, r_{16}, r_{17}, r_{18}, r_{19}\}$
- Stimuli Signals, $SS_{c_1} = \{context_signals, yaw_ref, h_ref, xe_ref, ye_ref, waypoints, fly, comms\}$
- Context Environment, $CE = \{Wind\}$

#### Test Cases

Other inputs to ASTERYSCO are the test cases needed to test the CPSUT. Although this is not part of the contribution of this chapter, our proposal for test case generation is described in Chapter 6. However, there are other options to generate reactive test cases. For instance, the tool Time Partition Testing (TPT), from piketec, is a Model-Based Testing (MBT) tool for testing embedded control systems [Pik15]. This methodology allows, among others, graphical test modelling, automatic test execution, etc [BK08]. This tool also permits testing MATLAB/Simulink models, ASCET models, AUTOSAR software, C-code, etc. There are other approaches for the automatic generation of reactive test cases (e.g., [ZN07], [Mje13]).

#### Requirement monitors

To obtain the results of the executed test cases, the test oracle uses a set of requirement monitors. These requirement monitors observe several properties of the test system and generate a set of test verdicts. The requirement monitors are taken as an input to ASTERYSCO. These requirement monitors also encapsulate variability, which is bound by the test system configurator for the selected system variant.

### 5.4.2 Test System Generation

The first step consists of generating a generic test system, which is later used by the test system configurator to obtain the final test system. Figure 5.12 shows the tasks performed to generate the generic test system. The first task corresponds to the parser, which reads the variability management file. During the second task, the

test system elements are automatically generated. To generate these elements, the data provided by the parser is processed and the required elements are taken from the outside libraries and allocated in a model. The CPSUT model is also allocated in the same model. Finally, this model is sent to the test system integrator, which automatically integrates the input and output ports of the different elements allocated in the model.

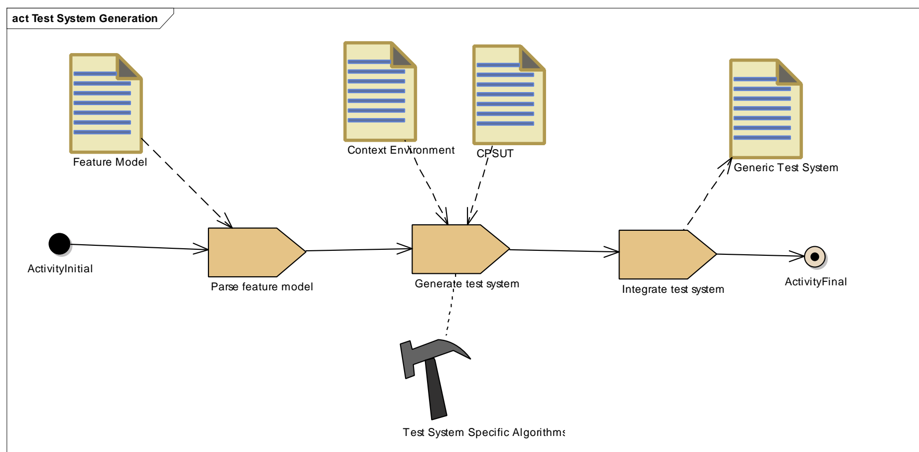



Figure 5.12: Overview of the tasks of the test system generator for the generation of the generic test system model.

ASTERYSCO was developed in MATLAB. The variability management tool is FeatureIDE using feature models. The models in FeatureIDE are saved in *.xml. For the chosen test system, when generating the Test Stimuli, the test system generator needs to process the name and number of the requirements and stimuli signals. When generating the test oracle, the name and number of each requirement is needed. Thus, the parser extracts all this data to send it to the test system elements generator. Once this data is extracted, the test system elements generator uses MATLAB scripts to generate the test system elements in accordance with the architecture explained in the Section 5.3.1. Finally, the test system integrator reads the test system elements model and joins all its ports.

Integrating the test stimuli, the test control and the test oracle is systematic, as the ports that have to be integrated do not vary. Nevertheless, the output signals of the test stimuli, the signals of the context environment and the signals of the test CPSUT vary from system to system. The automatic integration of these sources is possible when the names of the input and output ports to be linked are the same. Otherwise, a manual refinement process is needed for the generation of the whole test system.

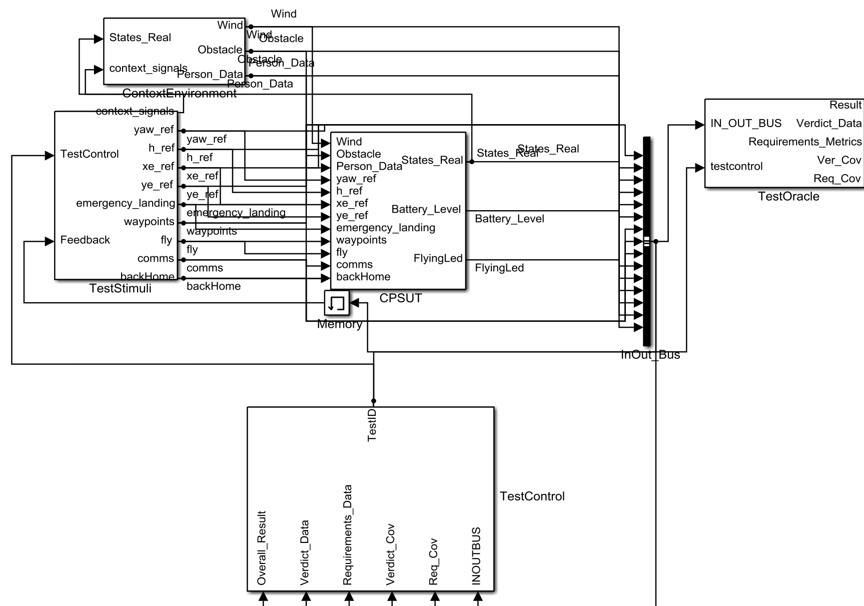



Figure 5.13: The automatically generated generic test system model

This integration is automatically performed with Algorithm 1. Figure 5.13 depicts the automatically generated generic test system model for the proposed case study by our test system generator.

### 5.4.3 Test System Configuration

Once the generic test system is generated, the test system configurator needs to rebuild it to test a specific system variant by generating the test system instance. The test system configurator works as follows: first, it parses the needed information from the variability management file as well as from the configuration file. The test system configurator relates the selected features in the configuration file to the features of the variability modeling file (which are generic files). After this relationship has been established, the elements related to the configuration file are replaced by the elements corresponding to the variability modeling file. Finally, the non-selected elements are deleted from the model.

FeatureIDE saves the models in *.xml format and the features of a specific system variant in *.config format. For the selected test system, we focused on requirements and stimuli signals. The configurator replaces the test cases related to the generic test system (which are empty subsystems), with the test cases for the specific system variant. The variability of the test cases is also bound (e.g., a specific signal that is not needed for the system variant might be deleted, etc.). The same happens with

```
   Data: Test System Elements model
   Result: Generic Test System
1  TestSystem = Read(Test System Elements model);
2  for i=1 to size(TestStimuli.Outputs) do
3  |  for j=1 to size(ContextEnvironment.Inputs) do
4  |  |  if TestStimuli.Outputs(i).Name == ContextEnvironment.Inputs(j).Name
   |  |  then
5  |  |  |  addLine(TestStimuli.Outputs(i),ContextEnvironment.Inputs(j);
6  |  |  end
7  |  end
8  |  for j=1 to size(CPSUT.Inputs) do
9  |  |  if TestStimuli.Outputs(i).Name == CPSUT.Inputs(j).Name then
10 |  |  |  addLine(TestStimuli.Outputs(i),CPSUT.Inputs(j);
11 |  |  |  addLine(TestStimuli.Outputs(i),InOutBus);
12 |  |  end
13 |  end
14 end
15 for i=1 to size(ContextEnvironment.Outputs) do
16 |  for j=1 to size(CPSUT.Inputs) do
17 |  |  if CPSUT.Outputs(i).Name == ContextEnvironment.Inputs(j).Name
   |  |  then
18 |  |  |  addLine(ContextEnvironment.Outputs(i),CPSUT.Inputs(j);
19 |  |  |  addLine(ContextEnvironment.Outputs(i),InOutBus);
20 |  |  end
21 |  end
22 end
23 for i=1 to size(CPSUT.Outputs) do
24 |  for j=1 to size(ContextEnvironment.Inputs) do
25 |  |  if CPSUT.Outputs(i).Name == ContextEnvironment.Inputs(j).Name
   |  |  then
26 |  |  |  addLine(CPSUT.Outputs(i),ContextEnvironment.Inputs(j);
27 |  |  end
28 |  end
29 end
```

**Algorithm 1:** Algorithm for the automatic integration of the Test Stimuli, CPSUT and Context Environment

the requirement monitors, where the generated requirement monitors are replaced by configuration-specific requirement monitors. Lastly, the generated empty test cases and empty requirement monitors that do not represent the chosen configuration are removed. Algorithm 2 shows the strategy followed for the configuration of the requirements.

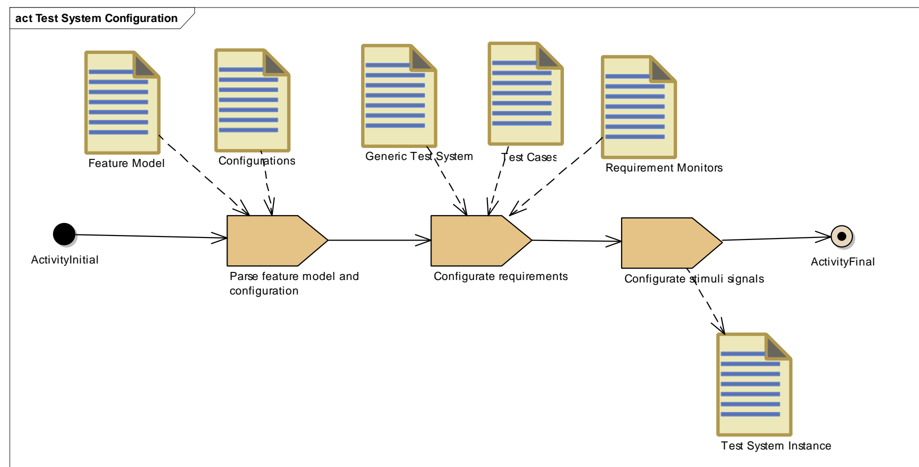



Figure 5.14: Overview of the processes corresponding to the test system configuration for a specific system instance.

In the case of the stimuli signals, the process is easier. The stimuli signals that are not needed to test a specific system variant are automatically removed. Algorithm 3 shows the strategy for configuring the stimuli signals.

Figure 5.14 shows the processes that ASTERYSCO performs to configure the generic test system for a specific test system instance. Note that the requirements configurations as well as the stimuli signals configurations are based on the chosen test system variability. In the case that another test system is chosen, these blocks should be replaced based on the variability points of the selected test system. In ASTERYSCO, after removing the requirement monitors of the test oracle, the verdict and coverage bus creator loses the signals of the removed requirement monitors. This is a problem for the arbitration algorithm when the different metrics have to be calculated. Thus, these two bus signals must also be adapted. Algorithm 4 configures these bus creators.

### 5.4.4 Output

Figure 5.15 shows the final results of the test system to test the chosen configuration. The non-selected components and signals have been removed from the model. For instance, the ports corresponding to “emergency_landing” or “back_home” from

```
   Data: GenericTestSystem, FeatureModel, ConfigurationFile
   Result: TestSystemInstance
 1 FM = parse(FeatureModel);
 2 CONFIG = parse(ConfigurationFile);
 3 for i=1 to size(CONFIG.Req) do
 4     Req = Config.Req(i);
 5     for j=1 to size(FM.Req) do
 6         if FM.Req(j).Children == 0 then
 7             //This means that it does not have children requirements
 8             if FM.Req(j) == CONFIG.Req(i) then
 9                 req_id(j) = 1;
10                 ReplaceTStimuliReq(FM.Req(j).Name,CONFIG.Req(i).Name);

11                 ReplaceTOracleReq(FM.Req(j).Name,CONFIG.Req(i).Name);

12             end
13         else
14             //Has children requirement;
15             for k=1:size(FM.Req(j).Children) do
16                 if FM.Req(j).Children(k) == CONFIG.Req(i) then
17                     req_id(j) = 1;
18                     ReplaceTStimuliReq(FM.Req(j).Children(k).Name,CONFIG.Req(i).Name);

19                     ReplaceTOracleReq(FM.Req(j).Children(k)Name,CONFIG.Req(i).Name);

20                 end
21             end
22         end
23     end
24 end
25 for for i=1 to size(FM.Req) do
26     if requirements_id(i) == 0 then
27         RemoveTStimuliReq(FM.Req(j).Name);
28         RemoveTStimuliReq(FM.Req(j).Name);
29     end
30 end
```

**Algorithm 2:** Algorithm for the automatic configuration of the test system for the selected configuration requirements

```
    Data: GenericTestSystem, FeatureModel, ConfigurationFile
    Result: TestSystemInstance
 1  FM = parse(FeatureModel.xml);
 2  CONFIG = parse(c_1.config);
 3  for i=1 to size(FM.StimuliSignals) do
 4  |   RemoveStimuliSignal = true;
 5  |   for j = 1 to size(CONFIG.StimuliSignals) do
 6  |   |   if FM.StimuliSignals(i).Name == CONFIG.StimuliSignals(j).Name
    |   |   then
 7  |   |   |   //The stimuli signal is in the selected config;
 8  |   |   |   RemoveStimuliSignal = false;
 9  |   |   end
10  |   end
11  |   if RemoveStimuliSignal == true then
12  |   |   RemoveStimuliSignal(FM.StimuliSignals(i).Name;
13  |   end
14  end
```

**Algorithm 3:** Algorithm for the automatic configuration of the test system for the selected stimuli signals

```
    Data: GenericTestSystem, FeatureModel, ConfigurationFile
    Result: TestSystemInstance
 1  FM = parse(FeatureModel);
 2  CONFIG = parse(ConfigurationFile);
 3  //Deletes buses for new number of inputs;
 4  deleteBusCreator(VerdictBus);
 5  deleteBusCreator(CoverageBus);
 6  deleteUnconnectedLines();
 7  //Adds bus creators with new number of inputs;
 8  addBusCreator(VerdictBus, size(CONFIG.Requirements));
 9  addBusCreator(CoverageBus, size(CONFIG.Requirements));
10  //Integration of bus with arbiter;
11  addLine(VerdictBus,Arbiter);
12  addLine(CoverageBus, Arbiter);
13  //Integrate Requirement monitors with buses;
14  for i=1 to size(CONFIG.Requirements) do
15  |   addLine(CONFIG.Requirements(i).Name,VerdictBus);
16  |   addLine(CONFIG.Requirements(i).Name,CoverageBus);
17  end
```

**Algorithm 4:** Algorithm for the automatic configuration of the test verdict and coverage bus from the test oracle for a configuration

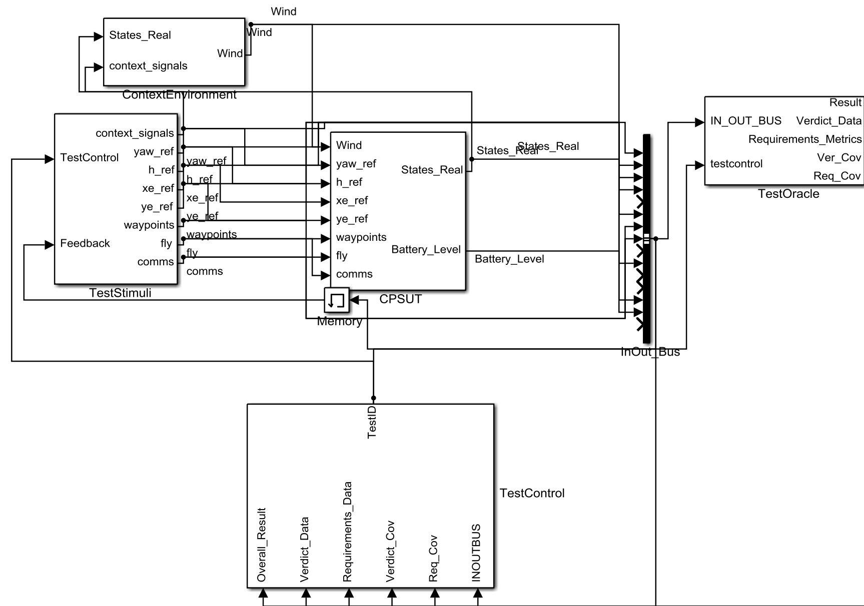



Figure 5.15: Configured Simulink Model for the configuration $c_i$ of the proposed illustrative example

the test stimuli have been removed. The same happens with the "Obstacle" and "Person_Data" from the context environment and the "FlyingLed" from the CPSUT.

### 5.4.5 Limitations

We have identified some limitations related to the current version of ASTERYSCO:

- Test System: We have used an extension of a test system that was designed for the validation of embedded systems from the automotive domain at system level. The developed tool is specific for the test system explained in Section 5.3.
- Simulation tool: Our prototype uses MATLAB/Simulink as the simulation tool. Other tools can be used for simulation, however, these tools must support scripts to generate the test system model. In addition, CPSs involve different technologies, and often, co-simulation is needed. We have not performed experiments using two independent simulation tools.
- Variability management tool: Our prototype used feature models employing the tool FeatureIDE [TKB+14]. Feature model and the tool FeatureIDE may have some limitations (e.g., FeatureIDE does not support cardinality type variability) that we have not considered. Other tools, such as SimPL [BYBS13], Clafer [BDA+15] or ZenCC [LYAZ16b] might be more adequate for variability modeling of CPSs.

Although we did not perform experiments with other tools, we believe that the only changes to be done would be the ones related to the parser.

## 5.5 Evaluation

This section evaluates the proposed approach for generating test system instances for configurable CPSs. We provide a case study and measure the time needed by ASTERYSCO to generate the test system instances for ten configurations. Later, we compare the difference of generating test systems employing our tool against to a manual process. Finally, the obtained results are discussed and some identified threats to validity of the performed evaluation are highlighted.

### 5.5.1 Case Study

The case study we employed for the evaluation of our approach is the one presented in Section 4.4.1. Before executing the case study, the test cases as well as the different requirement monitors for testing each functional requirement were developed. We selected a time-triggered execution strategy, i.e., a time slot was defined for each test case, and the test control measured the test execution time of each test case before executing the following one.

Regarding the test execution (i.e., the test stimuli), each requirement was tested by one or more reactive test cases. As for the test evaluation (i.e., the test oracle), each requirement was monitored by three requirement monitors. The first requirement monitor observed the tasks related to the high level control (i.e., if the position of the UAV was the correct one for the specified requirement). The second requirement monitor observed the tasks related to the low level control (i.e., if the stabilization of the UAV was correct). The third requirement monitor observed the specified property for the requirement. Figure 5.16 depicts the three requirement monitors for $r_1$. The test cases as well as the requirements monitor were stored in the Simulink library.

Once the test cases and the requirement monitors were developed the generic test system was generated. In a second step, we selected the configurations to test. These configurations were selected manually and automatically. We manually selected 10 different configurations ranging from c1 to c10. The simplest system variant is c1, which has minimum functionalities, whereas c10 is the most complex configuration with all the functionalities. We selected configurations of different complexities for two main reasons. The first reason is that users usually select products based on their price and based on their features. Products with more features tend to be more expensive, and as a result, many users cannot afford them. On the contrary, cheap

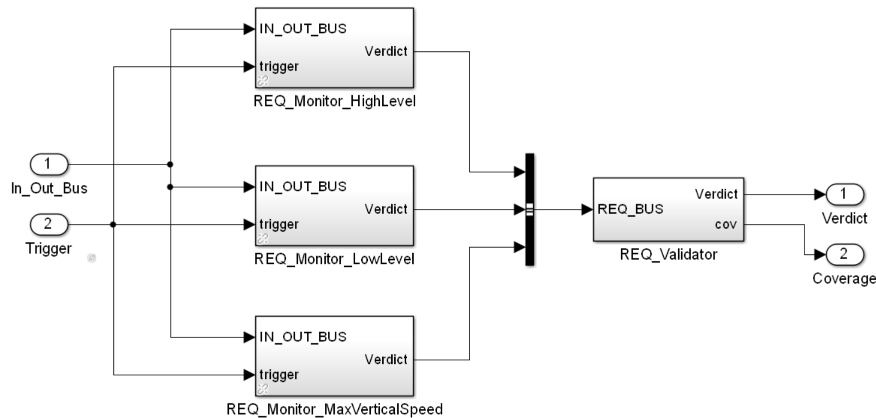



Figure 5.16: Detail of the three requirement monitors for the validation of requirement $r_1$ and its requirement validator

products tend to have few features, and while some users might consider the features of a cheap configuration to be enough, others could need a more sophisticated system. The second reason is related to the impact of the time required by ASTERYSCO to generate test systems based on the complexity of the configurations. We wanted to measure if our tool requires more time for configuring test systems with more requirements. We also selected configurations to test automatically. We employed FeatureIDE, which integrates different search algorithms for t-wise configuration selection. We employed the algorithm ICPL [JHF12] as it is the fastest configuration selection algorithm. We configured the algorithm to select the system variants to test following a pairwise criteria. The reason for selecting this criteria is that at least the interaction of two features is ensured, and thus, it can be ensured that the system does not fail due to the interaction of two features. In total, the ICPL algorithm selects 28 system variants. The name of the configurations that were automatically selected start with "pw."

Table 5.1 shows the number of features, requirements and stimuli signals of each selected configuration. Later, we generated the test system for each selected configuration with our approach and measured the time. We repeated this process 100 times to account for random variations when generating the test system. It can be appreciated that as the number of requirements that a specific configuration has increases, the time required to generate the test system tends to increase.

### 5.5.2 Results

Results of the case study are reported in this Section. Table 5.1 shows the time required to generate the test system instance for each configuration. Each configuration has a different number of requirements. The generation of each test system was repeated 100 times to account for random variations, as recommended by [AB11]. The test systems were generated with a computer running a Windows 7 operating system in an Intel Core i5, 2.5 GHz processor with 8 Gb of RAM memory. The column showing the generation time refers to the mean time required by our tool for generating each test system for the 100 repetitions. The standard deviation of the 100 repetitions is shown in the sixth column. The mean time for generating the generic test system was of 3.85 seconds, with a standard deviation of 0.11 seconds. The fastest test system configuration was for the pw15, which had 11 requirements and seven stimuli signals. The mean time for configuring this test system was of 2.84 seconds. On the contrary, c10, which had 20 requirements, was the configuration for which ASTERYSCO employed more time for the test system configuration, specifically, 3.53 seconds. Once the test systems were generated, the tests were executed. In total, about five hours were needed to test the 38 configurations.

### 5.5.3 Comparison with Manual Test System Generation

The steps that have to be taken to generate a test system as well as the differences between employing a manual test system generation or employing ASTERYSCO are shown in Table 5.2. First, a feature model as well as the test infrastructure must be elaborated. Second, configurations must be selected. Third, the generic test system is generated.[3] Lastly, the test system for each configuration is generated.

#### Step 1: Elaboration of Feature Model and test infrastructure

The first step corresponds to the elaboration of the feature model as well as the test infrastructure. Regarding the feature model, in the case of manual configuration, only the feature model of the system would be needed to be developed for managing variability of the system. The elaboration of the feature model also allows traceability with the CPS model, which warrants a systematic configuration. In addition, in the manual process, the test feature model would not be needed. Regarding the test infrastructure, just the test assets would not be developed or generated. The main *benefit* of employing a manual approach is that only the feature model related to the system would be developed, not having to develop the test feature model nor the test

[3] This step is not mandatory for the manual configuration.

Table 5.1: Time required by ASTERYSCO to generate the generic test system as well as different configurations

| | **Features** | **Reqs** | **Stimuli Signals** | **Generation Time (sec)** | **Standard Deviation** |
|---|---|---|---|---|---|
| GenericTS | | | | 3.85 | 0.11 |
| c1 | 17 | 10 | 7 | 2.87 | 0.1 |
| c2 | 20 | 11 | 8 | 2.93 | 0.07 |
| c3 | 21 | 12 | 8 | 3.04 | 0.06 |
| c4 | 22 | 13 | 9 | 3.11 | 0.06 |
| c5 | 23 | 14 | 10 | 3.2 | 0.07 |
| c6 | 24 | 15 | 10 | 3.26 | 0.07 |
| c7 | 25 | 17 | 10 | 3.4 | 0.07 |
| c8 | 28 | 18 | 10 | 3.49 | 0.06 |
| c9 | 29 | 19 | 10 | 3.5 | 0.06 |
| c10 | 31 | 20 | 10 | 3.53 | 0.14 |
| pw1 | 16 | 11 | 3 | 3.23 | 0.26 |
| pw2 | 26 | 13 | 8 | 3.27 | 0.32 |
| pw3 | 26 | 13 | 5 | 3.00 | 0.06 |
| pw4 | 26 | 16 | 8 | 3.18 | 0.05 |
| pw5 | 22 | 15 | 8 | 3.12 | 0.05 |
| pw6 | 27 | 15 | 6 | 3.19 | 0.22 |
| pw7 | 30 | 17 | 8 | 3.29 | 0.07 |
| pw8 | 27 | 16 | 10 | 3.18 | 0.06 |
| pw9 | 24 | 14 | 5 | 3.06 | 0.05 |
| pw10 | 28 | 14 | 8 | 3.05 | 0.05 |
| pw11 | 26 | 12 | 4 | 2.92 | 0.00 |
| pw12 | 28 | 15 | 5 | 3.16 | 0.00 |
| pw13 | 23 | 12 | 4 | 2.92 | 0.00 |
| pw14 | 26 | 14 | 5 | 3.07 | 0.00 |
| pw15 | 22 | 11 | 7 | 2.84 | 0.00 |
| pw16 | 26 | 15 | 5 | 3.14 | 0.00 |
| pw17 | 25 | 13 | 7 | 2.98 | 0.00 |
| pw18 | 23 | 13 | 3 | 3.01 | 0.00 |
| pw19 | 22 | 12 | 3 | 2.92 | 0.00 |
| pw20 | 21 | 13 | 3 | 3.02 | 0.00 |
| pw21 | 19 | 12 | 3 | 2.94 | 0.00 |
| pw22 | 24 | 13 | 3 | 3.16 | 0.00 |
| pw23 | 24 | 13 | 3 | 3.00 | 0.00 |
| pw24 | 24 | 13 | 3 | 3.01 | 0.00 |
| pw25 | 24 | 13 | 3 | 3.00 | 0.00 |
| pw26 | 24 | 13 | 3 | 3.01 | 0.00 |
| pw27 | 24 | 13 | 3 | 3.10 | 0.00 |
| pw28 | 24 | 13 | 3 | 3.13 | 0.00 |

Table 5.2: Comparison of ASTERYSCO with manual generation of test systems

| Steps | Manual | ASTERYSCO | Difference |
|---|---|---|---|
| **Step 1** | Elaborate Feature Model | Elaborate Feature Model and Test Feature Model | Employing our method, the feature model related to the test system as well as the reusable test assets must be elaborated |
| **Step 2** | Select relevant configurations | Select relevant configurations | The selected configurations with our method include the features related to the test system, whereas when not employing our approach just the features related to the system appear |
| **Step 3** | | Generate generic test system | Employing the manual approach it is not necessary to generate a generic test system |
| **Step 4** | Generate test system for each configuration | Generate test system for each configuration | ASTERYSCO generates automatically test system instances whereas with a manual methodology “clone and own” strategy would be needed,. This results in a time consuming, error-prone and non-systematic process. |

assets. On the contrary, the *limitation* would be that any information related to the variability of the system is available, and not information related to the test.

In the case of ASTERYSCO, the feature model of both the system as well as the test system has to be elaborated. In addition, in our proposed methodology, the dependencies between the features of the CPS and the features related to the test system must be traced with cross-tree constraints. As for the test infrastructure, different reusable test assets would be needed to develop. The *benefit* in this case is that information related to the variability of the test system is available in a feature model. However, the *limitations* are that the the feature model related to the test system must be manually elaborated. In addition, the reusable test assets must be generated. These tasks result in an extra investment of time.

### Step 2: Selection of Relevant Configurations

The second step corresponds to the selection of relevant configurations. We previously explained that this step can be performed either automatically (i.e., employing CIT algorithms) or manually. Configurations are usually generated automatically when feature pairwise coverage needs to be achieved. However, some customers might want specific configurations and thus, the features of the configurable CPS would be manually selected. The tool we employ for elaborating the feature model (i.e., FeatureIDE) supports both options and generates a *.config file for each configuration.

When not using the approach presented in this chapter, the features related to a configuration appear in a *.config file. The *benefit* in this case is that only information related to the CPSUT appears, and when configuring the CPSUT model there is no

need for dividing CPSUT information and information related to the test system. However, this implies an important *limitation*: as system requirements were not modeled in the feature model, system requirements do not appear in the *.config file, and as a result, test engineers have to select them manually. This is a problem in the following steps when generating test systems for specific configurations as test engineers have to constantly check the requirements documents and the selected features. Apart from not being a systematic process, the error proneness when generating the test system is increased (as some requirements might be omitted or incorrectly selected). Moreover, the process of manually selecting the requirements of each configuration when there are many configurations to test (which is the case of configurable CPSs) can be infeasible.

When employing ASTERYSCO, the features related to the test system are automatically selected based on the cross-tree constraints and they appear in the *.config file. One of the *benefits* in this case is that the requirements as well as other features (such as the stimuli signals or the context functions) are specified in a *.config file. This allows for the automatic configuration of the test system by our tool. A possible *limitation* might be that if just information about the CPSUT model is needed, then information related to the test system would need to be removed. The time for selecting relevant configurations (i.e., $T_CONF_SEL$) is the same for both methods.

#### Step 3: Generic Test System Generation

The third step corresponds to the generation of the generic test system. In the case of the manual approach for generating the test system, this step can be omitted. The major *benefit* might be that there is no need to spend time developing a generic test system. Nevertheless, a possible *limitation* for the manual approach is that when configuring a test system for a specific system variant, a generic test system could facilitate this process.

As explained in Section 5.4, in the case of ASTERYSCO, a generic test system is mandatory so that in the configuration step the test system can be configured efficiently. This generic test system is automatically generated. The *benefit* of using our tool for generating the generic test system is that it is fully automated and it can be generated in a few seconds (as shown in Table 5.2). One possible *limitation* when using of the method we propose is that an error free test feature model has to be developed. If the the feature model contains an error (e.g., the name of a stimuli signal is incorrect) a manual refinement process might be needed. In addition, there might be other major errors; for example, a requirement might not be well traced with a specific feature. This might imply that when a configuration holds this specific feature the requirement

is not tested. $T_{genericTS}$ is the time required by ASTERYSCO to generate a generic test system.

### Step 4: Configuration of the Test System

In the last step, the test system for each configuration is generated. This process is named configuration of the test system. In the case of manual elaboration of the test systems for each configuration, different strategies can be employed. The most typical would the one related to "clone and own." In this case, a test system as well as the test infrastructure (i.e., test cases and requirement monitors) would be generated for a specific configuration and later reused to generate other test systems for other system variants. The different elements of the test system would be needed to be manually generated (i.e., the stimuli signals, the test oracle, the context environment and the test control); this task can be time consuming as the selected test system can be complex for the test engineer. Moreover, all elements of the test system need to be manually integrated. In addition, there might be many configurations to test (i.e., a manual generation of the test system has to be performed for each system variant). A possible *benefit* of using a manual approach could be that if there are few configurations to test with systems with few requirements, a manual approach might be faster than developing the feature model in Step 1 and the reusable test infrastructure. However, this is not the case of configurable CPSs, which can be configured into many configurations. One *benefit* of manually configuring the test system is that it is not mandatory to have a *.config file with all the requirements. Nevertheless, the practice of manually configuring a test system for each configuration to test has important *limitations*. One of these *limitations* refers to the time required to manually develop a test system. Another *limitations* refers to the time required to generate the test cases and the requirement monitors for each configuration. Another *limitation* is that test engineers are exposed to fatigue when developing many test systems, which could lead to the generation of errors in the test system; this might lead to introduce errors into the test system. Given that there are N relevant configurations to test, $\sum_{i=1}^{N} MAN_T_{instTS_i}$ is the time required by a manual approach to generate the N test system instances. $\sum_{i=1}^{N} MAN_Infrastrut_{instTS_i}$ is the time required by a manual approach to generate the test infrastructure for the N configurations.

In our case, ASTERYSCO obtains the generic test system, and the test system for the specific configuration is configured in about 3 seconds (as illustrated in Table 5.2), reusing the previously generated test assets. This process is fully automated, which warrants the systematic generation of test system instances and reduces the error-proneness as well as the test system generation time. The *benefit* of this approach

is related to the configuration time, which is quite low (Table 5.2). Not only that, employing the tool proposed in this chapter, a systematic configuration of the test system is ensured while reducing the propensity of errors. A possible *limitation* could be that if the test feature model contained an error, this error could be propagated when generating a test system instance (e.g., a requirement of a specific configuration could be removed from the test system, or a requirement that a specific configuration does not have could be tested). Given that there are N relevant configurations to test, $\sum_{i=1}^{N} AST_T_{instTS_i}$ is the time required by ASTERYSCO to generate them.

#### Overall test system generation time

Based on the above-mentioned steps for the generation of test system instances employing ASTERYSCO or a manual approach, we separated the time required by each approach to generate N test system instances. The time required to manually generate N test system instances for testing relevant configurations is given in Equation 5.1. Equation 5.2 gives the time required to automatically generate the same configurations employing our method:

$$T_MAN = T_FM_{sys} + T_CONF_SEL + \sum_{i=1}^{N} (MAN_T_{instTS_i} + MAN_Infrastrut_{instTS_i}) \quad (5.1)$$

where, $T_FM_{sys}$ is the time required to develop the feature model, $T_CONF_SEL$ is the time for selecting the relevant configurations to test, $MAN_T_{instTS_i}$ is the time for manually generating a test system instance for configuration $i$, and $MAN_$ - $Infrastrut_{instTS_i}$ is the time required by a manual approach to generate the test infrastructure for configuration $i$.

$$T_AST = T_FM_{sys} + T_ASSETS + T_FM_{test} + T_CONF_SEL + T_{genericTS} + \sum_{i=1}^{N} AST_T_{instTS_i} \quad (5.2)$$

where, $T_FM_{sys}$ is the time required to develop the feature model, $T_ASSETS$ is the time required to develop the test assets, $T_FM_{test}$ is the time required to develop the feature model related to the test system, $T_CONF_SEL$ is the time for selecting the relevant configurations to test, $T_{genericTS}$ is the time required to generate the generic test system by ASTERYSCO, and $AST_T_{instTS_i}$ is the time required by ASTERYSCO to generate the test system instance for a configuration $i$.

### 5.5.4 Discussion

This section discusses the obtained results (Section 5.5.2) and the comparison between employing ASTERYSCO for generating test system instances with manual generation (Section 5.5.3).

ASTERYSCO is a tool that automatically generates test systems for specific system variants. To perform this task, the tool first generates a generic test system executing a model to model transformation. The proposed tool transforms a feature model into a generic test system for the tool Simulink. For the case study, the generation time for the generic test system was around 3.85 seconds with a standard deviation of 0.11 seconds. We consider the time required to generate the generic test system good, considering that the chosen case study has 20 requirements and 10 stimuli signals.

The generic test system can be reused across the configurations that the system can be set to. In order to do this, the test system configurator parses the configuration and allocates all the components needed to test the system variant in the test system. The performed experiments show that our tool automates the process of generating the test systems for specific system variants in about 3.5 seconds. On average, 119 seconds were needed to generate the 38 configurations. The test execution time for the 38 configurations took about five hours. Thus, we consider that 119 seconds is a quite small part of the total amount of time required to test the 38 configurations. It is important to highlight that the time required by the method we propose when configuring a test system increases as the number of requirements to be tested increases (Table 5.1). This might be caused due to the fact that the functions "Replace" (Algorithm 2) take more time to execute than the functions "Remove." However, the obtained results for configuring a specific test system indicate that our approach considerably reduces the time for generating test system instances. This means that the proposed method can substantially reduce verification and validation stage costs when testing configurable CPSs.

Section 5.5.3 compares ASTERYSCO with a manual process for the generation of test system instances. Four main steps must be given: (1) feature model elaboration and test infrastructure generation, (2) configuration selection, (3) generation of the generic test system and (4) configuration of the test system. For each step we compare the differences of the two approaches and highlight the main limitations and benefits. In general, the main limitation of of the method we propose as compared to a manual approach is that an amount of time must be invested building the test feature model and generating the reusable test assets. In addition, the test feature model has to be correctly built, otherwise, some errors can be propagated in the test system when generating it using ASTERYSCO. The main limitation of the tool we propose is

the main benefit of the manual process for building test system instances. When employing a manual approach for building the test system, it is not mandatory to build a test feature model. However, this can convert the manual generation of the test system into a real challenge, especially when the number of configurations to test is large. The test engineer must check the requirements of each configuration to be tested as well as other features related to the test system before building the test system. This issue involves a human factor, which might result in an increase of the error proneness. Moreover, a manual test system generation process requires a manual test case generation process for each configuration. This could be infeasible because of the time required for this task. The use of the methodology we propose ensures 100% requirements coverage if all the test cases for each configuration are executed. If the test cases are manually generated, the coverage can achieve 100% requirements coverage. Nevertheless, when the number of configurations to test is too large, the manual test case generation process would be very time consuming and the chances for achieving a high coverage would be reduced.

***Concluding Remark:*** The main limitation of the method we propose is that an investment of time is required when generating the test feature model in addition to the reusable test assets. Moreover, this test system has to be correctly performed and some testing is required to ensure the correctness of the feature model, otherwise, some errors can be introduced when generating the test system. However, the investment of time when developing the test feature model results in a systematic process when generating the test system. The tool allows full automation of the generation of test system instances for the configurations to test. When N (i.e., the number of configurations to test) is very low, a manual process could be more effective than employing ASTERYSCO. However, this is not usually the case of configurable CPSs. The number of configurations to test is large and thus, a benefit of our approach would be the time to generate test system instances. Equation 5.3 can be employed to guide practitioners when to use the proposed methodology. Note that we demonstrated that $T_{genericTS}$ and $AST_T_{instTS_i}$ were in the order of some seconds, and thus might be disregarded. The time required to build the test feature model (i.e., $T_FM_{test}$) and the reusable test assets (i.e., $T_ASSETS$) might require a lot of effort, but the time for manually generating each test system instance (i.e., $MAN_T_{instTS_i}$) and the test infrastructure (i.e., $MAN_Infrastrut_{instTS_i}$) can also take some hours. Moreover, it is important to note that as the number of configurations to test (i.e., N) increases, considerably more time is required when generating test systems manually. As the number of configurations to test in configurable CPSs is often very large, we recommend using ASTERYSCO for the purpose of generating test system instances.

$$T_FM_{test} + T_ASSETS + T_{genericTS} + \sum_{i=1}^{N} AST_T_{instTS_i} \leq \sum_{i=1}^{N}(MAN_T_{instTS_i} + MAN_Infrastrut_{instTS_i}) \quad (5.3)$$

### 5.5.5 Threats to Validity

This section summarizes the main threats that can invalidate our evaluation:
**External validity**: An *external validity* threat that usually affects most studies is the number of case studies employed. We employed one case study with ten configurations, which might not be enough to generalize our results. However, to reduce this threat, we employed a CPSUT with twenty functional requirements and ten configurations of different complexities. Another *external validity* threat of our study is related to the employed test system. This was an extension for configurable CPSs of that used in [ZN08] for embedded systems. Other kinds of systems might require some adaptation in the test system, and our tool might require more or less time with other test systems.
**Conclusion validity**: A *conclusion validity* threat involves the randomized time for generating test systems. This means that the time required by ASTERYSCO for generating test systems might suffer some variations. To reduce this threat, the generation of the generic test system as well as the configuration of each test system was repeated 100 times. Later, we calculated the mean time required by ASTERYSCO to generate test system instances as well as the standard deviation.

## 5.6 Related Work

Configuring configurable CPSs can often be very challenging. Some works have considered different modeling methodologies combined with search engines and constraint solving methods for the efficient configurations of these systems [NYA+13, BYBS13, BDA+15, LYAZ16b, LYAZ16a]. The objective of our study is not the configuration of CPSs however, but the generation of test systems for configurations of configurable CPSs. As the test system was modeled in Simulink, and the variability of Simulink models can be managed with feature models, we have chosen them for this task. We believe that our approach can be combined with the above-mentioned works that further study the configuration of CPSs by adapting the parsers.

Regarding test systems for the automated validation, Model-in-the-Loop for Embedded System Test (MiLEST) is a toolbox for MATLAB/Simulink developed in

[ZN08]. Some elements of our test system architecture, including part of the test stimuli and part of the test oracle have been extended from MiLEST. However, MiLEST is oriented towards the validation of individual embedded systems, and variability was not the objective of the tool. Our test system is designed for configurable CPSs and it is automatically generated by ASTERYSCO to test system variants of a configurable CPS. To achieve this goal, the test system supports variability in several points, as explained in Section 5.4. In addition, the elements of our test system are automatically generated taking into account the test feature model developed in the management stage using the tool FeatureIDE. Other test systems were also designed for CPSs. For instance, Kane et al. focused on requirement monitors that act as a test oracle [KFK14]. We also employ requirement monitors for deciding whether the results of the tests are valid or not. However, their approach was focused on automotive CPSs employing Hardware-in-the-Loop (HiL) simulations, whereas our approach is designed for Model-in-the-Loop (MiL) and Software-in-the-Loop (SiL) simulations. In addition, Kane et al. just employed test oracles [KFK14], whereas our test system also focuses on other sources (i.e., test stimuli, test oracle, context environment and test control).

Currently, many industrial projects are moving towards continuous integration and deployment and consequently require an automated solution to test all relevant variants. There are various approaches that used variability in the test system to test different system variants, especially in the Software Product Line (SPL) engineering domain. Unified Modeling Language (UML) and UML Testing Profile (UTP) was used as modeling languages and the variability of the test system was modeled using a UTP extension in [PPP09, PPG09]. Others used state machines as modeling languages, and for variability modeling, a delta oriented modeling strategy was chosen [LLSG12, DSLL13]. Perez et al. did not mention the test strategy followed [PPP09, PPG09]. Lity et al. used regression testing [LLSG12], whereas Dukaczewski et al. employed incremental testing [DSLL13]. There are several differences between these works and ours. Their SUT target are SPLs, whereas our test system is oriented towards configurable CPSs or CPS product lines. Their modeling language is UML and UTP or state machines. Although we use state flow (similar to state machines) for some requirement monitors, the modeling language we use is Simulink. With regard to the test strategy, we employed requirements based testing, as the validation is done at system level.

## 5.7 Conclusion and Future Work

The number of system variants that have to be tested to ensure that a configurable CPS will work properly across most of its configurations is large. Manually building a test system for each configuration to be tested is a non-systematic and time consuming task. ASTERYSCO is a tool that automates this process. For our prototypical version, we have extended MiLEST [ZN08]. In addition, a variability management tool is needed; for this task, we have used FeatureIDE [TKB+14]. The methodology we propose allows: (1) a systematic generation of test system instances for configurations of configurable CPSs and (2) a reduction in the time for generating these test system instances. An experiment with a case study has shown that employing the tool we proposed, a test system instance can be generated within three seconds. In addition, thanks to the developed tool, we were able to test all the selected configurations in about five hours without any need of human interaction.

As the variability in CPSs is increasing, we believe that this tool may help reduce the validation costs in several domains. For instance, demanding new user requirements or legislation changes lead to multiple development paths in CPSs from some domains [MSR14]. The variability can appear in the form of configurability or modifiability [TH02]. Configurability refers to the variability in product space, whereas modifiability to the variability in time space. Our approach is more focused on configurable CPSs. Nevertheless, evolution is also supported, although the process would need to begin from the beginning, i.e., modifications would be required in the variability management tool and the generic test system would be generated again.

In the future, we plan to extend ASTERYSCO to other test systems, as well as other simulation tools. For this task, it will be necessary to properly analyze its variability to adapt the two main elements of our tool, i.e., test system generator, and test system configurator. Regarding the simulation tool, we have used MATLAB/Simulink, due to its easiness to integrate mathematical models that represent the physical layer with software and control algorithms. Although MATLAB/Simulink is a powerful tool, it can have some restrictions (e.g., it is a proprietary software). In the future, we plan to use other simulation tools such as Modelica, LabView, etc. We also want to focus on a plan to transfer the tool to our industrial partners, adapting it to their needs. In fact, by the time this dissertation was submitted, the methodology proposed in this chapter was being evaluated by Hitachi engineers for the automotive domain.

Other future plans include the distributed simulation of various test systems. We believe that simulating several system variants in parallel, using different computers can reduce the overall validation time. Currently we are developing a higher level

test system that executes in parallel the tests of different system variant instances. Each system variant uses the test system generated by ASTERYSCO, (i.e., each of them has its own test control, test oracle, test stimuli and context environment). We have performed some experiments using the tool "Building Controls Virtual Test Bed" [WH08].

Chapter 6

# Automatic Test Case Generation

Testing Cyber-Physical Systems (CPSs) faces critical challenges and simulation-based testing is one of the most commonly used techniques for testing these complex systems. However, simulation models of CPSs are usually very complex and executing the simulations becomes computationally expensive, which often makes infeasible to execute all the test cases. To address these challenges, this chapter proposes a multi-objective test generation and prioritization approach for testing CPSs by defining a fitness function with four objectives and designing different crossover and mutation operators. We empirically evaluated our fitness function and designed operators along with five multi-objective search algorithms using four case studies. The evaluation results demonstrated that Non-dominated Sorting Genetic Algorithm II (NSGA-II) achieved significantly better performance than the other algorithms and managed to improve Random Search for on average 43.80% for each objective and 49.25% for the quality indicator Hypervolume (HV).

## 6.1 Introduction

CPSs have been cataloged as untestable systems and traditional testing techniques (e.g., model-based testing) are usually expensive, time consuming or infeasible to apply [BNSB16]. This is due to the fact that it is challenging for the traditional testing techniques to capture complex continuous dynamics and interactions between the system and its environment (e.g., people walking around when automatic braking systems are in use for automotive systems) [BNSB16, MNBB16]. As a result, simulation-based testing has been envisioned as an efficient means to test CPSs in a systematic and automated manner [BNSB16].

Apart from test case generation, test case prioritization in CPSs is highly important to detect faults as fast as possible, or, when employing reactive test cases, to reduce the test execution time. In our case, prioritization is tackled in two manners. The main one corresponds to purely prioritizing test cases, which is further explained in Chapter

8. However, this prioritization relies on a historical database, which needs a startup of a set of test executions in order to be effective. To overcome this problem, the test case generation approach presented in this Chapter can deal with test case prioritization. Specifically, we include a new objective based on a similarity measure, which aims at prioritizing dissimilar test cases.

To address the above-mentioned challenges, we proposed a search-based approach to cost-effectively generate and prioritize reactive test cases for testing industrial CPSs. First, we defined a fitness function with four cost-effectiveness measures (i.e., objectives) including requirements coverage, test case similarity, prioritization-aware test case similarity and test execution time. Second, we designed one crossover operator and two mutation operators (including mutation operator at test suite level and mutation operator at test case level). The defined fitness functions and designed crossover and mutation operators were incorporated with five multi-objective search algorithms and evaluated using four industrial case studies from different domains. Note that Random Search (RS) is also used as a baseline algorithm for the evaluation. The evaluation results showed that all the selected algorithms significantly outperformed RS in terms of addressing our test case generation and prioritization problem. Among all the algorithms NSGA-II achieved significantly better performance than the other algorithms and managed to improve RS for on average 43.80% for each objective and 49.25% for the quality indicator HV [WAY+16].

The rest of the chapter is structured as follows: The test generation and prioritization approach is presented in Section 6.2, and the tool support in Section 6.3. The approach is empirically evaluated in Section 6.4. The approach presented in this chapter is positioned with the current state-of-the-art in Section 6.5. Last, Section 6.6 concludes the chapter.

## 6.2 Multi-Objective Reactive Test Case Generation

In this section, the proposed search-based multi-objective approach for generating optimal reactive test cases cost-effectively is presented.

### 6.2.1 Cost-Effectiveness Measures

Four cost-effectiveness measures have been defined to guide the search towards generating optimal reactive test cases, i.e., requirements coverage, test case similarity, prioritization-aware test case similarity (effectiveness) and test execution time (cost).

### Requirements coverage

Reactive test cases are typically employed for functional testing at system level, meaning that functional requirements must be taken into account when generating test cases. For this reason, the *Requirements Coverage (RC)* measure is defined as the first effectiveness measure, which considers the number of requirements covered by a specific test suite with respect to the total number of requirements of a CPS. The *RC* is calculated following the Equation 6.1, where $|FR \lessdot sk|$ refers to the number of functional requirements covered by a solution while $|FR|$ denotes the total number of functional requirements in the CPS. Notice that *RC* is developed for each system with a script function by linking each requirement with a specific property of state or sequence of states of a reactive test case. For instance, if the acceleration functionality of the cruise control was tested, a state would set the car to 0 km/h whereas the proceeding state would set the speed to more than 150 km/h.

$$RC_{sk} = \frac{|FR \lessdot sk|}{|FR|} \tag{6.1}$$

### Test Case Similarity

Existing literature has shown that a set of diverse test cases has a higher chance to detect faults [FPCY16]. For this reason, the *Test Case Similarity (TCS)* was defined as the second effectiveness measure based on the *Hamming Distance (HD)*, which is a widely used similarity measure [HAB13]. More specifically, *TCS* measures the distance between two test cases and returns a value within the range [0,1]. A *TCS* value with 0 denotes that both test cases are identical, whereas when the *TCS* returns a value of 1, both of the test cases are considered as totally different. In addition to that, it can be followed in the context of reactive test cases that a reactive test case with more states is more likely to find errors than a test case with less states. With this concern in mind, the number of states is taken into account when calculating *TCS*.

The *TCS* of two test cases (i.e., $TC_a$ and $TC_b$) is calculated as expressed in Equation 6.2, where $|S_a|$ and $|S_b|$ are the number of states of each of the test cases and $|ss|$ is the number of stimuli signals for the CPS. $ss_{j_{tc_{ai}}}$ is the $j$-th signal's value for the $a$ test case's $i$-th state and $ss_{j_{tc_{bi}}}$ is the $j$-th signal's value for the $b$ test case's $i$-th state. In addition, $max_{ss_j}$ is the maximum value and $min_{ss_j}$ is the minimum value that the $j$-th signal can have. It is considered that these maximum and minimum values will always be different, and thus, the interval between $max_{ss_j}$ and $min_{ss_j}$ will not be 0. Finally, *MaxStates* refers to the maximum number of states a test case can have, which is predefined by the test engineer.

$$TCS(TC_a, TC_b) = \frac{\sum_{i=1}^{min(|S_a|,|S_b|)} \sum_{j=1}^{|ss|} \frac{abs(ss_{j_{tc_{ai}}} - ss_{j_{tc_{bi}}})}{abs(max_{ss_j} - min_{ss_j})} / |ss|}{MaxStates} \quad (6.2)$$

Consider as an example the *TCS* between the *TC1* and *TC2* from the sample Figure 6.1. Let us assume that the system has two stimuli signals (i.e., v and brake), the maximum speed that the system can be set to is 180 km/h, and the maximum number of states is 3. Thus, the *TCS* between *TC1* and *TC2* would be calculated as follows:

$TCS(TC1, TC2) = ((abs(0 - 150)/180 + abs(0 - 0)/1)/2$
$+ (abs(100 - 0)/180 + abs(0 - 0)/1)/2)/3 = 0.2315$

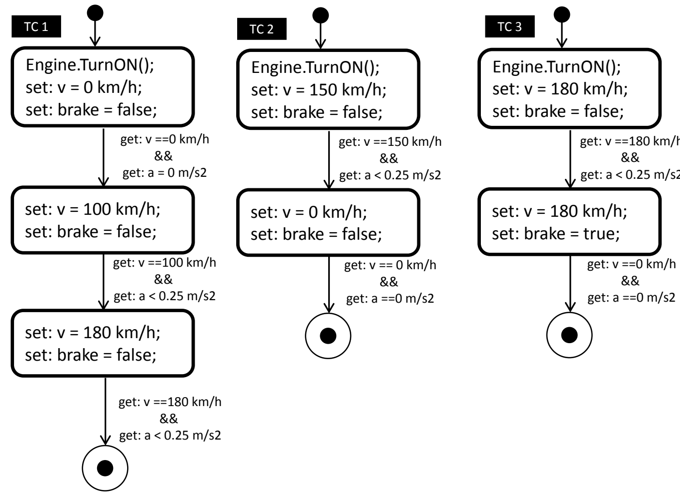



Figure 6.1: Example of three reactive test cases for the cruise control system testing of a vehicle

Furthermore, given a solution $sk$ with *NTC* number of test cases, the average similarity function can be measured by calculating the average *TCS* values for each test case pair. This is obtained following Equation 6.3, where *NTC* is the number of test cases in $sk$, $i$ is the $i$-th test case, $j$ is the $j$-th test case and $NTC \times (NTC - 1)/2$ is the total number of test case combinations in $sk$.

$$Sim_{sk} = 1 - \frac{\sum_{i=1}^{NTC} \sum_{j=i+1}^{NTC} TCS(TC_i, TC_j)}{NTC \times (NTC - 1)/2} \quad (6.3)$$

### Prioritization-aware Test Case Similarity

In other works, historical data is employed to prioritize test cases (e.g., [AWSE16b]). However, in this study, test cases are being prioritized while they are being generated,

what means that historical data is not available. For this reason, since diversifying the execution of test cases might imply having a higher chance of detecting faults, a new objective named Prioritization-aware Similarity (PSim) was included. This objective measures the average similarity of each of the test cases with its preceding test cases (i.e., test cases that were prioritized before). This permits guiding the search towards a prioritized test suite where the test cases in the initial positions of the suite are more different with one another. Given a prioritized test suite $PTS = \{TC_1, TC_2, ..., TC_{NTC}\}$, Equation 6.4 shows how the prioritization-aware similarity (PSim) objective is measured, where $NTC$ is the number of test cases in the test suite and $TCS(TC_i, TC_j)$ is the test case similarity between two test cases in position $i$ and in position $j$.

$$PSim_{sk} = 1 - \frac{\sum_{i=2}^{NTC} \frac{\sum_{j=1}^{i-1} TCS(TC_i, TC_j)}{i-1}}{NTC - 1} \tag{6.4}$$

### Test Execution Time

The time for executing a test suite is essential in the CPS testing context [AWSE16b, AWSE16a]. This is, to a large extent, caused by the high computational resources that simulation solvers consume to compute complex mathematical models related to the physical layer. Thus, the *Test Execution Time (TET)* is defined as a cost measure for the generation of test cases for the CPS context. Given a solution $sk$ of *NTC* test cases, each test case $i$ has $NS_{tc_i}$ number of states, the *TET* of $sk$ is calculated as proposed in Equation 6.5.

$$TET_{sk} = \sum_{i=1}^{NTC} (time(S_{NS_{TC_{i-1}}}, S_{1_i}) + \sum_{j=2}^{NS_{tc_i}} time(S_{j_i}, S_{j-1_i})) \tag{6.5}$$

It is important to highlight that the first part in Equation 6.5 refers to the initialization time of the test case. Since, apart from test generation, test prioritization is also performed, the time required to initialize the test case is taken into account (i.e., time between the last state of the previously executed test cases (i.e., $S_{NS_{TC_{i-1}}}$) and the first state of the test case (i.e., $S_{1_i}$)). This means that the test execution time of reactive test cases depends on the test prioritization.

Notice that the function time, which sets the time required by the system to change from one state to another, is system specific. This means that before launching the test generation approach, the test engineer must specify how the function time is computed. For instance, in the case of the cruise control example (Section 4.4.2), where the test cases are depicted in Figure 2.4, the time function between two states

would be $time(s1, s2) = a_c \times \Delta V$. Where, $\Delta V$ is the difference between both state speeds and $a_c$ is the acceleration coefficient, which changes if the car is accelerating, decelerating or braking.

### 6.2.2 Solution Representation

A solution in this context is a test suite ($TS$) composed by at least one Test Case ($TC$), i.e., $TS = \{TC1, TC2, ..., TC_N\}$, where $N$ is the total number of TCs in $TS$. Accordingly, a TC in our context is a reactive test case. Each of these test cases are composed by a set of states ($S$), (i.e., $TC_i = \{S_1, S_2, ..., S_{N_{tc_i}}\}$, where $N_{tc_i}$ is the number of states that the $i$-th $TC$ is composed of). Each state $S$ has a predefined set of stimuli signals ($ss$) that must be connected to the simulation model of the CPS. These stimuli signals are based on the simulation model, and can be of different types (i.e., Boolean, integer or double), and each of them has a maximum and a minimum value. Typically, CPSs test suites are composed of around 100 reactive test cases [AWSE16b].

Figure 6.2 depicts an instance for representing a particular solution, which denotes a test suite with *N* reactive test cases. As shown in Figure 6.2, each test case has a set of states and each state includes three stimuli signals, i.e., (1) *Eng* shows the current status of the engine (*on* or *off*), which can be represented as a Boolean input; (2) *V* refers to the set speed, which can be represented as an integer input; and (3) Brake means the brake pedal state, which can be represented as a Boolean input. In the example, the first test case (i.e., *TC1*) is composed by three states, while the third test case (i.e., *TC3*) by five. Notice that each of the states is composed by the three stimuli signals (Eng, V and Brake), which are directly connected to the inputs of the system with their specific values.

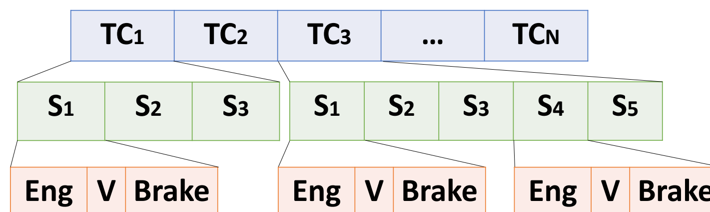



Figure 6.2: Representation of a Test Suite for *N* test cases with 3 stimuli signals

### 6.2.3 Crossover Operator

The implemented crossover operator exchanges the test cases between two different solutions. Given two test suites, a randomly generated crossover point is selected, in the range [1,$N$], where $N$ is the number of test cases related to the smallest test suite. When the crossover point is selected, two children are generated combining test cases between both test suites. A single point crossover was used but the algorithm can also be configured to use multi-point crossover.

Consider as an example two parent test suites as shown in Figure 6.3. One of them contains 6 test cases, whereas the other 10. Thus, the crossover function will randomly select a crossover point in the range [1,6]. In the illustrated example, the crossover point is the number 5. The child 1 maintains the first 5 test cases of parent 1, while it inherits the sixth to tenth test cases of parent 2. On the other hand, child 2 maintains the first 5 test cases of parent 2, while it inherits the sixth test case of parent 1. However, notice that one of the test cases corresponding to the test suite of Parent 1 and Parent 2 is the same (i.e., $TCe$). Child 1 inherits this test case from Parent 2 and thus, the test case is repeated in the test suite. When this happens, the test case in a later position is removed.

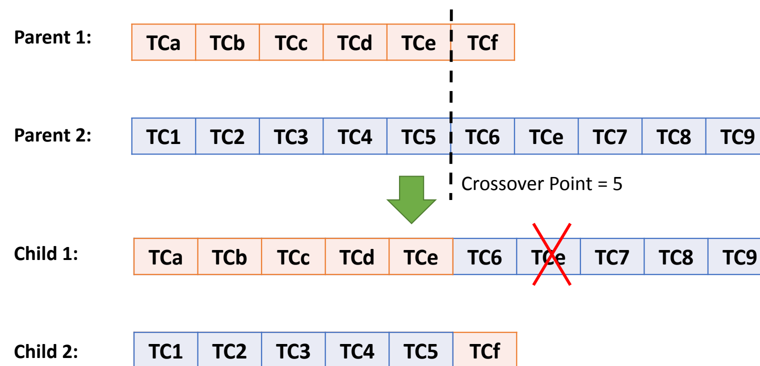



Figure 6.3: Crossover operator example for two test suites of 6 and 10 test cases, having the crossover point at the fifth place

### 6.2.4 Mutation Operators

The mutation operators are designed from two levels. The first level is the test suite level, whereas the second one is the test case level. The mutation operator at the test suite level randomly mutates test cases by adding, removing or exchanging them from the test suite. The mutation operator at the test case level mutates the states by modifying them.

### Mutation operator at test suite level

For the mutation operator at test suite level, three sub-mutation operators have been developed. When the mutation operator at test suite level is selected, one of these sub-mutation operators is randomly chosen by the algorithm. The first sub-mutation operator consists of the addition of a new test case into the test suite. Notice that the new test case is randomly generated. When a new test case is added, this operator randomly decides its position in the test suite as well as the number of states that this test case must have. When the number of states is decided, it randomly selects the values of their stimuli signals based on the type and the maximum and minimum values they can be set to. The second sub-mutation operator consists of the removal of a test case from the test suite. It randomly selects the test case to be removed from the test suite and a child test suite is generated without the selected test case. A third sub-mutation operator has been developed at the test suite level to account for the test case prioritization, which consists of exchanging the position of two test cases. When this sub-mutation operator is selected, it randomly selects two test cases and exchanges their position. Figure 6.4 illustrates the sub-mutation operators for the test suite level. As for the first sub-mutation operator, a new test case is added in the third position of the test suite. As for the second sub-mutation operator, the fourth test case is removed from the test suite. Finally, the exchange sub-mutation operator selects the second and third test cases and their positions are swapped.

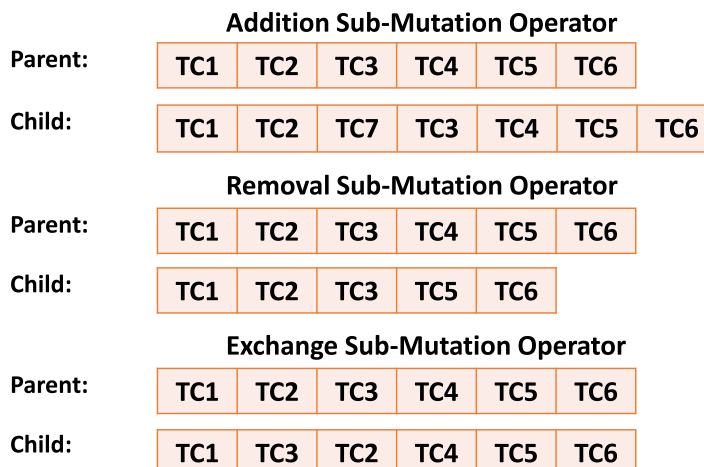



Figure 6.4: Sub-mutation operators at test suite level

### Mutation operator at test case level

The mutation operator at test case level operates within the states that certain test cases have. At this level, four sub-mutation operators have been developed, consisting of state addition, state removal, exchange of states and change of variable. When the mutation operator at test case level is selected, one of these sub-mutation operators is randomly chosen.

For the state addition operator, a test case from the test suite is randomly chosen and a new state is added in a random position of that test case. Consider as an example Figure 6.5, where a new state is added to the second position of the test case.

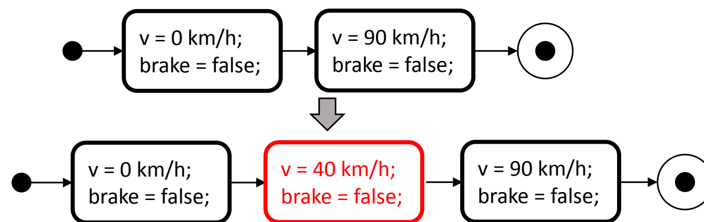



Figure 6.5: Addition sub-mutation operator for the test case level

For the state removal operator, a test case is randomly chosen from the test suite and one of its states is randomly removed. Consider as an example the test case depicted in Figure 6.6, where the second state is removed.

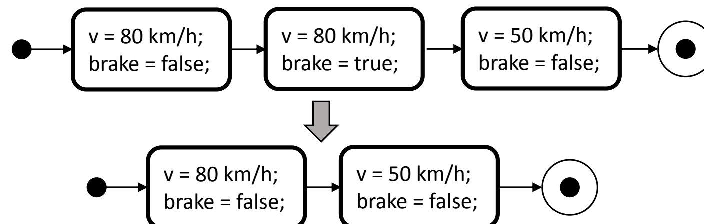



Figure 6.6: Removal sub-mutation operator for the test case level

For the state exchange operator, a test case is randomly chosen from the test suite. Later, two states are randomly selected and their positions are exchanged. Consider as an example Figure 6.7, where the second state is exchanged with the third one to form a new child test case.

Lastly, the change of variable operator randomly chooses one of the stimuli signals of a test case and its value is changed according to its type and maximum as well as minimum values. In the example provided in Figure 6.8, the brake stimuli signal of the second state is changed.

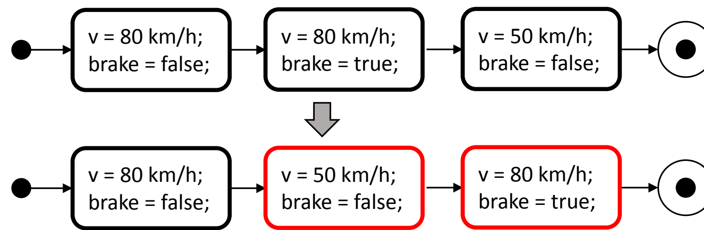



Figure 6.7: Exchange sub-mutation operator for the test case level

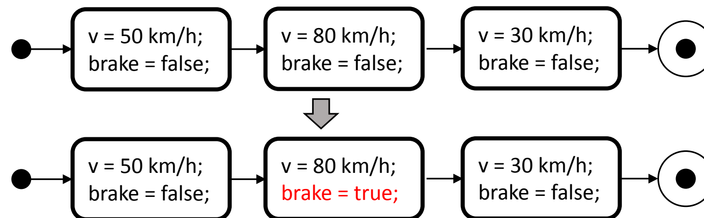



Figure 6.8: Change of variable sub-mutation operator for the test case level.

## 6.3 Tool Support

We have integrated the proposed test case generation approach with ASTERYSCO for the efficient validation of configurable CPSs. To this end, we have considered the tools that are used for the rest of the parts (i.e., test system generation, test selection and test prioritization). The developed tool employs feature models, specifically FeatureIDE [TKB+14], to manage the solution representation. Specifically, in the test feature model the stimuli signals are modeled (as proposed in Chapter 5). FeatureIDE permits embedding small description inside the features. By using these descriptions we were able to embed the type of data (i.e., if they were boolean, integer or double), as well as the maximum and minimum values of each of the stimuli signals. Later, our test generation algorithm was able to parse the feature model, which was saved into an *.xml format, to obtain the representation of the solution. Specifically, it obtains the total amount of stimuli signals, the type of each of them and their maximum and minimum values. With this data it is already possible to specify how a solution (i.e., a test suite) should be represented. Figure 6.9 shows an example of embedding a description for the Unmanned Aerial Vehicle (UAV) case study. Specifically, the "h_ref" stimuli signal is set as an integer with the minimum value of 0 and the maximum of 1500.

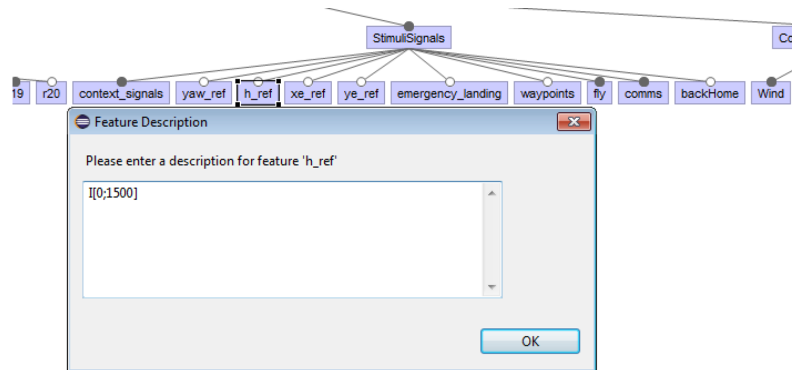



Figure 6.9: Example of the tool support of the test case generation approach for the UAV case study

## 6.4 Empirical Evaluation

This section reports an empirical evaluation for the presented test case generation approach using four different case studies.

### 6.4.1 Research Questions

To evaluate the proposed approach, two Research Questions (RQs) were aimed to answer, which are detailed as below:

- **RQ1:** Are the selected multi-objective algorithms cost-effective when compared to *RS* for solving the test case generation and prioritization problem?
- **RQ2:** Which of the selected multi-objective algorithms fares best when solving the test case generation and prioritization problem?

### 6.4.2 Experimental Setup

This section explains the experimental setup in detail.

#### Case Studies

The previously described (Chapter 4) four case studies were employed in the proposed empirical evaluation. Table 6.1 reports the key characteristics of the selected case studies. Specifically, the *Reqs* column specifies the number of requirements of the systems. The *Stimuli Signals* column is the number of stimuli signals of their Simulink models (B means the number of Boolean signals and I means the number of integer

signals). The last two columns are related to the Simulink models of the systems; the *Blocks* column is related to the number of blocks that the systems has, whereas the column *Depth* refers to the number of hierarchical levels of the system model.

Table 6.1: Key characteristics of the case studies

| Case Study | Reqs | Stimuli Signals | Blocks | Depth |
|---|---|---|---|---|
| **DC Engine** | 25 | 4 (2-B, 2-I) | 257 | 3 |
| **UAV** | 22 | 10 (6-B, 4-I) | 843 | 4 |
| **ACC** | 10 | 7(4-B, 3-I) | 415 | 5 |
| **Tank** | 9 | 3 (3-I) | 112 | 4 |

#### Algorithms Parameters Configurations

Apart from Random Search (RS), which was the algorithm employed as a baseline, five search algorithms were selected: Non-dominated Sorting Genetic Algorithm II (NSGA-II) [DPAM02], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [ZLT+01], Pareto Envelope-based Selection Algorithm II (PESA-II) [CJKO01], Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [ZL07] and Non-dominated Sorting Genetic Algorithm III (NSGA-III) [DJ14]. The designed crossover and mutation operators were integrated into the selected algorithms. As recommended by one of the most commonly applied multi-objective optimization Java framework jMetal [DN11], we set the crossover rate as 0.9, the population size was 100 and the number of fitness evaluations was 100,000. The mutation probability is 1/N, being N the number of test cases for the mutation operator at test suite level, being N the number of states for the mutation operator at test case level and three first mutation sub-operator and being N the number of stimuli signals for the mutation operator at test case level and the fourth mutation sub-operator. In addition, each algorithm was run 100 times to account for random variations as recommended by Arcuri and Briand [AB11].

#### Evaluation Metrics

Based on the guide [WAY+16], the *HV* quality indicator was selected as the evaluation metric for the empirical evaluation. To be specific, *HV* measures the volume in the objective space covered by the produced solutions [DN11] with the range from 0 to 1 and a higher value of *HV* denotes a better performance of the algorithm. It is important to note that *HV* has been applied to assess similar multi-objective test optimization approaches (e.g., test case generation approach [BANBS16]). In addition to the *HV* quality indicator, the four fitness values of each selected objectives were measured,

with the aim of measuring the ability of each algorithm in terms of optimizing each objective.

#### Statistical Analysis

As recommended by Arcuri and Briand, the Mann-Whitney U test (i.e., significance test) was used to determine the significance of the results produced by different algorithms. The significance level was set to 95%, whereby, there is a statistically significant difference between the results of two algorithms if the p-value is less than 0.05. To determine the difference existing between two algorithms the Vargha and Delaney statistics was used to calculate the $\hat{A}_{12}$ measure [AB11, VD00].

### 6.4.3 Results and Analysis

This section discusses the key results and observations. Tables 6.2 and 6.3 report the performed statistical analysis. For each evaluation metric and each pair of algorithm the $\hat{A}_{12}$ and the p-value is provided. Since the objective of the algorithms is to maximize the *HV* indicator and the Requirements Coverage metric, for these two metrics an $\hat{A}_{12}$ value between 0 and 0.5 means that the algorithm in the left performed better than the algorithm in the right. Conversely, a value between 0.5 and 1 means that the algorithm in the right performed better than the algorithm in the left. For the remaining metrics, since the objective of the algorithms is to minimize them, an $\hat{A}_{12}$ value between 0 and 0.5 means that the algorithm in the right performed better than the algorithm in the left. An $\hat{A}_{12}$ value between 0.5 and 1 means the opposite.

The first RQ aims at comparing the five selected algorithms with RS to assess that the problem to solve is not trivial. As shown in Figures 6.10, 6.11, 6.12, 6.13 and 6.14 and corroborated by means of statistical tests (summarized in Tables 6.2 and 6.3), all the algorithms significantly outperformed RS, although there were some exceptions. As for the *HV* quality indicator, for the Adaptive Cruise Control (ACC) case study, RS outperformed with statistical significance (according to the Mann-Whitney U-Test) SPEA2, PESA-II and NSGA-III. Furthermore, in the UAV case study RS also outperformed NSGA-III. For the ACC case study, regarding the remaining objectives, all the algorithms significantly outperformed RS with the exception of the Similarity and the Prioritization-aware similarity, where RS significantly outperformed SPEA2 and PESA-II, according to the Mann-Whitney U-Test. In addition, RS significantly outperformed MOEA/D in three out of four case studies for the TET objective. Nevertheless, NSGA-II significantly outperformed RS since all the p-values are less than 0.05 for each case study and each objective.

Table 6.2: Results for the statistical tests related to the Mann-Whitney U-test and the Vargha and Delaney $\hat{A}_{12}$ measure for RQ1

| | | **Independent Objectives** | | | | | | | | **Overall** | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Req. Coverage** | | **TET** | | **Similarity** | | **Similarity Prio.** | | **HV** | |
| | | $\hat{A}_{12}$ | **p-value** | $\hat{A}_{12}$ | **p-value** | $\hat{A}_{12}$ | **p-value** | $\hat{A}_{12}$ | **p-value** | $\hat{A}_{12}$ | **p-value** |
| DC Engine | RS vs NSGA-II | 1.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 1.00 | <0.0001 |
| | RS vs SPEA2 | 1.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 1.00 | <0.0001 |
| | RS vs MOEA/D | 0.78 | <0.0001 | 0.48 | 6.24E-01 | 0.00 | <0.0001 | 0.00 | <0.0001 | 1.00 | <0.0001 |
| | RS vs PESA-II | 0.90 | <0.0001 | 0.00 | <0.0001 | 0.02 | <0.0001 | 0.02 | <0.0001 | 0.80 | <0.0001 |
| | RS vs NSGA-III | 0.94 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.96 | <0.0001 |
| UAV | RS vs NSGA-II | 0.99 | <0.0001 | 0.16 | <0.0001 | 0.00 | <0.0001 | 0 | <0.0001 | 1.00 | <0.0001 |
| | RS vs SPEA2 | 0.79 | <0.0001 | 0.04 | <0.0001 | 0.00 | <0.0001 | 0 | <0.0001 | 0.95 | <0.0001 |
| | RS vs MOEA/D | 1.00 | <0.0001 | 0.89 | <0.0001 | 0.00 | <0.0001 | 0 | <0.0001 | 1.00 | <0.0001 |
| | RS vs PESA-II | 0.58 | 3.04E-02 | 0.02 | <0.0001 | 0.01 | <0.0001 | 0.01 | <0.0001 | 0.75 | <0.0001 |
| | RS vs NSGA-III | 0.08 | <0.0001 | 0.04 | <0.0001 | 0.00 | <0.0001 | 0 | <0.0001 | 0.15 | <0.0001 |
| ACC | RS vs NSGA-II | 1.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0 | <0.0001 | 1.00 | <0.0001 |
| | RS vs SPEA2 | 1.00 | <0.0001 | 0.01 | <0.0001 | 0.69 | <0.0001 | 0.66 | 1.34E-03 | 0.31 | <0.0001 |
| | RS vs MOEA/D | 0.99 | <0.0001 | 1.00 | <0.0001 | 0.00 | <0.0001 | 0 | <0.0001 | 1.00 | <0.0001 |
| | RS vs PESA-II | 1.00 | <0.0001 | 0.11 | <0.0001 | 0.88 | <0.0001 | 0.87 | <0.0001 | 0.01 | <0.0001 |
| | RS vs NSGA-III | 0.75 | <0.0001 | 0.06 | <0.0001 | 0.44 | 2.38E-03 | 0.44 | 2.01E-03 | 0.15 | <0.0001 |
| Tank | RS vs NSGA-II | 0.92 | <0.0001 | 0.01 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 1.00 | <0.0001 |
| | RS vs SPEA2 | 0.72 | <0.0001 | 0.01 | <0.0001 | 0.01 | <0.0001 | 0.01 | <0.0001 | 0.98 | <0.0001 |
| | RS vs MOEA/D | 0.67 | <0.0001 | 0.93 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.99 | <0.0001 |
| | RS vs PESA-II | 0.77 | <0.0001 | 0.05 | <0.0001 | 0.00 | <0.0001 | 0.01 | <0.0001 | 1.00 | <0.0001 |
| | RS vs NSGA-III | 0.73 | <0.0001 | 0.03 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.98 | <0.0001 |

The second RQ assessed which of the selected algorithms performed best. As shown in Figure 6.10, according to the *HV* quality indicator, NSGA-II outperformed the rest of the algorithms. Furthermore, NSGA-II significantly outperformed the other selected algorithms based on the results of Mann-Whitney U-Test (all the p-values are less than 0.05). Conversely, for the *HV* indicator, the algorithm showing worst performance was NSGA-III. Apart from the *HV* quality indicator, for the remaining objectives, in general, NSGA-II was also the best algorithm. However, there were some exceptions. For instance, for the DC engine and the ACC case studies, the SPEA2 algorithm outperformed with statistical significance the NSGA-II algorithm for the Requirements Coverage objective. In addition, for the test execution time, PESA-II and the NSGA-III also outperformed the NSGA-II algorithm in the UAV case study. Furthermore, except for the tank case study, for the similarity and the prioritization-aware similarity, the MOEA/D performed better that the NSGA-II. However, the MOEA/D algorithm was outperformed by the rest of algorithms for the TET objective.

Figure 6.15 depicts the distribution of the algorithms running times to generate the test suites. As it can be appreciated, on the one hand, the slowest algorithms were PESA-II and NSGA-III, which employed on average around 2200 and 1800 seconds respectively to generate the test suites. On the other hand, the fastest one was MOEA/D, which took around 600 seconds to generate test suites. The NSGA-II algorithm, which is the algorithm showing the overall best performance, took around

Table 6.3: Results for the statistical tests related to the Mann-Whitney U-test and the Vargha and Delaney $\hat{A}_{12}$ measure for RQ2

| | | Independent Objectives | | | | | | | | Overall | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Req. Coverage** | | **TET** | | **Similarity** | | **Similarity Prio** | | **HV** | |
| | | $\hat{A}_{12}$ | **p-value** | $\hat{A}_{12}$ | **p-value** | $\hat{A}_{12}$ | **p-value** | $\hat{A}_{12}$ | **p-value** | $\hat{A}_{12}$ | **p-value** |
| DC Eng | NSGA-II vs SPEA2 | 0.93 | <0.0001 | 0.71 | <0.0001 | 0.58 | 1.34E-01 | 0.60 | 4.27E-02 | 0.03 | <0.0001 |
| | NSGA-II vs MOEA/D | 0.06 | <0.0001 | 1.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.09 | <0.0001 |
| | NSGA-II vs PESA-II | 0.54 | 0.7978 | 0.61 | 2.52E-03 | 0.88 | <0.0001 | 0.88 | <0.0001 | 0.00 | <0.0001 |
| | NSGA-II vs NSGA-III | 0.47 | 0.4045 | 0.73 | <0.0001 | 0.13 | <0.0001 | 0.13 | <0.0001 | 0.12 | <0.0001 |
| | SPEA2 vs MOEA/D | 0.01 | <0.0001 | 1.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.67 | 2.14E-04 |
| | SPEA2 vs PESA-II | 0.20 | <0.0001 | 0.45 | 3.89E-01 | 0.84 | <0.0001 | 0.83 | <0.0001 | 0.13 | <0.0001 |
| | SPEA2 vs NSGA-III | 0.14 | <0.0001 | 0.57 | 2.31E-02 | 0.12 | <0.0001 | 0.12 | <0.0001 | 0.57 | 1.49E-01 |
| | MOEA/D vs PESA-II | 0.80 | <0.0001 | 0.00 | <0.0001 | 1.00 | <0.0001 | 1.00 | <0.0001 | 0.07 | <0.0001 |
| | MOEA/D vs NSGA-III | 0.83 | <0.0001 | 0.01 | <0.0001 | 0.92 | <0.0001 | 0.91 | <0.0001 | 0.44 | 1.36E-01 |
| | PESA-II vs NSGA-III | 0.46 | 0.5846 | 0.60 | 4.09E-02 | 0.06 | <0.0001 | 0.07 | <0.0001 | 0.84 | <0.0001 |
| UAV | NSGA-II vs SPEA2 | 0.13 | <0.0001 | 0.31 | <0.0001 | 0.62 | 1.94E-03 | 0.69 | <0.0001 | 0.01 | <0.0001 |
| | NSGA-II vs MOEA/D | 0.37 | 2.36E-03 | 0.99 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.34 | 1.01E-03 |
| | NSGA-II vs PESA-II | 0.09 | <0.0001 | 0.14 | <0.0001 | 0.81 | <0.0001 | 0.86 | <0.0001 | 0.00 | <0.0001 |
| | NSGA-II vs NSGA-III | 0.02 | <0.0001 | 0.12 | <0.0001 | 0.27 | <0.0001 | 0.34 | 4.33E-04 | 0.00 | <0.0001 |
| | SPEA2 vs MOEA/D | 0.78 | <0.0001 | 0.99 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.94 | <0.0001 |
| | SPEA2 vs PESA-II | 0.36 | 1.79E-03 | 0.25 | <0.0001 | 0.71 | <0.0001 | 0.74 | <0.0001 | 0.17 | <0.0001 |
| | SPEA2 vs NSGA-III | 0.07 | <0.0001 | 0.20 | <0.0001 | 0.21 | <0.0001 | 0.23 | <0.0001 | 0.04 | <0.0001 |
| | MOEA/D vs PESA-II | 0.16 | <0.0001 | 0.00 | <0.0001 | 1.00 | <0.0001 | 1.00 | <0.0001 | 0.01 | <0.0001 |
| | MOEA/D vs NSGA-III | 0.03 | <0.0001 | 0.01 | <0.0001 | 0.96 | <0.0001 | 0.96 | <0.0001 | 0.00 | <0.0001 |
| | PESA-II vs NSGA-III | 0.12 | <0.0001 | 0.39 | 3.10E-02 | 0.13 | <0.0001 | 0.13 | <0.0001 | 0.14 | <0.0001 |
| ACC | NSGA-II vs SPEA2 | 0.64 | 0.0017 | 0.42 | 1.18E-01 | 0.97 | <0.0001 | 0.97 | <0.0001 | 0.00 | <0.0001 |
| | NSGA-II vs MOEA/D | 0.31 | <0.0001 | 1.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.25 | <0.0001 |
| | NSGA-II vs PESA-II | 0.27 | <0.0001 | 0.39 | 5.38E-01 | 1.00 | <0.0001 | 0.99 | <0.0001 | 0.00 | <0.0001 |
| | NSGA-II vs NSGA-III | 0.12 | <0.0001 | 0.36 | 4.27E-02 | 0.67 | <0.0001 | 0.67 | 1.87E-04 | 0.01 | <0.0001 |
| | SPEA2 vs MOEA/D | 0.26 | <0.0001 | 1.00 | <0.0001 | 0.00 | <0.0001 | 0.00 | <0.0001 | 0.98 | <0.0001 |
| | SPEA2 vs PESA-II | 0.21 | <0.0001 | 0.47 | 5.51E-01 | 0.76 | <0.0001 | 0.77 | <0.0001 | 0.14 | <0.0001 |
| | SPEA2 vs NSGA-III | 0.10 | <0.0001 | 0.43 | 3.20E-01 | 0.36 | <0.0001 | 0.37 | <0.0001 | 0.25 | <0.0001 |
| | MOEA/D vs PESA-II | 0.47 | 0.3049 | 0.02 | <0.0001 | 1.00 | <0.0001 | 1.00 | <0.0001 | 0.00 | <0.0001 |
| | MOEA/D vs NSGA-III | 0.23 | <0.0001 | 0.00 | <0.0001 | 1.00 | <0.0001 | 1.00 | <0.0001 | 0.02 | <0.0001 |
| | PESA-II vs NSGA-III | 0.26 | <0.0001 | 0.46 | 2.92E-01 | 0.18 | <0.0001 | 0.19 | <0.0001 | 0.57 | 1.95E-02 |
| Tank | NSGA-II vs SPEA2 | 0.33 | <0.0001 | 0.61 | 2.64E-03 | 0.92 | <0.0001 | 0.91 | <0.0001 | 0.08 | <0.0001 |
| | NSGA-II vs MOEA/D | 0.32 | <0.0001 | 1.00 | <0.0001 | 0.64 | 1.65E-04 | 0.65 | 9.59E-05 | 0.14 | <0.0001 |
| | NSGA-II vs PESA-II | 0.30 | <0.0001 | 0.63 | 2.11E-03 | 0.97 | <0.0001 | 0.96 | <0.0001 | 0.05 | <0.0001 |
| | NSGA-II vs NSGA-III | 0.44 | 0.01 | 0.62 | 2.36E-03 | 0.74 | <0.0001 | 0.75 | <0.0001 | 0.13 | <0.0001 |
| | SPEA2 vs MOEA/D | 0.47 | 0.41 | 1.00 | <0.0001 | 0.18 | 2.71E-11 | 0.19 | <0.0001 | 0.56 | 7.35E-02 |
| | SPEA2 vs PESA-II | 0.50 | 0.95 | 0.50 | 9.32E-01 | 0.58 | 6.31E-02 | 0.57 | 1.15E-01 | 0.46 | 3.87E-01 |
| | SPEA2 vs NSGA-III | 0.55 | 0.426 | 0.52 | 4.67E-01 | 0.26 | <0.0001 | 0.27 | <0.0001 | 0.56 | 8.16E-02 |
| | MOEA/D vs PESA-II | 0.54 | 0.4536 | 0.01 | <0.0001 | 0.88 | <0.0001 | 0.87 | <0.0001 | 0.40 | 5.61E-03 |
| | MOEA/D vs NSGA-III | 0.53 | 0.3072 | 0.01 | <0.0001 | 0.59 | 1.93E-02 | 0.59 | 2.23E-02 | 0.50 | 9.03E-01 |
| | PESA-II vs NSGA-III | 0.59 | 0.4260 | 0.51 | 8.18E-01 | 0.19 | <0.0001 | 0.20 | <0.0001 | 0.60 | 9.97E-03 |

1100 seconds on average to generate the test suites.

As for the improvements, Table 6.4 shows the average percentage improved for the *HV* as well as each of the objectives by NSGA-II (which overall resulted in the best algorithm) with respect to RS, which was taken as the baseline algorithm. On average, the *HV* quality indicator was improved in 49.25%, the requirements coverage in 51.74%, the test execution time in 62.96%, the test case similarity in 29.86% and the prioritization-aware similarity in 30.65%.

### 6.4.4 Discussion of the Results

The first RQ aimed to answer whether the test generation and prioritization problem for the CPS context was not trivial to solve. To this end, the selected algorithms

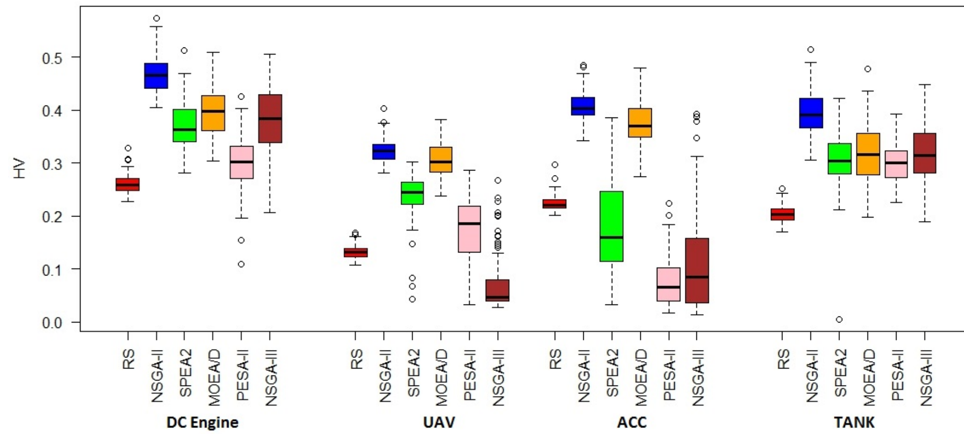



Figure 6.10: Distribution of the *HV* indicator for all the runs in each case study

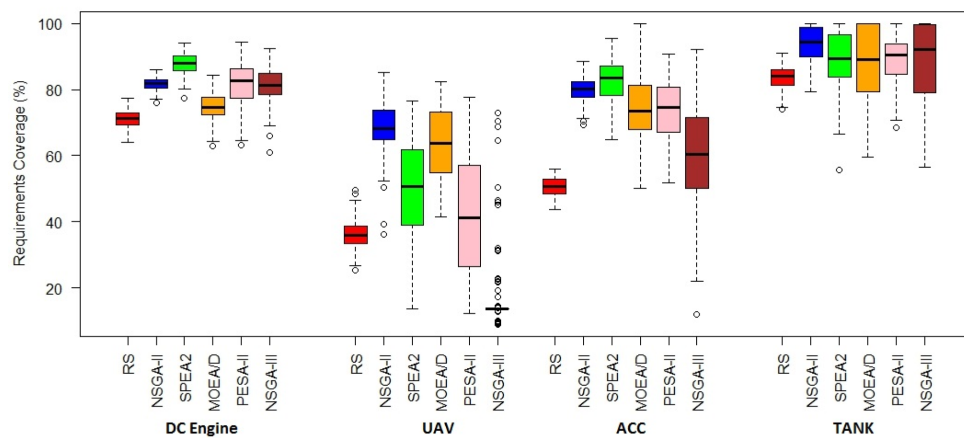



Figure 6.11: Distribution of the Requirements Coverage objective for all the runs in each case study

were compared with RS. Results indicated that in general the selected multi-objective search algorithms outperformed RS. Specifically, NSGA-II outperformed RS for all the four objectives and the *HV* in the four case studies with statistical significance. Thus, the first RQ can be answered as follows:

> *Based on the experimental results and the statistical tests of our study we can conclude that the test case generation and prioritization approach for the CPS context is a non-trivial problem and thus, search algorithms are recommended to be used.*

The second RQ aimed to identify which of the algorithms performed best when

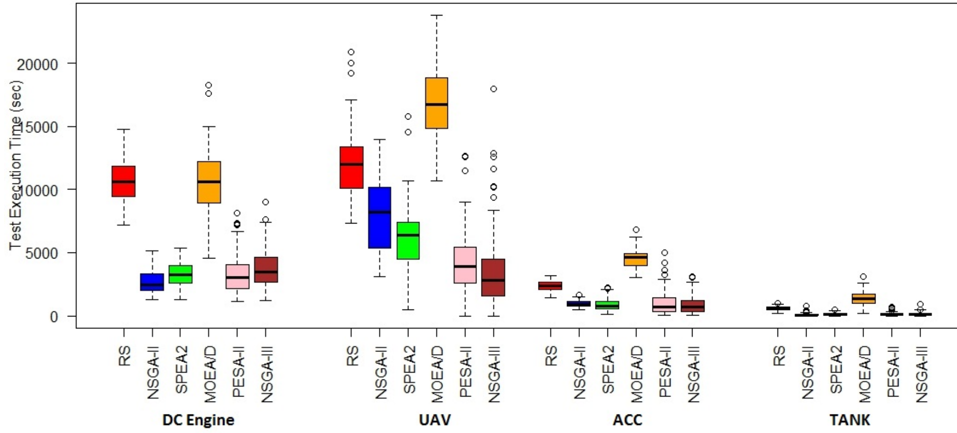



Figure 6.12: Distribution of the Test Execution Time objective for all the runs in each case study

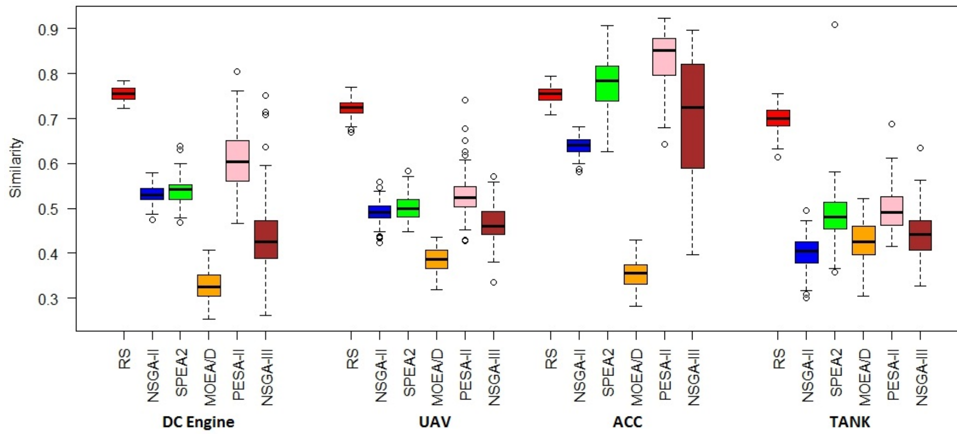



Figure 6.13: Distribution of the Similarity objective for all the runs in each case study

solving the test generation and prioritization problem. Overall, NSGA-II showed the best performance. However, MOEA/D showed the best performance for both defined similarity measures in three out of four case studies. This means that MOEA/D is good at generating dissimilar test cases. Nevertheless, for the rest of objectives (i.e., test execution time and requirements coverage) as well as for the *HV* the NSGA-II was the best algorithm. Thus, the second RQ can be answered as follows:

> *Based on the experimental results and the statistical tests, in general, NSGA-II performed best and we recommend it to be used. However, if the similarity of test cases is crucial, MOEA/D is recommended.*

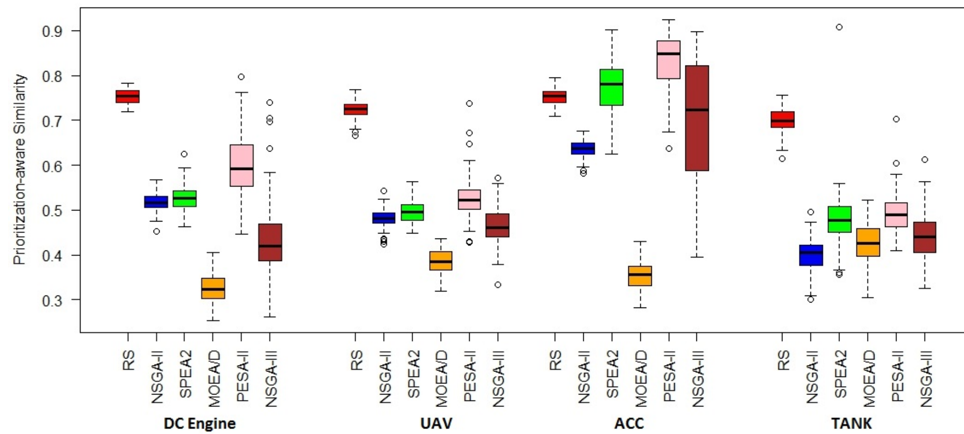



Figure 6.14: Distribution of the Prioritization-aware Similarity objective for all the runs in each case study

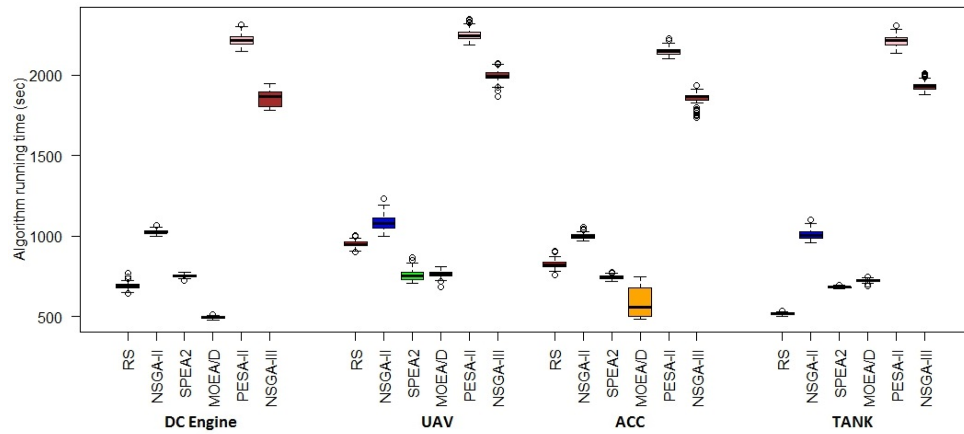



Figure 6.15: Distribution of the running time employed by each algorithm to generate the test suite

### 6.4.5 Threats to Validity

This section summarizes the identified threats that could invalidate the performed empirical evaluation:

**Internal validity**: One of the *internal validity* threats lies on the parameter configurations of search algorithms (e.g., population size, number of generation, crossover rate) since different parameter settings may lead to different performance of algorithms [AF13]. To reduce this threat, the settings suggested in the guidelines provided by Arcuri and Briand [AB11] as well as the default settings of jMetal [DN11] were selected.

**External validity**: An *external validity* threat could be related to the generalization

Table 6.4: Average percentage of improvement of NSGA-II compared with RS for each objective and each case study

| | HV | ReqCov | TET | Similarity | PrioSim |
|---|---|---|---|---|---|
| **DC Engine** | 44.37% | 36.76% | 75.02% | 29.54% | 31.13% |
| **UAV** | 59.28% | 50.04% | 32.92% | 32.23% | 33.45% |
| **ACC** | 44.95% | 59.47% | 60.17% | 15.05% | 15.32% |
| **Tank** | 48.42% | 60.71% | 83.74% | 42.60% | 42.69% |
| **Average** | 49.25% | 51.74% | 62.96% | 29.86% | 30.65% |

of the results. To deal with this issue four independent case studies from different application domains and with different complexities were used for evaluating our approach. Moreover, one of the case studies was a real-world industrial case study.
**Conclusion validity**: A *conclusion validity* threat could be the random variations produced by search algorithms. To reduce this threat, the executions of the algorithms were repeated 100 times and analyzed by means of statistical tests, as recommended in [AB11].
**Construct validity**: A *construct validity* threat might be that the measures used are not comparable across the selected algorithms. To reduce this threat the same stopping criterion for all the algorithms was used, i.e., the number of fitness evaluations is set to 100,000 to seek the best solutions for test generation.

## 6.5 Related Work

Search-based test generation has been widely applied in the software engineering community [McM04, ABHPW10]. The proposed test generation approach in this chapter generates whole test suites, which was proposed for unit testing [FA13]. The use of search-based algorithms have widely been used for the generation of test suites [AF14, FAM13b, FA11]. Several empirical evaluations have recently demonstrated the effectiveness of this technique for testing several systems, such as complex industrial applications [AHF+17] or non-trivial open source classes [CGF+17]. Typically, the approach of generating whole test suites aims at covering all structural coverage goals at the same time. Conversely, our algorithm focuses on generating test suites composed of reactive test cases following a multi-objective approach for maximizing requirements coverage and the test suite diversity while minimizing the test execution time. Moreover, our algorithm returns a prioritized test suite so that test cases are executed in a diversified way, which would allow for a faster fault detection.

In Section 3.2.2 we provide references for the most relevant works on test case generation for CPSs. Some of them focus on generating test cases employing either

Model-Based Testing (MBT) (e.g., [MM16, AHF+14, ARM16]). However, these approaches face scalability issues. Instead, several search-based approaches have proposed test generation by simulating the system in each iteration to find worst-case scenarios (e.g., [MNB+15, BANBS16, MNBB16, MNB17, VLW+13]). The main drawback of employing simulation models for finding worst case scenarios is that it might be extremely time consuming to generate test cases (e.g., in [BANBS16] a test generation budget of 120 minutes was used). Our approach does not use simulation to guide the search towards optimal solutions. Instead of trying to find worst case scenarios, we focus on system testing by trying to produce cost-effective reactive test cases taking into account requirements coverage, test execution time, test similarity and prioritization-aware similarity.

Our test case generation approach generates a test suite composed of reactive test cases. In the current state of the art some works focused on the generation of reactive test cases (e.g., [ZN08, Leh00, Mje13]). However, all these works, apart from being semi-automatic, they focused on testing requirements, but they did not take other properties into account, such as the test execution time or test similarity, something that is considered in this chapter.

In our case, the developed tool support allows for the execution of test cases in Simulink models. The generation of test cases for testing Simulink models has been widely applied (e.g., [BHM+10, BT+15, RSB+13, LTMHT14, ZC05, ZC08, LLNB17, YRW+15, HWRS08]). However, all these studies use a single objective function to generate test cases, while we employ multiple objectives. Moreover, a test case in their context is a set of signals that stimulate the inputs of the Simulink models, whereas in our case, the test cases are reactive test cases, which allows for the observation of the system to address the unpredictability of the physical environment.

At the same time of generating test cases, our approach also focuses on prioritization. The prioritization in our context allows for the test execution time reduction (in the context of reactive test cases the test execution time varies based on the prioritization [AWSE16b]) as well as executing diversified test cases. This is important when there is no historical information about the capacity of test cases to detect faults. In the current state of the art, test prioritization has been widely studied [CM13, HMZ12, YH12]. Search-based approaches have been widely investigated for prioritizing test cases [EYHB15, WSKR06, HFM15]. However, to the best of our knowledge, there are no studies that prioritize reactive test cases at the same time at generating them, as proposed in this chapter.

## 6.6 Conclusion and Future Work

We proposed a search-based multi-objective approach for systematically generating and prioritizing cost-effective reactive test cases for testing CPSs. To this end, we defined a fitness function with four objectives to guide the search towards finding optimal solutions. We also designed one crossover operator and two mutation operators. The fitness function and designed operators were integrated with five different search algorithms and evaluated using four case studies. The results showed that NSGA-II together with our fitness function and operators demonstrated the best performance and managed to outperform RS for on average 43.80% for each objective and 49.25% for the HV quality indicator.

As for the future work, we plan to include more industrial case studies to further strengthen our approach. We also plan to involve industrial practitioners for testing our approach in their current practice. Moreover, in the current work, the requirements coverage function is developed for each of the study subjects with a script function. In the future, we would like to investigate a formal approach to link reactive behaviors of the system with functional requirements to automatically generate these scripts. This would enrich our work and make the process of generating test cases more systematic.

# Part III

# Optimization

Chapter 7

# Test Case Selection

Cyber-Physical Systems (CPSs) are often tested at different test levels following "X-in-the-Loop" configurations: Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL). While MiL and SiL test levels aim at testing functional requirements at the system level, the HiL test level tests functional as well as non-functional requirements by performing a real-time simulation. Testing configurable CPSs configurations is costly due to the fact that there are many variants to test, test cases are long, the physical layer has to be simulated and co-simulation is often necessary. It is therefore extremely important to select the appropriate test cases that cover the objectives of each level in an allowable amount of time. We propose an efficient test case selection approach adapted to the "X-in-the-Loop" test levels. Search algorithms are employed to reduce the amount of time required to test configurations of configurable CPSs while achieving the test objectives of each level. We empirically evaluate three commonly-used search algorithms, i.e., Genetic Algorithm (GA), Alternating Variable Method (AVM) and Greedy (Random Search (RS) is used as a baseline) by employing two case studies with the aim of integrating the best algorithm into our approach. Results suggest that as compared with RS, our approach can reduce the costs of testing configurable CPSs configurations by approximately 80 % while improving the overall test quality.

## 7.1 Introduction

The high number of configurations that need to be tested during the test and validation stages of configurable CPSs leads to the need of optimizing as much as possible the complete test activities. One of the approaches proposed in this thesis to cost-effectively test configurable CPSs is to select relevant test cases for testing each configuration. It is worth mentioning that one of the key challenges in the context of CPS testing is that the three levels (i.e., MiL, SiL and HiL) have their own characteristics, which require defining corresponding objectives for the test case selection.

Reducing the cost while achieving high effectiveness (e.g., fault detection capability) is the ultimate goal for optimization testing of CPSs and thus, we take test execution time as a prime objective.

Software modeled at the MiL level uses floating point arithmetic, which means that simulations are performed using high precision. On the contrary, at the SiL level, the software uses fixed point arithmetic, which implies that simulations are performed with a fixed precision of the order of some bits. At these two levels, the hardware of the system is not taken into account and thus, cost-effectively testing functional requirements gains greater importance since improper functional behaviors might provoke important damages. The HiL test level is the most realistic simulation since the software of the system is integrated with the rest of the hardware as well as the real-time infrastructure (e.g., drivers, Real-Time Operating System (RTOS), networks). The integration of the software with the real hardware permits testing CPSs focusing on both functional and non-functional properties. Non-functional properties are critical in the context of CPSs. First, the time it takes to execute a task or its execution deadline might lead to a completely new behavior of the system, unlike in general purpose software, in which these properties are related to performance [DLSV11, LBB15]. Second, critical parts such as the network environment that aims to connect different digital units for communication needs to be thoroughly tested. In some cases, testing the network environment is quite challenging, unrealistic and time-consuming at the MiL or SiL level, and thus the HiL level is mandatory. For this reason, we take into account non-functional requirements coverage at the HiL level. Moreover, the capacity of detecting faults is always one of the core objectives for testing and thus, this objective is taken into account at the three test levels.

The main contribution of this chapter is highlighted in Figure 7.1. Based on the key characteristics of different test levels of CPSs (i.e., MiL, SiL and HiL), we define corresponding cost-effectiveness objectives (Figure 7.1) and propose a search-based software engineering approach. This cost-effectively selects the test cases for testing configurations of configurable CPSs at different test levels. However, the selection process can be challenging if there are many test cases that can be chosen. Given that there are N test cases that can be selected, there are $2^N$ possible test case selection combinations. This means that if there are 100 test cases in the test suite, there are $1.2677 * 10^{30}$ test selection possibilities. Exploring the whole search space to determine which is the best solution is impracticable, and thus, an efficient search process is needed.

To address the above-mentioned challenges, we first capture the variability of configurable CPSs with a feature model in FeatureIDE [TKB+14]. This feature model

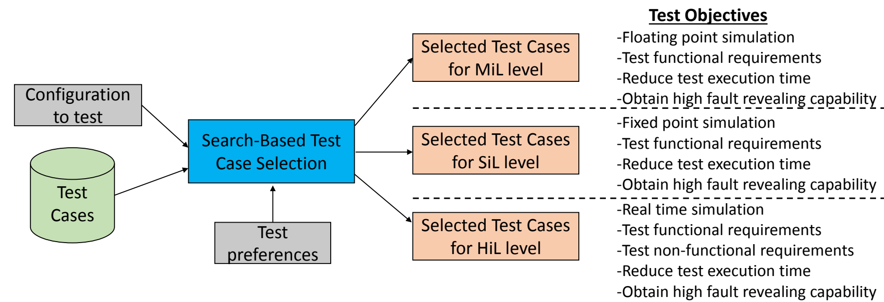



Figure 7.1: Overall overview of the main contribution

is also used to integrate the features of the system with functional and non-functional requirements, which are used as a basis for the objectives of the test process at different levels. Once a specific product configuration is selected, an automated search process is triggered to cost-effectively select the appropriate test cases for fulfilling the objectives of each test level. To enable the automated search process, we define five effectiveness measures and two cost measures followed by a fitness function to guide the search algorithms towards finding optimal solutions.

The rest of the chapter is structured as follows: Section 7.2 presents our approach to cost-effectively select test cases in the context of configurable CPSs. The approach is evaluated in Section 7.3, where an empirical evaluation with two case studies is performed. Section 7.4 positions our work with other approaches in the current literature. Finally, conclusions and future work are summarized in Section 7.5.

## 7.2 Search-Based Test Case Selection

This section presents the proposed approach for the selection of relevant test cases in the context of configurable CPSs. Figure 7.2 depicts the overall process employing a Software Process Engineering Meta-model (SPEM) diagram. Continuous lines indicate the process flow whereas the discontinuous lines indicate the object flow. The approach shows four different steps. The first step corresponds to the variability modeling part, where feature models are employed to manage the variability of configurable CPS and trace their features with functional and non-functional requirements. The output of this process is a test feature model, which is employed in the following process. Specifically, we used the tool FeatureIDE [TKB+14] for variability modeling, due to its robustness and availability. The second step corresponds to the configuration selection. FeatureIDE allows both manual configuration selection as well as automatic (employing different Combinatorial Interaction Testing (CIT) algorithms). The output of this step is a configuration file that employs a *.config extension. Once

the configurations are selected, we employ search-based algorithms to cost-effectively select test cases for testing each configuration. In this step, the algorithms process the feature model as well as the configuration files to identify which requirements have been selected for each configuration. Later, search algorithms are employed to select test cases at three test levels: MiL, SiL and HiL. The last step corresponds to the test case execution. In our case, we employed the tool Simulink (due to the fact that our systems were modeled in this tool). A test system similar to the one proposed in [ASE14b, ASE15a] is employed to manage the execution of the test cases.

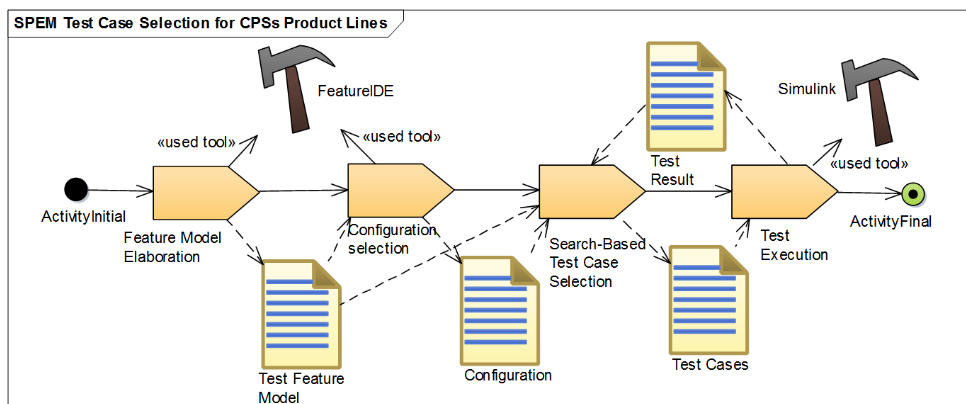



Figure 7.2: Approach overview for the test case selection process

### 7.2.1 Feature Modeling for Cyber-Physical Systems Validation

In the proposed test selection approach, in order to be consistent with the rest of the approaches proposed in this thesis, variability is managed using feature models, which are the most common notations used in industry to model variability [BRN+13]. In this case, functional and non-functional requirements of the system are modeled within the feature model. These functional and non-functional requirements can be optional or mandatory. Requirements and features of the system are traced with cross-tree constraints so that when a configuration is selected based on its features, its requirements are selected automatically. Figure 7.3 depicts an example of a test feature model for a configurable CPS. Figure 7.4 shows an example of a configuration selection. The functional and non-functional requirements are automatically selected based on the traceability with cross-tree constraints.

Note that FeatureIDE [TKB+14] was used in our case for building the feature model, which provides a user friendly interface to enable several options such as automatic generation of configurations and manual configuration. Although the use of feature models can have some drawbacks in the context of CPS product line

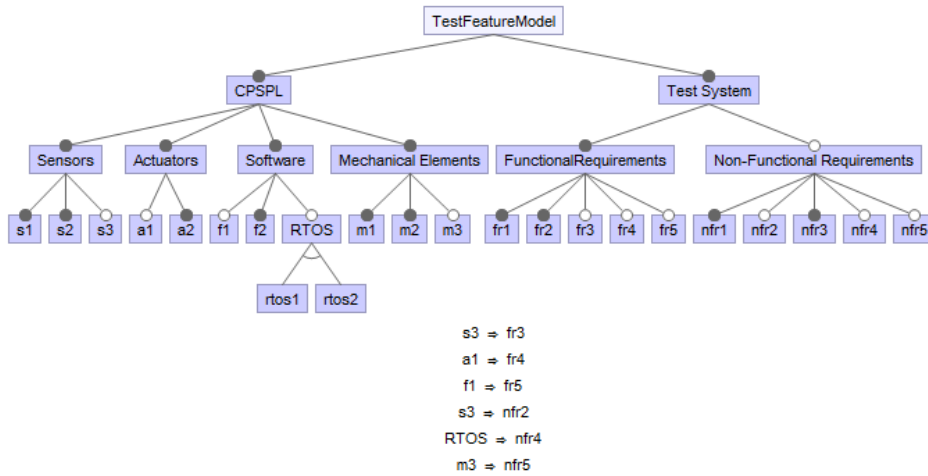



Figure 7.3: Test feature models for configurable CPSs

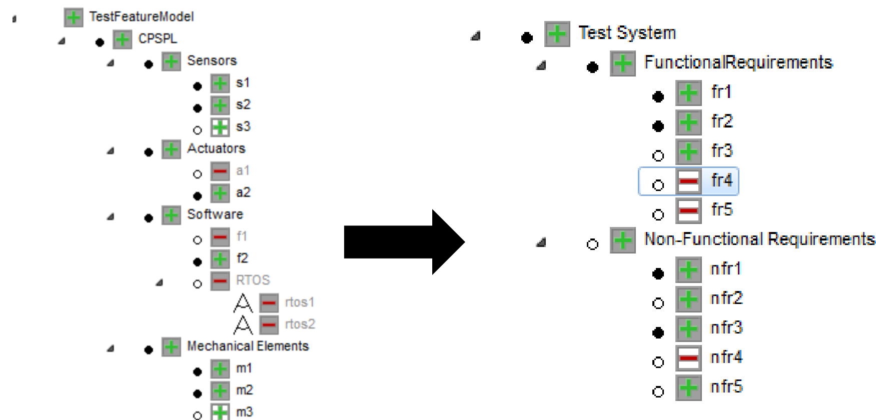



Figure 7.4: Example of a configuration selection

engineering, it satisfies our current needs. However, our approach could easily be integrated with other variability modeling techniques suitable for the context of CPS product line engineering (e.g., [LYAZ16b, SYAL16, BDA+15, Beh12, BYBS13]).

### 7.2.2 Search-Based Test Case Selection

Our search-based test case selection approach employs weight-based search algorithms, which assign a weight to each optimization objective and convert a multi-objective problem into a single-objective one [WAG15]. Given that the optimization problem has N number of Objectives ($Obj$ ), each $Obj$ must range between 0 and 1 (i.e.,

$0 \leq Obj_i \leq 1$) [WAG13]. A weight ($w$) is assigned to each $Obj$, and the sum of all the weights must result in 1 (i.e., $\sum_{i=1}^{N} w_i = 1$). Thus, the fitness function of an optimization problem of N objectives for a weight-based search algorithm is expressed in Equation 7.1.

$$\sum_{i=1}^{N} w_i * Obj_i \tag{7.1}$$

We chose weight-based search algorithms due to several reasons. The first is that they support user preferences, i.e., a user can give more importance to one objective or to another. This is very important for testing configurable CPSs, as there are many options that might be more interesting than others (e.g., detecting faults might be more important than achieving high coverage). The second reason is that based on the weight-based theory, any global or local algorithm can be easily applied [WAG13]. This can be helpful, for instance, when optimal solutions are located in a local search space. In this case, local search algorithms can be used. On the contrary, if optimal solutions are located in a global search space, global search algorithms can be employed. For this study, four search algorithms were used. As representative of local search algorithms, Greedy and the AVM algorithms were chosen. The GA was chosen for the representative of global search algorithms because: (1) the GA is the most commonly applied algorithm based on the existing literature [JH11, LHLE15]; and (2) the GA can achieve very good performance especially for solving various software testing problems based on previous related work [WAG13, WAG15]. RS was the algorithm taken as a baseline to assess the performance of the selected search algorithms.

### 7.2.3 Quality Measures

Quality measures, which are also referred as effectiveness measures throughout this chapter, determine how effective a specific test suite is. We selected five measures to determine the effectiveness of specific test suites for different test levels.

#### Fault Detection Capability

The Fault Detection Capability (FDC) of a specific test suite is a factor that measures how well a test suite detects faults [WAG13]. The FDC is obtained observing historical data from previously executed test cases. In the context of CPSs, it is often difficult to differentiate different faults: in some cases a fault might provoke many symptoms whereas, in other cases, two different faults might provoke similar symptoms. To overcome this problem, we calculate the FDC as proposed by Wang et al. [WAG13]

(Equation 7.2). In this case, a test case is successful if at least one fault is revealed. On the contrary, the test case has failed if it is executed and no fault is detected. $sk$ is a solution given by the search algorithm. In the context of this study, a solution is a test suite with a set of $nt_{sk}$ test cases. $N_{tc}$ is the number of test cases available in the test database for testing the configurable CPS, and $SR_{tc_i}$ is the successful rate for a test case $i$.

$$FDCsk = \frac{\sum_{i=1}^{nt_{sk}} SR_{tc_i}}{N_{tc}} \tag{7.2}$$

The successful rate ($SR$) of a test case $i$ (i.e., $tc_i$) is given by the number of times it has been successful ($NumSuc_{tc_i}$) with respect to the number of times it was executed, i.e., the number of times it was successful and the number of times it failed ($NumFail_{tc_i}$) [WAG13], as expressed in Equation 7.3.

$$SRtc_i = \frac{NumSuc_{tc_i}}{NumSuc_{tc_i} + NumFail_{tc_i}} \tag{7.3}$$

#### Functional Requirements Coverage

Functional Requirements Coverage (FRC) measures the percentage of requirements that have been covered by a specific solution $sk$. This measure is calculated following Equation 7.4, which is given by the number of functional requirements covered by a solution $sk$ (i.e., $|FR_{cps_i} \lessdot sk|$), with respect to the number of functional requirements coverage that a configuration $i$ has (i.e., $Nfr_{cps_i}$).

$$FRCsk = \frac{|FR_{cps_i} \lessdot sk|}{Nfr_{cps_i}} \tag{7.4}$$

#### Pairwise Functional Requirement Coverage

In the context of CPSs, there might be conflicting requirements. For instance, consider as an example the adaptive cruise control of a car where the first requirement (Req 1) is defined as follows: "*The car shall obtain the indicated speed by the user with a maximum error of 0.5 m/s*". The second requirement (Req 2) is defined as follows: "*The car shall progressively reduce the speed if a car is detected in front at less than 50 meters*". In this case, let's assume that the driver sets the speed of the car to 30 m/s. Thus, the speed of the car should range between 29.5 and 30.5 m/s. However, if a car is detected in front, the speed will be progressively reduced to meet Req 2 (which will have higher priority), but, in that case, Req 1 will not be met. To ensure that all these

cases are tested, we define the Pairwise Functional Requirements Coverage (pw_FRC) measure.

Empirical studies have demonstrated pairwise coverage has a good capability of detecting more than 50 % of the faults (between 50 % and 97 %) with an acceptable amount of test case execution time [KKLH09]. A higher strength coverage may lead to better solutions in terms of detected faults, but the execution time of the solution (selected test cases) could be exponentially increased. The pairwise requirements coverage is given in Equation 7.5, which is calculated with the number of functional requirement pairs covered by a solution $sk$ (i.e., $|PAIRFR_{cps_i} \lessdot sk|$) with respect to all the possible requirement pairs in a configuration $cps_i$ (i.e., $\sum_{j=1}^{Nfr_{cps_i}-1} j$). For instance, given that a configuration has four requirements, there are six possible requirement pairs. If a solution covers three requirement pairs, the pw_RC will be 0.5.

$$pw_FRCsk = \frac{|PAIRFR_{cps_i} \lessdot sk|}{\sum_{j=1}^{Nfr_{cps_i}-1} j} \tag{7.5}$$

### Non-Functional Requirements Coverage

Non-functional requirements in the context of CPSs are critical. For instance, in software systems, the execution time of a task is a performance issue, but in CPSs, it might be critical for the system to behave correctly [DLSV11, LBB15]. CPSs have different non-functional requirements such as deadlines of the tasks, response time of certain software units or CPU or memory usages. We consider to test these requirements at the HiL level. As another quality measure, we define the Non-Functional Requirements Coverage (nFRC), which measures the number of non-functional requirements covered by a solution $sk$ (i.e., $|nFR_{cps_i} \lessdot sk|$) with respect to the total number of non-functional requirements in configuration $i$ (i.e., $Nnfr_{cps_i}$).

$$nFRCsk = \frac{|nFR_{cps_i} \lessdot sk|}{Nnfr_{cps_i}} \tag{7.6}$$

### Pairwise Non-Functional Requirement Coverage

As in the case of functional requirements, in non-functional requirements conflicts between requirement pairs can also occur. For this reason we define the pairwise non-Functional Requirements Coverage (pw_nFRC), which is calculated as in Equation 7.7. It is expressed as the number of non-functional requirements pairs covered by $sk$ (i.e., $|PAIRnFR_{cps_i} \lessdot sk|$) with respect to all the possible requirements pairs in a configuration $i$ (i.e., $\sum_{j=1}^{Nnfr_{cps_i}-1} j$).

$$pw_nFRCsk = \frac{|PAIRnFR_{cps_i} \lessdot sk|}{\sum_{j=1}^{Nnfr_{cps_i}-1} j} \tag{7.7}$$

### 7.2.4 Cost Measures

Although a test case might be highly effective, its execution cost can also be high. For this reason, it is important to define cost measures so that the selection of the test cases are cost-effective. In our case, two cost measures are defined, which is presented in detail below.

#### Test Execution Time

Unlike in the case of software unit testing, where each test case lasts in the order of some milliseconds [AIB10], the execution of CPS test cases are in the order of some seconds or even minutes. Moreover, the difference of the test execution time from one test case to another can be lengthy. For this reason, we have chosen test execution time as one of the cost measures.

Given a solution $s_k$ of $nt_{sk}$ test cases for testing a configuration $cps_i$, the time required to execute the solution (i.e., $ET_{sk}$) is given in Equation 7.8. $N_{tc}$ is the number of test cases for the test suite in charge of testing the whole configurable CPS (i.e., $TS_{CPSPL}$), and $ET_{tc_i}$ is the time required to execute a test case $i$. Unlike the rest of the measures, the execution time must be normalized to prevent the objectives ranging from 0 to more than 1 (which is not allowed in weight-based search algorithms [WAG13]). This is because test cases can range from a few seconds to around 5500 seconds. We normalize it with the function provided in [GR04] (i.e., $nor(x) = x/(x+1)$).

$$ET_{sk} = nor(\frac{\sum_{i=1}^{nt_{sk}} ETtc_i}{N_{tc}}) \tag{7.8}$$

#### Test Suite Similarity

Two similar test suites across two test levels can be inadequate. Executing the same test cases at the MiL and at the SiL levels may not be effective. Executing test cases that at the MiL level were not executed at the SiL level can help to discover new faults. Thus, it is also possible to employ test suite similarity as another cost measure. We only use this cost measure at the SiL level. At the MiL test level, this cost measure cannot be applied as it is the first test level, and there are no previous solutions. For HiL, the non-functional requirements are tested; employing this cost measure might

lead to unfulfillment of the measures related to non-functional requirements. Thus, we chose not to use this measure at the HiL test level. For the SiL level, this cost measure prevents repeating test cases between the MiL and SiL test levels. $sk_{MiL}$ refers to a solution of $nt_{sk_{MiL}}$ test cases obtained by a search algorithm for the MiL test level. $sk_{SiL}$ is a solution for testing a configuration $cps_i$ at the SiL test level. Similarity ($Sim_{sk_{SiL}}$) is obtained following Equation 7.9, which refers to the number of repeated test cases at the SiL test level as compared to the MiL test level ($|sk_{MiL} \cap sk_{SiL}|$), with respect to the total number of test cases at the MiL level ($nt_{sk_{MiL}}$). 0 means that the solution $sk_{SiL}$ does not contain any test case used at the MiL level, whereas 1 means that the solution $sk_{SiL}$ contains all the test cases executed at the MiL level.

$$Sim_{sk_{SiL}} = \frac{|sk_{MiL} \cap sk_{SiL}|}{nt_{sk_{MiL}}} \tag{7.9}$$

### 7.2.5 Fitness Functions

We chose three test levels: MiL, SiL and HiL. Processor-in-the-Loop (PiL) was not selected due to the fact that its errors can be detected at the HiL level and support is not available for some processors or some simulation tools. Each test level has its own fitness function based on specific objectives. Table 7.1 shows the objectives we selected for each test level.

Table 7.1: Selected objective for each test level

| Test Level | FDC | FRC | PW-RC | nFRC | PW-NFRC | ET | Sim |
|---|---|---|---|---|---|---|---|
| **MiL** | X | X | X | - | - | X | - |
| **SiL** | X | X | X | - | - | X | X |
| **HiL** | X | X | X | X | X | X | - |

The MiL simulation level has testing functional requirements as an objective. For this reason, as a quality measure we chose the FDC, FRC and pw_FRC. As a cost measure, we chose test execution time. Equation 7.10 is employed to calculate the fitness function for test case selection at the MiL level. A lower value of $F_{MiL}$ means a better solution. As for the weights, $w_{fdc}$ is the weight assigned to the FDC objective, $w_{rc}$ to the FRC, $w_{pwrc}$ to pw_FRC and $w_t$ to the test execution time objective.

$$\begin{aligned} F_{MiL} = w_{fdc} * (1 - FDCsk) + w_{rc} * (1 - FRCsk) \\ + w_{pwrc} * (1 - pw_FRCsk) + w_t * ET_{sk} \end{aligned} \tag{7.10}$$

The objective of the SiL level is to test functional requirements employing executable software that represents the embedded software. In this case the objectives

are similar to the MiL test level, therefore we included an extra cost function that consisted of the similarity of test suites. As a result, the selected test cases for the MiL test level are taken into account at the SiL level. The fitness function of the SiL level is calculated following Equation 7.11. The weight assigned to test suite similarity is $w_{sim}$.

$$F_{SiL} = w_{fdc} * (1 - FDCsk) + w_{rc} * (1 - FRCsk) + w_{pwrc} * (1 - pw_FRCsk) + w_t * ET_{sk} + w_{sim} * Sim_{sk_{SiL}} \quad (7.11)$$

The last test level is the HiL phase, where the system is tested in real-time. At the HiL test level, in this case, the real Electronic Control Unit (ECU) is used, and the embedded software is integrated with the rest of the real-time infrastructure (e.g., communications, RTOS, drivers). For this reason non-functional requirements were also tested at the HiL test level. Thus, at this level, we included in the fitness function the nFRC as well as the pw_nFRC as quality measures. As it is not critical to repeat test cases, we did not use the test suite similarity objective at the HiL level. The fitness function for the selection of test cases at the HiL level is computed with Equation 7.12. The weights assigned to nFRC and pw_nFRC are $w_{nfrc}$ and $w_{pwnfrc}$ respectively.

$$F_{HiL} = w_{fdc} * (1 - FDCsk) + w_{rc} * (1 - FRCsk) + w_{pwrc} * (1 - pw_FRCsk) + w_{nfrc} * (1 - nFRCsk) + w_{pwnfrc} * (1 - pw_nFRCsk) + w_t * ET_{sk} \quad (7.12)$$

## 7.3 Empirical Evaluation

The proposed test case selection approach is empirically evaluated in this section.

### 7.3.1 Research Questions

The objective of the empirical evaluation is to assess the performance of the selected algorithms for the test case selection problem of configurable CPS configurations. The following two Research Questions (RQs) are answered with the proposed experiment:

**RQ1:** *Are the selected search algorithms cost-effective as compared to RS?* RS is taken as a baseline to assess whether our test case selection problem is non-trivial to solve [AB11]. Based on our experience, RS could return results that are better than or similar to search algorithms (e.g., GA) in some cases, especially when the

problems are easy to solve [WAG13, WAG15]. Thus, it is important to first compare the approach with RS to see whether the problem is complex enough to apply search algorithms (e.g., GA).

**RQ2:** *Which algorithm shows the best performance?* The goal of this RQ is to obtain the best search algorithm with the aim of integrating it into our search-based test case selection approach.

### 7.3.2 Experiment Setup

Two of the case studies explained in Chapter 4 were used to evaluate the effectiveness of the proposed approach. The first case study was the tank in charge of controlling the level of different kinds of liquids (water or chemical liquid), explained in Section 4.4.3. The second case study was the Unmanned Aerial Vehicle (UAV) (Section 4.4.1). The number of features, constraints as well as functional and non-functional requirements for each case study can be found in Table 7.2. For the tank case study, we used five configurations of different complexities, whereas ten configurations were used in the UAV case study. Each configuration was evaluated with each selected search algorithm (i.e., GA, AVM, Greedy and RS), as well as with different test suite sizes (80, 90, 100, 110 and 120 test cases). Each test case had a set of key attributes. For instance, for the tank case study, the attributes of a test case included the temperature or the level of the liquid. For the case of the UAV, the attributes consisted of the coordinates points as well as several communication variables with the ground station (e.g., different working modes, flying commands). The mean execution time for each test case of the tank case study was 1500 seconds (with the longest execution time of 3700 seconds and shortest execution time of 300 seconds). As for the case study of the UAV, each test case took on average 800 seconds for executing (the longest took 5500 seconds and the shortest 10 seconds). The test cases were generated based on our domain knowledge. Both case studies were modeled in MATLAB/Simulink and evaluated using mutation testing.

Table 7.2: Characteristics of the selected case studies. FR is the number of functional requirements. NFR is the number of non-functional requirements

| Case Study | Features | Constraints | FR | NFR |
|---|---|---|---|---|
| Tank Control | 24 | 2 | 6 | 20 |
| UAV | 46 | 11 | 20 | 40 |

Search algorithms involve randomized variations. For this reason, we run each algorithm 100 times as recommended by Arcuri and Briand [AB11], to reduce the probability of having specific results “by chance”. For the tank case study 25 artificial problems were chosen and for the UAV case study 50. Each artificial problem for each

Table 7.3: Selected weights for each search objective and each case study

| Case Study | Test Level | $w_{fdc}$ | $w_{rc}$ | $w_{pwrc}$ | $w_{nfrc}$ | $w_{pwnfrc}$ | $w_t$ | $w_{sim}$ |
|---|---|---|---|---|---|---|---|---|
| TANK | MiL | 0.4 | 0.15 | 0.15 | - | - | 0.3 | - |
| | SiL | 0.35 | 0.125 | 0.125 | - | - | 0.25 | 0.15 |
| | HiL | 0.35 | 0.1 | 0.1 | 0.1 | 0.1 | 0.25 | - |
| UAV | MiL | 0.4 | 0.15 | - | - | - | 0.3 | - |
| | SiL | 0.35 | 0.25 | - | - | - | 0.25 | 0.15 |
| | HiL | 0.35 | 0.2 | - | 0.2 | - | 0.25 | - |

search algorithm was composed of (1) a specific configuration from the configurable CPS (i.e., $cps_i$ ), (2) specific test suite size and (3) 100 algorithm solutions and their evaluation employing mutation testing and simulation. For instance, Artificial Problem 1: {Case study: Tank, Configuration: $cps_3$, Test Suite Size: 100 test cases, 100 algorithm solutions}.

We employed mutation testing to assess the FDC of our approach, since it has been demonstrated to be a good representation of real faults [JJI+14]. Mutation testing aims at creating some mutants of the system. These mutants are different versions of the system with a specific fault [JH11]. The injected fault is named mutation. The inputs of each mutant are simulated with the same inputs as the non-mutated system. The outputs of the mutants are compared with the non-mutated system. When the outputs differ, it is considered that the mutant has been killed (i.e., the fault has been detected) [JH11]. We injected 20 mutants into each configuration of each case study. The distribution of the mutations was 50% in the cyber layer and 50% in the physical layer. For the cyber layer (i.e., in our case the software), we followed the guidelines proposed in [Mat15, MNBB16], which is based on discussion with industrial practitioners. As for the physical layer, different faults were injected (e.g., noise or stuck-at in sensor signals, short circuits in actuators, etc.). We distributed the type of faults across the whole system to avoid subsumed mutants [PHH+16].

We evaluated our approach taking into account different measures. For MiL and SiL simulations we used the same measures: (1) ET (which measures the time needed to execute the simulation with the selected test cases), (2) mutation score (which measures the percentage of killed mutants in the simulation), (3) FRC (which measures the percentage of functional requirements covered by the selected test cases) and (4) pw_FRC (which measures the percentage of functional requirement pairs covered by the selected test cases). At the HiL level, we used the measures for the MiL and SiL test levels and included two additional measures: (1) nFRC (which measures the percentage of non-functional requirements coverage covered by the selected test cases) and (2) pw_nFRC (which measures the percentage of non-functional requirements pairs covered by the selected test cases).

The weights assigned to each objective were based on discussion with industrial practitioners. We decided to give highest priority to FDC since it is considered the most critical factor when performing test case selection [WAG15]. Test Execution Time was also a priority as the simulation of the systems was costly. Finally, we chose to place less importance on test coverage as it was easier to find solutions achieving a good test coverage data (i.e., FRC, pw_FRC, nFRC, pw_nFRC) rather than high FDC or low Test Execution Time (TET). Table 7.3 provides the selected weights for each objective and each case study. Note that in the UAV case study, pairwise coverage was not selected neither for functional requirements nor non-functional requirements. This was due to the fact that each test case was designed to test only one requirement. Note that the weights were the same for all the algorithms.

The population size for the GA was set to 100 and the number of fitness evaluations was set to 50,000 (i.e., we obtained the optimal solutions after the 50,000th fitness evaluation). We used a standard one-point crossover with a rate of 0.8 and the mutation of a variable is done with the standard probability 1/N, where N is the number of variables (i.e., number of test cases). We used this setup based on other studies and recommendations [AB11, WAG13].

### 7.3.3 Results

We employed statistical tests to assess the RQs. More specifically, we compared the results of the different algorithms with the Mann-Whitney U-test. This test is used to determine the significance of differences between two sets of data. The significance level was set to 95% (i.e., the p-value < 0.05 for the Mann-Whitney U test). In the results table for RQ1 and RQ2, three numbers are shown separated by a slash for each pair of algorithms and each test level. The first number refers to the number of artificial problems for which the algorithm in the left has shown significantly better performance than the algorithm in the right. The second number refers to the number of artificial problems for which the algorithm in the right has shown significantly better performance than the algorithm in the left. Finally, the last number refers to the number of artificial problems for which there was no significant difference between both algorithms.

#### Test Execution Time

Table 7.4 and 7.5 compare the performance of the different algorithms for both case studies for test execution time. In RQ1, where the performance of the selected algorithms with RS is compared, the Mann-Whitney U tests suggest that all the algorithms significantly outperformed RS, for all artificial problems, across the three

test levels (i.e., MiL, SiL and HiL). Regarding RQ2, where the performance of the selected search algorithms is compared, results were not consistent between both case studies. As shown in Table 7.4, for the tank case study Greedy[1] significantly outperformed GA and AVM at the three test levels. AVM also showed significantly better performance than the GA for all the artificial problems. However, in the second case study (Table 7.5), the GA outperformed greedy in 45 problems out of 50 at the MiL level and 43 problems at the HiL level. As compared with AVM, GA significantly outperformed AVM in 43 problems at the MiL level and 26 times at the HiL level. However, at the SiL level, both greedy and AVM showed significantly better performance than the GA in 43 and 39 problems, respectively.

Table 7.4: Results for the Mann-Whitney U-Test for time and tank case study

| | Algorithm | MiL | SiL | HiL |
|---|---|---|---|---|
| **RQ1** | RS vs GRE | 0/25/0 | 0/25/0 | 0/25/0 |
| | RS vs GA | 0/25/0 | 0/25/0 | 0/25/0 |
| | RS vs AVM | 0/25/0 | 0/25/0 | 0/25/0 |
| **RQ2** | GRE vs GA | 25/0/0 | 25/0/0 | 25/0/0 |
| | GRE vs AVM | 25/0/0 | 23/0/2 | 25/0/0 |
| | GA vs AVM | 0/25/0 | 0/25/0 | 0/25/0 |

Table 7.5: Results for the Mann-Whitney U-Test for time and UAV case study

| | Algorithm | MiL | SiL | HiL |
|---|---|---|---|---|
| **RQ 1** | RS vs GRE | 0/50/0 | 0/50/0 | 0/50/0 |
| | RS vs GA | 0/50/0 | 0/50/0 | 0/50/0 |
| | RS vs AVM | 0/50/0 | 0/50/0 | 0/50/0 |
| **RQ 2** | GRE vs GA | 5/45/0 | 47/0/3 | 7/43/0 |
| | GRE vs AVM | 5/30/13 | 21/22/7 | 5/29/16 |
| | GA vs AVM | 43/2/5 | 4/39/7 | 26/01/23 |

In terms of reduced time, Figure 7.5 depicts the percentage of saved time of the different search algorithms with respect to RS at different levels for each selected configurations. Our approach reduced time by 80% on average, including in some cases more than 90% (especially at the SiL test level). It is worth noting that, in terms of time the difference between the search algorithms is not significant. In the case of the tank case study, for the HiL test level, the amount of time reduced by the GA dropped to 50%. This was because this algorithm tried to achieve as much pairwise requirement coverage as possible.

### Fault Detecion Capability

Table 7.6 and 7.7 compare the performance of different algorithms when revealing the injected faults. In this case, RS killed all the mutants in all the runs. For the

[1] Greedy is represented in all tables as “GRE”

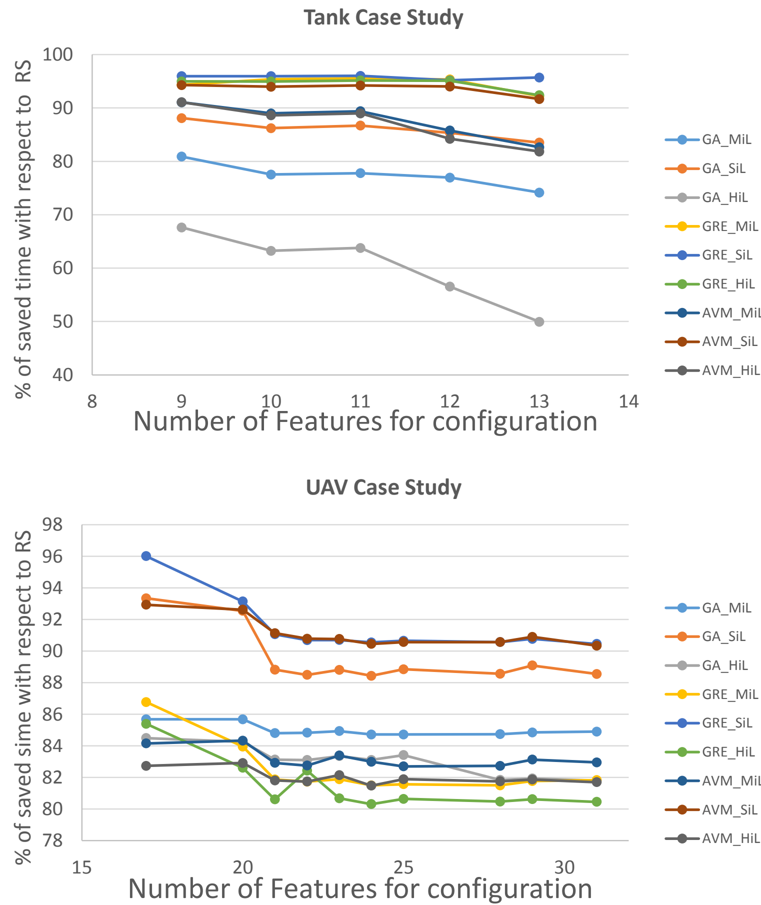



Figure 7.5: Average percentage of time reduced by each search algorithm in each level as compared with RS with respect to configuration complexity

first case study, greedy showed the worst performance as compared to RS, where RS significantly outperformed it in 24 problems at the MiL level and in 5 problems at the SiL level. RS only outperformed GA and AVM at the MiL level in 5 and 7 problems for the first case study. However, at the SiL level, GA and AVM managed to reveal all the faults. On the contrary, for the second case study, RS outperformed greedy in only 2 problems out of 50. The GA and AVM were outperformed in 20 problems by RS at the MiL test level. However, at the SiL level, all the search algorithms revealed all the faults. Regarding RQ2, where the performance between each pair of algorithms is compared, for the first case study, GA showed the best results whereas greedy performed best in the second case study. These results reveal that perhaps, it would be more appropriate to be more conservative when assigning the weights, giving further importance to the FDC.

Table 7.6: Results for the Mann-Whitney U-Test for fault detection and tank case study

| | Algorithm | MiL | SiL | HiL |
|---|---|---|---|---|
| | RS vs GRE | 24/0/1 | 5/0/20 | 0/0/25 |
| **RQ1** | RS vs GA | 5/0/20 | 0/0/25 | 0/0/25 |
| | RS vs AVM | 7/0/18 | 0/0/25 | 0/0/25 |
| | GRE vs GA | 0/24/1 | 0/5/20 | 0/0/25 |
| **RQ2** | GRE vs AVM | 0/21/4 | 0/5/20 | 0/0/25 |
| | GA vs AVM | 7/4/14 | 0/0/25 | 0/0/25 |

Table 7.7: Results for the Mann-Whitney U-Test for fault detection and UAV case study

| | Algorithm | MiL | SiL | HiL |
|---|---|---|---|---|
| | RS vs GRE | 2/0/48 | 0/0/50 | 0/0/50 |
| **RQ1** | RS vs GA | 20/0/30 | 0/0/50 | 0/0/50 |
| | RS vs AVM | 20/0/30 | 0/0/50 | 0/0/50 |
| | GRE vs GA | 18/02/30 | 0/0/50 | 0/0/50 |
| **RQ2** | GRE vs AVM | 18/02/30 | 0/0/50 | 0/0/50 |
| | GA vs AVM | 02/10/40 | 0/0/50 | 0/0/50 |

### Test Coverage

Table 7.8 and 7.9 compare the performance of the different algorithms for the achieved test coverage. For RQ1, where search algorithms are compared with RS, the following can be noted: for Requirements Coverage (RC), if the system did not have many requirements, RS and the search algorithms did not show significant difference, which means that the problem is not complex (e.g., Table 7.8). However, if the system had many requirements, search algorithms considerably outperform RS (e.g., Table 7.9). As for pairwise requirement coverage (PWRC), search algorithms considerably outperformed RS in most of the problems at the three levels. The same happened with non-functional requirements, which are only applied at the HiL level. Regarding pairwise non-functional requirements coverage, GA outperformed RS in 25 out of 25 problems (Table 7.8). However, surprisingly, RS outperformed greedy in 19 problems and showed similar performance in 6 problems. Something similar happened with AVM, where RS outperformed it in 9 problems. This explains the good performance of AVM and greedy for the objective of the test execution time.

RQ 2 compares the performance between two different algorithms. For the UAV case study, the three search algorithms showed similar performance. However, in the tank case study, where pairwise test coverage is also analyzed, GA showed significantly better performance, especially in the pairwise non-functional requirement coverage, where GA outperformed greedy and AVM in 25 out of 25 problems.

Table 7.8: Results for the Mann-Whitney U-Test for requirements coverage and tank case study. RC refers to functional requirements coverage, PWRC refers to pairwise functional requirements coverage, NFRC refers to non-functional requirements coverage and PWNFRC refers to pairwise non-functional requirements coverage.

| | | RC | | | PWRC | | | NFRC | PWNFRC |
|---|---|---|---|---|---|---|---|---|---|
| | **Algorithm** | **MiL** | **SiL** | **HiL** | **MiL** | **SiL** | **HiL** | **HiL** | **HiL** |
| **RQ1** | RS vs GRE | 0/0/25 | 0/0/25 | 0/0/25 | 3/15/7 | 1/16/8 | 0/19/6 | 10/13/2 | 19/0/6 |
| | RS vs GA | 0/0/25 | 0/0/25 | 0/0/25 | 0/19/6 | 0/20/5 | 0/20/5 | 0/23/2 | 0/25/0 |
| | RS vs AVM | 0/0/25 | 0/0/25 | 0/0/25 | 0/19/6 | 0/19/6 | 0/20/5 | 1/19/5 | 9/12/4 |
| **RQ2** | GRE vs GA | 0/0/25 | 0/0/25 | 0/0/25 | 0/0/25 | 0/5/20 | 0/1/24 | 0/10/15 | 0/25/0 |
| | GRE vs AVM | 0/0/25 | 0/0/25 | 0/0/25 | 3/5/17 | 7/5/13 | 0/1/24 | 3/10/12 | 4/20/1 |
| | GA vs AVM | 0/0/25 | 0/0/25 | 0/0/25 | 8/0/17 | 11/0/14 | 0/0/25 | 3/0/22 | 25/0/0 |

Table 7.9: Results for the Mann-Whitney U-Test for requirements coverage and UAV case study

| | | RC | | | NFRC |
|---|---|---|---|---|---|
| | **Algorithm** | **MiL** | **SiL** | **HiL** | **HiL** |
| **RQ1** | RS vs GRE | 0/37/13 | 0/35/15 | 0/34/16 | 0/45/5 |
| | RS vs GA | 0/37/13 | 0/35/15 | 0/34/16 | 0/45/5 |
| | RS vs AVM | 0/37/13 | 0/35/15 | 0/34/16 | 0/45/5 |
| **RQ2** | GRE vs GA | 0/0/50 | 0/0/50 | 0/0/50 | 0/0/50 |
| | GRE vs AVM | 0/0/50 | 0/0/50 | 0/0/50 | 0/0/50 |
| | GA vs AVM | 0/0/50 | 0/0/50 | 0/0/50 | 0/0/50 |

### 7.3.4 Discussion

RQ1 aimed at assessing whether the proposed problem was non-trivial to solve by comparing the proposed search algorithms with RS. In most of the objectives, the proposed algorithms were significantly better than RS. However, in some cases, RS was more effective in detecting faults. Although the proposed algorithms detected most of the faults, there were some solutions, especially at the MiL level, where one or two faults were not revealed. However, all the faults were revealed at the SiL level with the GA and AVM. In terms of time, the average amount of time saved as compared to RS was around 82 % for both case studies. At the SiL phase, this amount increased to around 90 %. This effect could have been caused by the similarity objected added for the SiL test level. This objective could have removed long test cases not to be repeated at the SiL test level. As for the test coverage, all the algorithms improved RS, especially when there were many requirements. Regarding functional requirements coverage, for the first case study, results were the same for the selected search algorithms and RS. This may have been because the low number of functional requirements that this case study had. Nevertheless, for the rest of test coverage measures, the proposed search algorithms significantly outperformed RS. Moreover, for functional requirements coverage of the UAV case study, the proposed algorithms performed better than RS. Thus, the first RQ can be answered as follows:

> *Based on the experimental results and the statistical tests, it can be assumed that the selected search algorithms perform in general better than RS, and thus, the problem to be solved is non-trivial.*

RQ2 compared the performance of the selected search algorithms (i.e., GA, AVM and Greedy). For the test execution time, statistical tests suggest that in the tank case study greedy performed the best, whereas in the case study related to the UAV system, GA was better. The reason for this might be that in the tank case study the GA focused on selecting test cases that were better for pairwise functional and non-functional requirements. However, for the objective related to test execution time, the time saving was not that high, except for the case of the GA at the HiL level for the tank case study. In that case, the GA reduced around 60 % of time, whereas AVM and greedy reduced around 90 % as compared with RS. Regarding test effectiveness, the GA was in general the algorithm performing best. In the performed experiment, as compared to RS, the only exception was for revealing faults at the MiL level for the UAV case study. However, in that case, for the SiL and the HiL level, the GA performed well. Subsequently, the second RQ can be answered as follows:

> *Despite for the test execution time in the tank case study local search algorithms being better, overall, the algorithm showing best performance was GA.*

*Concluding Remark:* All the selected search algorithms considerably outperformed RS, especially when reducing the test execution time. There are some solutions that did not reveal all the faults at the MiL level, but all the faults were revealed at the subsequent levels. In addition, by employing the proposed search algorithms, the execution time for MiL and SiL test levels lasted less time than executing just the MiL level with the test cases selected by RS. We believe that the fault detection capability can be improved by assigning a higher weight to this objective. Based on our experimental results, we recommend applying GA along with our defined cost-effectiveness measures for solving the test case selection problem. For complex problems we believe that the GA solves them better than greedy and AVM. Finally, it is important to be conservative when tunning the weights for detecting all the faults at the MiL test level.

### 7.3.5 Threats to Validity

We have identified some threats to validity for the performed experimental evaluation.

**Internal validity**: Regarding ***internal validity*** threats, one could be that the configurations related to the GA (e.g., population size, crossover rate, etc.) were not changed. However, the selected settings are in accordance with the common guidelines in the literature and other studies related to the application of search algorithms to testing [AB11, WAG15]. Another ***internal validity*** threat refers to the type and amount of mutants we employed. We used 20 mutants, but a system might have more faults. Nevertheless, we tried to reduce this threat injecting different types of faults distributed across the whole system (i.e., the physical and cyber layer).

**External validity**: An ***external validity*** threat with any experiment is related to the number of case studies used. Two case studies might not be enough to generalize the results, but we tried to minimize this threat by using two case studies from different domains and different complexities.

**Conclusion validity**: A ***conclusion validity*** threat involves randomized algorithms, which produce random variations [WAG15]. To reduce this threat, we repeated the executions of the algorithms 100 times and we applied statistical analysis.

**Construct validity**: The ***construct validity*** threat is that the measures used are not comparable across the algorithms. In our case, we used the same stopping criterion for all the algorithms, i.e., the number of fitness evaluations (50,000 times), which is a comparable measure across all the algorithms.

## 7.4 Related Work

According to Lopez-Herrejon et al., domain testing is the field where search-based software engineering is most used [LHLE15]. According to [LKL12], test case selection methodologies are marginally investigated in the context of Software Product Line (SPL) testing at the application engineering layer. Nevertheless, there are approaches that considered using search-based software engineering at the application engineering level testing similar to the approach presented in this chapter (e.g., [WAG15, ASE15a]). Wang et al. proposed a test suite minimization approach for reducing test cases in SPLs [WAG13, WAG15]. In [WAG13], they proposed to apply weight-based genetic algorithms for cost-effectively testing SPLs and in [WAG15] they defined more cost-effectiveness measures and included more diverse search algorithms. Our approach builds upon the approaches proposed in [WAG13, WAG15] by: (1) testing configurable CPSs and not SPLs, (2) selecting test cases and not minimizing them, (3) employing simulation and mutation testing for evaluation and (4) selecting test cases for different levels. Stricker et al. considered a model-based technique relying on data-flow dependencies to select test cases for customer-specific prod-

ucts with the objective of avoiding redundant test activities [SMP10]. As compared with [SMP10], there are at least two differences in our work, which include: (1) we defined corresponding objectives for test case selection at the three levels of CPSs (e.g., non-functional requirement coverage for the HiL level), which is not addressed in [SMP10]; and (2) our approach is based on search algorithms which are easy to implement and apply while [SMP10] focused on a model-based approach, which requires modeling effort based on the domain knowledge.

Several works have proposed search-based approaches for selecting or minimizing test cases of a test suite (e.g., [YH07, HB10, PWAY16, LFN+17]). Our test case selection method faces several differences to the works in the current state of the art in this field. First, we focus on selecting test cases for configurable CPSs configurations. To this end, we integrated our approach with a variability notation to warrant a systematic and a fully automated test selection methodology. Second, the approach selects test cases for different test levels (i.e., MiL, SiL and HiL). Each of these test levels have their own characteristics, and thus, we developed three independent fitness functions to satisfy the particularities of each of the test levels when selecting test cases. Lastly, we provide an empirical evaluation considering both, local and global search algorithms together with 75 artificial problems and we employ mutation testing to assess the capacity of the selected test cases when finding faults.

## 7.5 Conclusion and Future Work

This chapter proposed an approach for the selection of test cases for simulation-based testing of configurable CPSs. Two main steps are performed in our approach: (1) the feature modeling part for the selection of configurations and their requirements and (2) a search-based test case selection for three different test levels (i.e., MiL, SiL and HiL). We empirically evaluated three search algorithms with the aim of integrating the best algorithm into our approach. Although for some cases Greedy and AVM, which are local search algorithms, showed very good performance, we believe that the most appropriate algorithm for solving the test case selection problem is the GA. The proposed search-based test case selection approach manages to reduce more than 80 % the costs for testing configurable CPSs configurations as compared to RS while achieving better test quality.

In the future, we plan to conduct investigations from the following perspectives: (1) further studying the impact of assigning different weights to each objective on the performance of search algorithms; (2) employing other search algorithms (e.g., bacteriologic algorithm) for assessing the performance; (3) investigating new objectives,

e.g., higher strength coverage; and (4) applying our approach in real industrial settings for assessing to what extent the current practice of testing CPS product lines can be improved.

Chapter 8

# Test Case Prioritization

In this chapter we propose a search-based approach that aims to cost-effectively optimize the test process of configurable Cyber-Physical System (CPS) by prioritizing the test cases that are executed in specific products at different test levels (i.e., Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL)). The prioritized test suite aims at reducing the fault detection time, the simulation time and the time required to cover functional and non-functional requirements.

We compared our approach by integrating four search algorithms as well as Random Search (RS) using four case studies. As compared with RS, the search algorithms managed to reduce the fault detection time by 47%, the simulation time by 23%, the functional requirements covering time by 22% and the non-functional requirements covering time by 47%. Moreover, we observed that the performance of search algorithms varied for different case studies but the local search algorithms were more effective than global search algorithms.

## 8.1 Introduction

The process of testing configurable CPSs is a time consuming process, and as a result, cost-effective methods are required to optimize the test and validation stages. While most of the papers in the product line engineering community focus on deriving relevant products following Combinatorial Interaction Testing (CIT) techniques [$PSK^{+}10$, $POS^{+}12$, KKLH09] or prioritizing them [SSRC14a, SSPRC15, $PSS^{+}16$, $DPC^{+}14$, $DPC^{+}15$, $AHTM^{+}14$, $AHTL^{+}16$], in this dissertation we focus on optimizing the test process at the application engineering level. While the previous chapter focuses on test case selection, this chapter focuses on test case prioritization. To this end, a search-based approach is proposed to optimize the testing process of configurable CPSs by prioritizing the order in which the test cases are triggered for each test level (i.e., MiL,SiL and HiL).

When testing CPSs at system level, the execution time of a test case is long, unlike

unit testing, where the execution time of a test case is in the order of milliseconds [AIB10]. Furthermore, the test suites for testing CPSs are often composed of many test cases, and thus, the search space for test prioritization is huge (i.e., given a set of N test cases, the search space is N!, which means that for a test suite of 50 test cases $3.04 \times 10^{64}$ possible prioritization solutions are possible). As a consequence, exploring the whole solution space could be impracticable, and thus a search process is mandatory to efficiently prioritize the test cases. By means of our test case prioritization approach, the overall test execution time, the time to detect faults and the time to test requirements is reduced in the context of configurable CPSs.

The proposed approach has been empirically evaluated using the four case studies presented in Chapter 4 with different configurations each. We compared four search algorithms (two local search algorithms and two global search algorithms) taking RS as a baseline. Results indicate that the selected search algorithms are significantly better than RS, what implies that the problem to solve is non-trivial. In fact, when compared to RS, on average, the selected algorithms can improve the fault detection time on average 47%, the simulation time on 23%, the functional requirements covering time on 22% and the non-functional requirements covering time on 47%. Moreover, local search algorithms significantly outperformed global search algorithms, being in general the Alternating Variable Method (AVM) algorithm better at reducing the faults detection time and the simulation time, and the Greedy algorithm better at reducing the requirements covering time.

The Chapter is structured as follows. Section 8.2 presents our test case prioritization approach for testing configurable CPSs. The experimental evaluation that was performed to evaluate the approach is presented in Section 8.3. Results for the performed experiments and their discussion are presented in Sections 8.4 and 8.5. The main threats to validity are discussed in Section 8.6. Section 8.7 positions our work with similar techniques in the field of product line engineering and test case prioritization. Finally, Section 8.8 summarizes the conclusions of our study and future work.

## 8.2 Search-Based Test Case Prioritization Approach

The overall overview of our approach for test case prioritization is depicted in Figure 8.1. A feature model and the configuration file of the product to be tested is first processed by the requirements parser. This parser obtains the requirements assigned to the product to be tested. This information is employed by the search algorithm to prioritize the test cases in such way to reduce the amount of time required to test

these requirements. In addition, the search algorithm employs information related to user preferences, which is highly valued in testing in order to give preference to some objectives rather than to others. The search algorithm also uses information related to previously executed test cases in other products, which permits a more precise test prioritization. When the search algorithm returns a prioritized test suite, this is executed using simulation, and when the execution of test cases is finished, the test results are updated in the test history. This process is iteratively repeated every time a product of the configurable CPS is required to be tested.

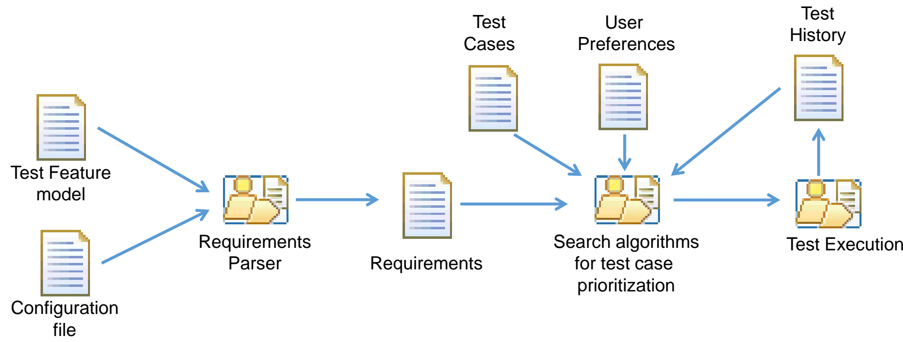



Figure 8.1: Overall overview of the proposed approach for test case prioritization

### 8.2.1 Basic Concepts

Let $CCPS$ be a configurable CPS that can be configured into $N$ configurations $cps$, i.e., $CCPS = \{cps_1, cps_2, ..., cps_N\}$.

Let $FM$ be a feature model that captures the variability of $CCPS$ with $Nf$ number of features ($f$) and $Nc$ number of constraints $c$, i.e., $FM_f = \{f_1, f_2, ..., f_{Nf}\}$, $FM_c = \{c_1, c_2, ..., c_{Nc}\}$.

Let $FR$ be a set of $Nfr$ functional requirements ($fr$) of $CCPS$, and $NFR$ are a set of $Nnfr$ non-functional requirements ($nfr$) of $CCPS$, i.e., $FR = \{fr_1, fr_2, ..., fr_{Nfr}\}$ and $NFR = \{nfr_1, nfr_2, ..., nfr_{Nnfr}\}$.

Let $TS_{CCPS}$ be a test suite of $Ntc$ test cases ($tc$) that tests $CCPS$, i.e., $TS_{CCPS} = \{tc_1, tc_2, ..., tc_{Ntc}\}$.

Let $F_{cps_i} = \{f_1, f_2, ..., f_{Nf_{cps_i}}\}$ be the features corresponding to a specific configuration $i$, where $F_{cps_i}$ is a subset of $FM$, $f_j$ can be any feature in $FM$ (i.e., $f_j \in FM$). $Nf_{cps_i}$ is the number of features representing $Nf_{cps_i}$, where $Nf_{cps_i} \leq NF$ [BSRC10]. Moreover, $cps_i$ satisfies those constraints specified in $FM_c$.

Let $FR_{cps_i} = \{fr_1, fr_2, ..., fr_{Nfr_{cps_i}}\}$ be a set of $Nfr_{cps_i}$ functional requirements for the configuration $i$, $fr_j$ can be any functional requirement in $FR$ (i.e., $fr_j \in FR$) and $Nfr_{cps_i}$ are the number of functional requirements in $cps_i$, where $Nfr_{cps_i} \leq Nfr$. Accordingly, let $NFR_{cps_i} = \{nfr_1, nfr_2, ..., nfr_{Nnfr_{cps_i}}\}$ be a set of $Nnfr_{cps_i}$ non-functional requirements corresponding to the configuration $i$, $nfr_j$ can be any non functional requirement in $NFR$ (i.e., $nfr_j \in NFR$) and $Nnfr_{cps_i}$ are the number of non functional requirements in $cps_i$, where $Nnfr_{cps_i} \leq Nnfr$.

$TS_{cps_i} = \{tc_1, tc_2, ..., tc_{Ntc_{cps_i}}\}$ is a test suite of $Ntc_{cps_i}$ test cases that tests configuration $cps_i$ (i.e., $FR_{cps_i}$). Any test case $j$ corresponding to $TS_{cps_i}$ can be any test case from $TS_{CCPS}$ (i.e., $tc_j \in TS_{CCPS}$) and $Ntc_{cps_i} \leq Ntc$. Given a test suite $TS_{cps_i}$, the goal of our approach is to find a test order that fulfills the objectives of the different test levels (i.e., MiL, SiL and HiL).

### 8.2.2 Variability Management for Test Optimization

As proposed in our previous chapter for test case selection, in this case we also combine the test case prioritization approach with a variability modeling tool. We have employed the tool FeatureIDE [TKB+14] to manage the different features of our systems. The feature model is divided into two main parts. In the first part the variability of the system (e.g., the variability in sensors, actuators, software system, etc.) is modeled. In the second part, the variability related to functional and non-functional requirements is included. Later, the features of both parts are integrated by means of cross-tree constraints. This allows for the automatic selection of the features related to the test system when the features of the system are selected. This permits all these features to be stored into a configuration file, which is later parsed by our prioritization algorithms to perform an efficient test prioritization. For deeper detail of how variability is managed refer to Section 7.2.1.

### 8.2.3 Weight-based Search Algorithms

Weight-based search algorithms convert a multi-objective problem into a single objective function by assigning a particular weight to each optimization objective [WAG15]. Given that there are $N$ objectives (*Obj*), each *Obj* should range between 0 and 1 (i.e., $0 \leq Obj \leq 1$) [WAG13]. If the magnitude of the objectives does not permit a range between 0 and 1, these values must be normalized. The sum of all weights ($w$) must be 1 (i.e., $\sum_{i=1}^{N} w_i = 1$)[WAG13]. The general fitness function for Weight-based search algorithms is given in Equation 8.1.

$$FitnessFunction = \sum_{i=1}^{N} w_i * Obj_i \tag{8.1}$$

Weight-based search algorithms were used in this study for different reasons. The first reasons is that they allow optimization of a multi-objective problem into single objective algorithms (such as Greedy or AVM). Moreover, they let test engineers give preference to a set of predefined properties by assigning higher or lower weights to each objective. A higher weight means that the objective to which the weight is assigned has greater importance. The weights of each objective can be assigned either statically (i.e., a predefined set of weights are assigned to each objective), or dynamically (i.e., the weights are assigned randomly in each generation) [WAG13].

### 8.2.4 Cost-Effectiveness Measures

This section presents the selected cost-effectiveness objectives to build the fitness function.

#### Test Case Execution Time

The execution time of a reactive test case varies depending on the number of states of the test case, i.e., time for switching from a state to another, as well as the time needed to initialize and finalize the test case. Based on the model presented in Figure 2.4, given a Prioritized Test Suite (PTS), of $N$ Test Cases ($TC$s), (i.e., $PTS = \{TC_1, TC_2, ..., TC_N\}$), $ET_{TC_i}$ is the estimated execution time of a test case corresponding to the position $i$ of PTS. To estimate the execution time of a reactive test case, three phases have to be taken into account (1) initialization time, (2) test execution time and (3) test finalization time.

In reactive test cases, the test execution time can vary from one test case to another, for instance by the number of test states that the reactive test case can have. However, this time cannot be minimized by test case prioritization, unlike test initialization time. The initialization time of a reactive test case varies based on the previously executed test case. By prioritizing the test cases efficiently, the initialization part of the test case can be skipped. Figure 8.2 illustrates an example of when the initialization phase can be skipped and when not. TC1 sets the system with the engine turned on and a speed of 90 km/h before triggering TC3. The initialization of TC3 consists of turning the engine on and setting the speed to 90 km/h. Thus, if TC3 was executed after TC1, its initialization phase would be skipped. On the contrary, TC2 sets the system with the engine turned off and a speed of 0 km/h. If TC3 is executed after TC2 the initialization phase cannot be skipped: the engine would need to be turned on and speed set to

90 km/h. Thus, as stated in Equation 8.2, the initialization time of a test case in the position $i$ ($IT_{TC_i}$) depends on the finalization state of the previously executed test case ($TC_{i-1}$), as well as the initialization state of the test case $i$.

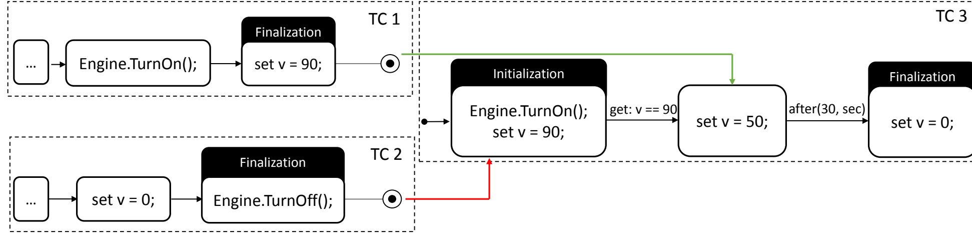



Figure 8.2: Example of how to reduce initialization time of reactive test cases with test case prioritization

$$IT_{TC_i} = time(S_{TC_{i-1_{fin}}}, S_{TC_{i_0}}) \tag{8.2}$$

The test execution time ($TET$) of a test case in position $i$ of PTS depends on the time needed to execute all the states. Thus, given that a reactive test case has NStates number of states in the test case execution phase, the $TET$ is estimated as shown in Equation 8.3. $S_{TC_{i_j}}$ is the state $j$ of the test case in position $i$. In the context of this study, reactive test cases do not contain loops.

$$TET_{TC_i} = \sum_{j=1}^{NStates} time(S_{TC_{i_{j-1}}}, S_{TC_{i_j}}) \tag{8.3}$$

The finalization time ($FT$) of a reactive test case $i$ is calculated taking into account the last state of the test execution phase (i.e., $S_{TC_{i_{NStates}}}$) and the finalization state (i.e., $S_{TC_{i_{fin}}}$), as expressed in Equation 8.4.

$$FT_{TC_i} = time(S_{TC_{i_{NStates}}}, S_{TC_{i_{fin}}}) \tag{8.4}$$

Thus, the estimation time for a test case in position $i$ of PTS is performed adding the three elements on Equations 8.2, 8.3 and 8.4, as represented in Equation 8.5.

$$ET_{TC_i} = IT_{TC_i} + TET_{TC_i} + FT_{TC_i} \tag{8.5}$$

### Simulation time

In test case prioritization, the whole test suite is typically executed. Thus, it is also important to reduce as much as possible the overall test execution time. As the

execution time of the reactive test cases varies based on the prioritization, we propose to reduce the overall simulation time that it takes to execute a specific test case. This is performed taking the initialization time of each test case into account, which is given in Equation 8.2. Given that PTS has N test cases, the simulation time is tried to be reduced following the Equation 8.6. Notice that the rest of the test case execution time provided in Section 8.2.4 is not calculated since the $TET_{TC_i}$ and $FT_{TC_i}$ would remain constant and it would not have an influence in the final simulation time.

$$SimTime = \sum_{i=1}^{N} IT_{TC_i} \tag{8.6}$$

### Fault Detection Capability

To measure the effectiveness of a test case, we use the Fault Detection Capability (FDC), which is the success rate of the test case [WAG13]. Another test quality metric for effectiveness could be the percentage of faults detected, which requires the different faults to be diagnosed. However, the faults might be difficult to diagnose in CPSs. For instance, there are cases in which two different faults provoke similar symptoms, whereas in other cases, one fault can provoke many different symptoms. In addition, another effectiveness metric could be test coverage, which is widely used in software systems to prioritize test cases [CM13]. Measuring different test coverage metrics in software systems might be easy, but in CPSs, apart from software a physical layer is also involved, encompassing sensors, actuators, mechanical elements, etc. Measuring test coverage in the physical layer of a CPS is not easy as their models are not discrete models, unlike in software. To overcome this fault localization problem, we chose the FDC metric, which has also been applied in similar contexts [WAG13, WAG15]. FDC identifies whether executing a test case can detect faults rather than distinguishing specific faults detected by the test case.

The FDC of a test case ($TC_i$) is given by the number of times it has been successful when testing other configurations ($NumSuc_{TC_i}$) with respect to the number of times the test case has been executed, i.e., the number of times it has been successful and the number of times it has failed ($NumFail_{TC_i}$). Equation 8.7 reflects how the FDC is computed for $TC_i$. In this study, we consider a test case successful if during its execution it was capable of detecting faults. On the contrary, we consider a test case has failed if it has not detected any faults.

$$FDC_{TC_i} = \frac{NumSuc_{TC_i}}{NumSuc_{TC_i} + NumFail_{TC_i}} \tag{8.7}$$

### Functional requirements coverage

When testing the functionality of any system, functional requirements are taken into account. Our approach is designed to reduce the time to test functional requirements. To achieve this objective, we propose to analyze the time required by a specific PTS to test all the functional requirements. The time required by a PTS to test all the requirements is given in Equation 8.8. Given a set of $N_{fr}$ functional requirements, $N_{tc_{fr}}$ is the position of the last test case that covers the last functional requirement of the configuration. Equation 8.8 measures the time required by PTS to test the $N_{fr}$ functional requirements in $cps_i$.

$$FRCT = \sum_{i=1}^{N_{tc_{fr}}} ET_{TC_i} \tag{8.8}$$

### Non-Functional requirements coverage

Non-functional requirements are specified to test non-functional properties of the system, such as the execution time of certain tasks, or the quality of service of a communication system [ATF09]. In the CPS context, non-functional requirements are critical. For instance, the execution time of certain tasks or their deadline might lead to a completely new behavior of the CPS, unlike in general purpose software, in which these properties are related to performance [DLSV11, AWSE16a]. Moreover, some parts such as the network environment needs to be throughly tested [AWSE16a]. All these properties are challenging to be tested at the MiL and SiL test levels, and thus, they are tested at the HiL test level [AWSE16a]. The time required by a PTS to test all the non-functional requirements is given in Equation 8.9. Given a set of $N_{nfr}$ non-functional requirements, $N_{tc_{nfr}}$ is the position of the last test case that covers the last functional requirement in the product. Equation 8.8 measures the time required by PTS to test the $N_{nfr}$ functional requirements in $cps_i$.

$$NFRCT = \sum_{i=1}^{N_{tc_{nfr}}} ET_{TC_i} \tag{8.9}$$

## 8.2.5 Fitness Function

The fitness function for a solution is shown in Equation 8.10. A lower value of $F$ indicates a better solution with higher effectiveness (i.e., FDC) and lower cost (i.e., test execution time). Each objective has its own weight. The higher the weight, the higher the preference of the objective. Notice that the weights might be assigned

differently for each test level. For instance, at the MiL and SiL test level, non-functional requirements of CPSs cannot be tested, and thus $w_{nfrct}$ is set to 0. Finally, $i$ is the position of a test case in PTS.

$$F = w_{fdc} \times (1 - nor(\sum_{i=1}^{\#TC} \frac{FDC_{TC_i}}{i})) + w_{et} \times nor(\sum_{i=1}^{\#TC} \frac{ET_{TC_i}}{i}) + \\ w_{simtime} \times nor(SimTime) + \\ w_{frct} \times nor(FRCT) + w_{nfrct} \times nor(NFRCT) \quad (8.10)$$

Given a PTS, of $N$ $TC$s, (i.e., PTS $= \{TC_1, TC_2, ..., TC_N\}$), the fitness function must give preference to the test cases located in the initial positions of the PTS. This means that the first scheduled test case (i.e., $i = 1$) is more important than the last one (i.e., $i = N$) when trying to find faults as quickly as possible. Ranking is given by dividing the cost-effectiveness measures by the position of the test cases (i.e., $i$) of a certain solution PTS. Subsequently, $F$ is more sensitive to the cost-effectiveness measures of the test cases located in the initial positions of PTS. It is noteworthy that this is not applied for simulation time, Functional Requirements Covering Time (FRCT) and Non-Functional Requirements Covering Time (NFRCT). In addition, the objectives of the fitness function must range between 0 and 1. Thus, we use Equation 8.11 to normalize the cost-effectiveness measures [GR04].

$$nor(x) = \frac{x}{x+1} \quad (8.11)$$

## 8.3 Experimental Setup

The aim of the experiment is to evaluate the performance of the proposed search-based test case prioritization algorithms. RS has been used as a baseline to assess that the problem to be solved is not trivial (Research Question 1 (RQ1)). We wanted to identify the best search algorithm to solve test case prioritization in the context of configurable CPSs (RQ2). Finally, we wanted to assess the scalability of the search approach by incrementing the test suite size (RQ3). The RQs raised are the following:

**RQ 1:** Are the selected search algorithms cost-effective as compared to RS?

**RQ 2:** Which of the selected search algorithms fares best when solving test case prioritization problem?

**RQ 3:** How does the increment of test cases impact the performance of the selected search algorithms?

### 8.3.1 Case Studies

The four case studies presented in Chapter 4 were employed to assess the test case prioritization approach presented in this chapter. The characteristics of the selected case studies and products are shown in Tables 8.1 and 8.2. Notice that the requirements related to the DC Engine case study are artificial requirements. We employed a high amount of artificial requirements in this case study to assess the scalability of the approach.

Table 8.1: Main characteristics of each case study. Column blocks is referred to the number of blocks of each Simulink model. FR refers to the total number of functional requirements. NFR refers to the total number of non-functional requirements. Feature and constraints refer to the number of feature and constraints related to the feature model of the case study.

| | **Blocks** | **FR** | **NFR** | **Features** | **Constraints** |
|---|---|---|---|---|---|
| ACC | 415 | 19 | 24 | 14 | 2 |
| UAV | 843 | 20 | 40 | 46 | 5 |
| TANK | 112 | 6 | 20 | 24 | 2 |
| DC ENGINE | 257 | 40 | 80 | 8 | 1 |

Table 8.2: Main characteristics of selected product. NFR refers to the total number of non-functional requirements. The column features refers to the number of features that the selected product configuration has.

| | **Product 1** | | | **Product 2** | | | **Product 3** | | | **Product 4** | | | **Product 5** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FR | NFR | F | FR | NFR | F | FR | NFR | F | FR | NFR | F | FR | NFR | F |
| ACC | 6 | 8 | 7 | 7 | 12 | 8 | 13 | 18 | 10 | 18 | 22 | 12 | 19 | 24 | 14 |
| UAV | 10 | 13 | 17 | 11 | 16 | 20 | 13 | 20 | 22 | 17 | 24 | 25 | 20 | 40 | 29 |
| TANK | 3 | 11 | 14 | 4 | 14 | 16 | 5 | 14 | 18 | 5 | 16 | 21 | 6 | 20 | 23 |
| DC Eng. | 20 | 45 | 3 | 30 | 60 | 4 | 35 | 70 | 5 | 40 | 80 | 6 | | | |

### 8.3.2 Evaluation Metrics

We employed four different evaluation metrics to assess the performance of the selected algorithms. Two of them measured the fault revealing capability of the approach, one of them the simulation time to execute the entire test suite and an additional one (divided into two parts) the time required by a test suite to cover functional and non-functional requirements.

#### Faults Detection Time (FDT)

To evaluate the proposed approach, we defined a metric called Faults Detection Time (FDT), which measured the time required by a specific prioritized test suite (i.e., a solution given by the selected test case prioritization search algorithm) to detect

the seeded faults. To simulate the faults mutation testing was employed. The idea of mutation testing is to obtain an original program under test and create different version of this program (i.e., mutants) by adding small syntactic variations (i.e., artificial faults) [JH11, JJI+14]. We selected this technique since mutation testing has been demonstrated to be a valid substitute of real faults [JJI+14].

We selected the FDT because, in the context of this study, the Average Percentage of Faults Detected (APFD) might show some limitations. The first reason is that we are interested in timing performance, whereas APFD measures the relative position of the ordered test cases and the number of faults detected. This means that for reactive test cases, where the time variance is high, the APFD metric could give a good result if the first test case to be executed detects all the injected faults. However, the first test case could employ too much time to detect all the faults. In a contrary example, the faults could be found using four short test cases, and the total amount of time could be shorter than a large test case that finds all the injected faults. Moreover, in large test suites the APFD measurement might not be meaningful to compare prioritization effectiveness and it cannot measure activities such as automatic fault localization or fault severity [WSKR06].

### Average Percentage of Faults Detected (APFD)

Despite it having some disadvantages in the context of this study, to measure test case prioritization approaches and criteria, many approaches are evaluated using the APFD metric, which measures the weighted average of the percentage of faults detected over the life of the suite [CM13]. Since the APFD metrics is typically used to measure the performance of test case prioritization techniques, we decided to include it in our evaluation. To measure the APFD, let T be a test suite containing n test cases and F be a set of m faults detected by T. Let $TF_i$ be the first test case in an ordering $\pi$ of T that reveals fault $i$. Given an ordering $\pi$ of the test suite T, the APFD metric is defined as expressed in Equation 8.12.

$$APFD(\pi) = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \cdot m} + \frac{1}{2n} \tag{8.12}$$

### Simulation Time

As explained before, the overall simulation time can be reduced by reducing as much as possible the initialization time of each test case. We defined the simulation time metric, which measured the time required by each of the solutions provided by the search algorithm to execute the whole test suite.

#### Requirements Covering Time (RCT)

The Requirements Covering Time (RCT) takes the time needed by a specific solution to cover all the requirements of the configuration being tested into account. We employed the RCT for both, functional requirements (at the MiL, SiL and HiL test levels) as well as for non-functional requirements (at the HiL test level). The metric was coined as FRCT for functional requirements coverage and as NFRCT for non-functional requirements coverage.

### 8.3.3 Selected Algorithms

Two local search algorithms were selected as representative for local search algorithms (i.e., the Greedy algorithm and the AVM algorithm). Regarding global search algorithms, other two algorithms were selected based on their good performance in similar studies (e.g., [WAG13]) (i.e., the Weight-Based Genetic Algorithm (WBGA) and the Randomly-Weighted Genetic Algorithm (RWGA)). The assigned weights for Greedy, AVM and WBGA where the same for all objectives. On the other hand, notice that RWGA assigns weights randomly in each iteration (following the weight-based theory). In addition, notice that for the MiL and SiL test levels, the weight assigned to the non-functional requirements covering time objective (NFRCT) was 0, since at these levels, non-functional requirements are not tested.

### 8.3.4 Algorithms Configurations

We selected the same weight for each of the objectives, i.e., the weight for all the four objectives of the MiL and SiL test levels were set as 0.25 and the five weights for the five objectives of the HiL test level was set as 0.2. The population size for the genetic algorithms was 100 and the number of fitness evaluations was set as 50,000 (i.e., we obtain the optimal solutions after 50,000th fitness evaluations). We used a standard one-point crossover with a rate of 0.8 and the mutation of a variable was done with the standard probability 1/n, where n is the number of variables (i.e., in our context, number of test cases). We used this setup based on other studies and recommendations related to search-based software testing [AB11, WAG13].

### 8.3.5 Artificial Problems and Experiment Runs

We divided the evaluation of the approach in different experimental scenarios, named as artificial problems. For each case study, each artificial problem involved (1) a test suite composed of a certain number of test cases, (2) a specific product configuration, (3) 50 algorithm runs to obtain 50 solutions and (4) evaluation of each of the solutions

using the CPS case study model and 20 mutants. As recommended by Arcuri and Briand [AB11], we ran each algorithm 50 times to account for random variation produced by these algorithms. We employed five different product configurations for the first three case studies and four product configurations for the DC engine case study. We set the first test suite size to 30 test cases, and increment by 10 test cases when all the algorithm runs were performed, until a test suite of 120 test cases. In total, for the first three case studies, 50 independent artificial problems were employed, while for the last case study 40. The algorithm below shows the pseudocode of the developed script for executing the experiment.

```
1  for Each Case Study do
2      for Each Search Algorithm do
3          Test Suite Size = 30 test cases;
4          for Each Configuration of Case Study do
5              for Each test suite do
6                  Run search algorithm certain number of times
7                  for Each run do
8                      Evaluate solution using the CPS model;
9                      Calculate values of each evaluation metric;
10                 end
11                 Test suite size = Test suite size + 10 test cases;
12             end
13         end
14     end
15 end
```

**Algorithm 5:** Pseudocode of the performed experiment

On the one hand, the FDC of the test cases related to the ACC and Tank case studies was obtained based on historical data of previously executed configurations. For the Adaptive Cruise Control (ACC) case study, the historical data was obtained from 47 previously tested configurations, that seeded a total amount of 12 faults. For the tank case study, the historical data was obtained from 16 previously tested configurations, that in total seeded 8 faults. On the other hand, the FDC of the Unmanned Aerial Vehicle (UAV) and the DC engine case studies was estimated using the mutation testing technique. Specifically, twenty mutants were employed and we measured whether each test case was capable of detecting each mutant or not. This was performed this way because we did not have information from previous executions.

To evaluate our approach, mutation testing was used, i.e., a set of faults were injected in the Simulink models of the case studies [JH11, JJI+14].[1] In order to

[1] Notice that the mutants employed for the evaluation were different to the mutants employed for

study the performance of the algorithms with different number and types of faults, we manually injected 20 faults in each of the case studies. The distribution of the faults was 50 % in the physical layer (i.e., sensors, actuators, mechanical elements or communications) and 50 % in the cyber layer (i.e, software system). Since one of the threats to validity of mutation testing is the subsumed mutants [PHH+16], we employed different mutation operators in different parts of the system. The injected mutations were typical CPSs faults: faults in sensors (such as stuck-at or noise), faults in actuators (communication delays with the cyber layer), or in the software system. For the software system we mutated some Simulink subsystems following the patterns described by Matinnejad [MNBB16], in which the most common Simulink faults were identified based on discussions with engineers from the automotive industry, as well as the mutation operators proposed for Simulink models [HBT16]. In addition to the statistical tests, the average improvement of the best algorithm with respect to the remaining algorithms was calculated and reported in Table 8.3.

### 8.3.6 Statistical Tests

We employed a rigorous statistical analysis to statistically assess the performance of the proposed algorithms. To answer RQ 1 and RQ 2 we compared the algorithms with RS and each pair of them with the Mann-Whitney U test based on the result of the FDT for each given solution for each artificial problem. Notice that for the Tables showing the statistical summary involving RQ1 and RQ2 (e.g., Table A.1), for each of the test levels, the column indicating "+" means the number of artificial problems where the algorithm in the column A significantly outperformed the algorithm B (i.e., for all metrics except the APFD $\hat{A}_{12} < 0.5$ and p-value < 0.05, and for the APFD metric, $\hat{A}_{12} > 0.5$ and p-value < 0.05). The column indicating "-" means the opposite. The column "=" indicates the number of artificial problems where there was no statistical significance between both algorithms in terms of the FDT (p-value > 0.05).

The RQ 3 analyzes the performance of the algorithms as the amount of test cases in the test suite increases. To answer RQ 3, Spearman's rank correlation ($\rho$) has been applied, which measures the correlation of the evaluation metric with respect to the number of test cases. The test returns a $\rho$ value within the range [-1,1] and a p-value. For all the metrics with the exception of the APFD, a negative $\rho$ value indicates that the performance increases as the test suite size increases, while a positive $\rho$ indicates the opposite. For the APFD, a positive $\rho$ value indicates that the performance increases, whereas a negative $\rho$ indicates the opposite. Finally, the p-value indicates whether there is statistical significance in the correlation OR NOT.

estimating the FDCs of the test cases

## 8.4 Results and Analysis

In this section we analyze the results of the empirical evaluation for each of the evaluation metrics.

### 8.4.1 Fault Detection Time (FDT)

The FDT measures the time required by each of the prioritization techniques to kill the 20 mutants. While Figure 8.3 shows the distribution of the FDT results for all the artificial problems, Table A.1 reports the statistical analysis performed for each case study in terms of the FDT for the three test levels (i.e., MiL, SiL and HiL) and RQs 1 and 2. RQ1 aims to answer whether the problem to solve was non-trivial by comparing each of the selected algorithms with RS. In general, all the algorithms outperformed RS for most of the artificial problems. Taking all the case studies as well as all the test levels into account, Greedy outperformed RS with statistical significance in 306 out of 570 artificial problems, AVM in 371 out of 570 artificial problems, WBGA in 318 out of 570 artificial problems and RWGA in 327 out of 570 artificial problems.

Regarding RQ2, where the performance among the rest of algorithms was compared, results were not consistent for each of the case studies. The AVM algorithm was the best one for the ACC and the DC engine case study, whereas Greedy performed best in the UAV and the Tank case studies. This suggests that in terms of the FDT, local search algorithms perform better than global search algorithms. However, while AVM outperformed the rest of the algorithms for both the UAV and the tank case studies, both WBGA and RWGA outperformed Greedy in the ACC and the DC engine case studies. Moreover, the difference between the AVM and Greedy for the UAV and the tank case studies were not very high (on average, as reported in Table 8.3, around 16.6% and 22.71%), whereas for the ACC and the DC engine case studies AVM reduced the FDT as compared with Greedy on average in 49.78% and 47.84%.

RQ3 aimed at assessing the scalability of the approach. To this end, we applied the Spearman's rank correlation. The summary of the results for the Spearman's rank correlation with respect to each case study product is shown in Table A.2. All the selected algorithms showed improvement in terms of the FDT with the test suite size increase for the tank case study. For the rest of case studies, in the case of RS as well as the global search algorithms (i.e., WBGA and RWGA) in most of the products their performance decreased with the test suite increase. Conversely, the performance of the AVM increased in most of the cases statistically significantly (i.e., $\rho < 0$ and p-value $< 0.05$), except for the DC engine case study. However, even if for the DC engine case

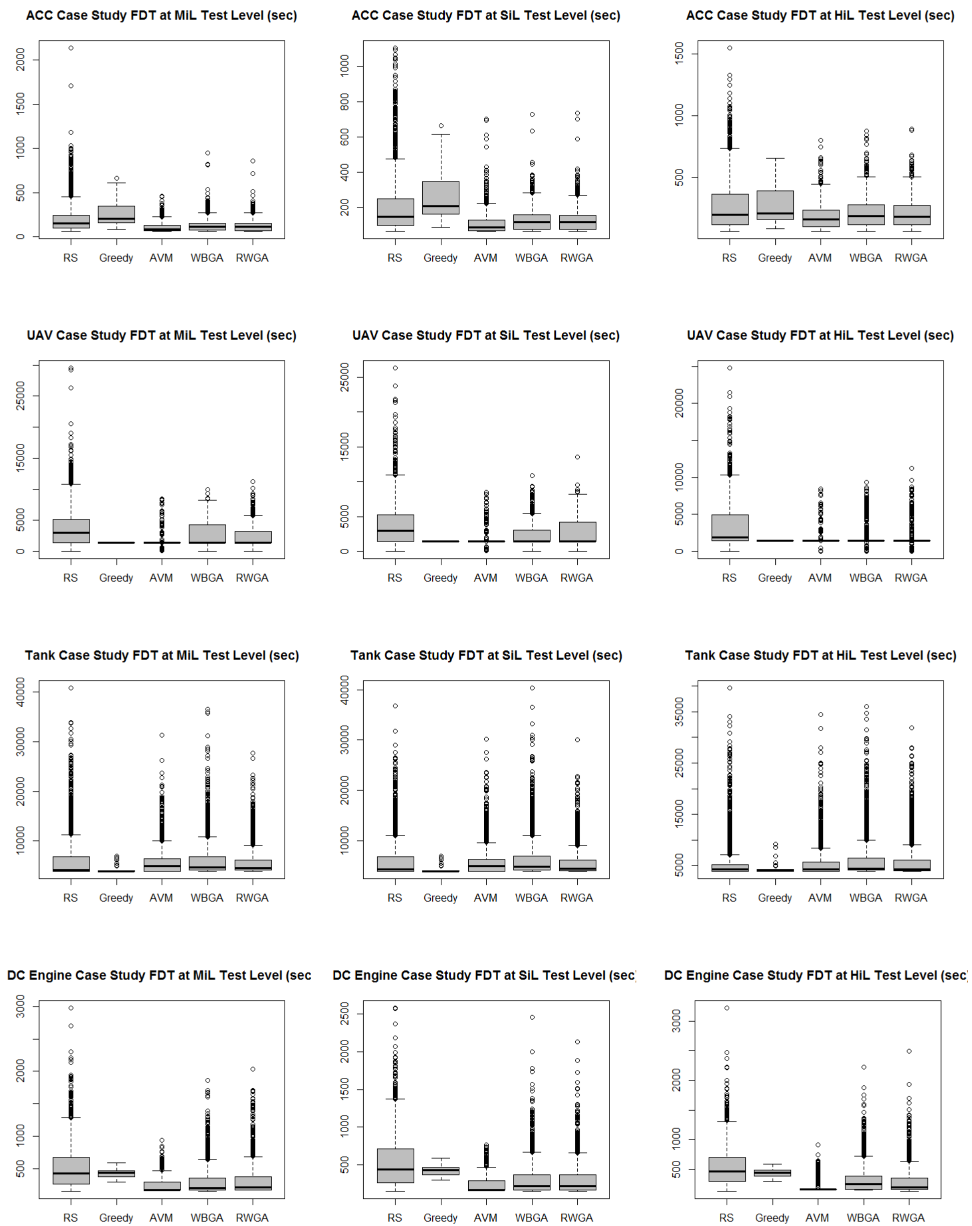



Figure 8.3: Distribution of the FDT for all the artificial problems of each case study

Table 8.3: Percentage of improvement by the best algorithm for each case study with respect to the remaining algorithms

| | Case Study | Best Alg. | RS | Greedy | AVM | WBGA | RWGA | Average |
|---|---|---|---|---|---|---|---|---|
| **FDT** | ACC | AVM | 41.85 | 49.78 | - | 14.39 | 12.20 | 29.56 |
| | UAV | Greedy | 60.66 | - | 16.60 | 39.54 | 38.56 | 38.84 |
| | TANK | Greedy | 25.96 | - | 22.71 | 28.49 | 23.67 | 25.21 |
| | DC Eng | AVM | 58.04 | 47.84 | - | 28.53 | 27.03 | 40.36 |
| **APFD** | ACC | AVM | 1.01 | 11.80 | - | 0.60 | 0.33 | 3.44 |
| | UAV | WBGA | 0.44 | 0.16 | 0.08 | - | 0.02 | 0.17 |
| | TANK | AVM | 0.09 | 1.51 | - | 0.44 | 0.24 | 0.57 |
| | DC Eng | WBGA | 2.40 | 53.63 | 0.04 | - | 0.02 | 14.02 |
| **SIM. TIME** | ACC | AVM | 25.51 | 5.97 | - | 8.51 | 10.95 | 12.74 |
| | UAV | Greedy | 24.02 | - | 5.64 | 17.69 | 16.61 | 15.99 |
| | TANK | Greedy | 37.74 | - | 0.25 | 15.04 | 15.37 | 17.10 |
| | DC Eng | AVM | 2.83 | 1.76 | - | 0.11 | 0.12 | 1.20 |
| **FRCT** | ACC | RS | - | 10.06 | 13.59 | 12.32 | 11.87 | 11.96 |
| | UAV | RS | - | 4.34 | 8.76 | 9.11 | 17.83 | 10.01 |
| | TANK | Greedy | 17.92 | - | 20.12 | 16.34 | 16.28 | 17.66 |
| | DC Eng | Greedy | 84.32 | - | 91.07 | 87.90 | 87.90 | 87.80 |
| **NFRCT** | ACC | Greedy | 58.32 | - | 2.73 | 4.31 | 6.41 | 17.94 |
| | UAV | Greedy | 29.99 | - | 0.08 | 0.35 | 4.56 | 8.75 |
| | TANK | Greedy | 14.99 | - | 17.21 | 17.91 | 14.71 | 16.20 |
| | DC Eng | Greedy | 86.31 | - | 91.61 | 88.49 | 88.71 | 88.78 |

study in most of the products the performance of the AVM algorithm did not increase, its performance did not decrease with statistical significance (i.e., p-value $>$ 0.05).

### 8.4.2 Average Percentage of Faults Detected (APFD)

The APFD measures the position of the test cases with respect to the detected faults, which are in this case mutants. The obtained APFD distributions are depicted in Figure 8.4. For each of the case studies we report the statistical tests summary in Table A.3, where the number of artificial problems in which each pair of algorithms outperform each other are reported. RQ1 evaluates whether the problem to solve is non-trivial by comparing each of the selected algorithms with RS. In general, the selected algorithms outperformed RS, although there are some exceptions. In this case, considering the statistical tests, RS outperformed Greedy for most of the artificial problems (i.e., 420 out of 570 artificial problems). However, the remaining algorithms outperformed RS in terms of the APFD metric. Specifically, AVM outperformed RS in 336 out of 570 artificial problems, WBGA outperformed RS in 280 out of 570 artificial problems and RWGA in 294 out of 570 artificial problems.

Regarding RQ2, in terms of the APFD metric, AVM outperformed the rest of algorithms in 3 out of 4 case studies (i.e., ACC, Tank and DC engine case studies). As for the UAV case study, unlike in terms of the FDT, both global search algorithms (i.e., WBGA and RWGA) outperformed local search algorithms for the APFD metric. This might mean that WBGA and RWGA prioritized those test cases with high FDC

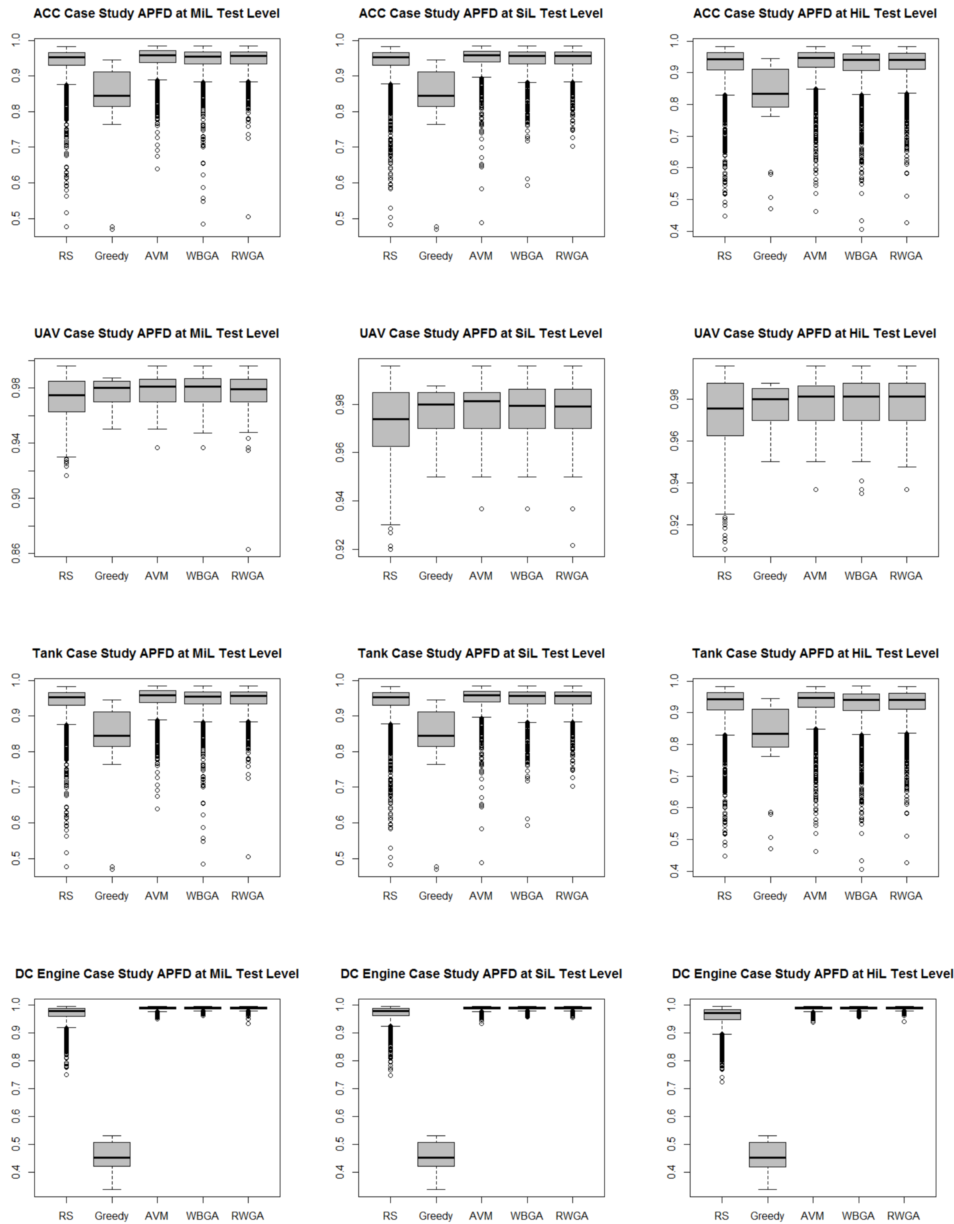



Figure 8.4: Distribution of the APFD for all the artificial problems of each case study

but giving lower importance to their test execution time. However, as shown in Figure 8.4 and the reported average improvement values in Table 8.3, the differences between AVM and both global search algorithm is not high. It is noteworthy that these algorithms outperformed Greedy for the remaining three case studies in terms of the APFD.

As for RQ3, where the scalability of the approach is assessed, we applied the Spearman's rank correlation test. In this case, since we were aiming to increase the APFD, a positive $\rho$ indicated an improvement of the algorithm with the increase of the test suite size. The obtained results for the Spearman's rank correlation test are reported in Table A.4. As it can be appreciated, for all the case studies and products, the performance of all the algorithms improved with the test suite size in terms of the APFD metric with statistical significance. This means that the algorithms are scalable in terms of the APFD.

### 8.4.3 Simulation Time

Figure 8.5 depicts the obtained normalized results for the simulation time metric. Notice that these values are normalized for representation because a larger test suite implies a larger simulation time. The values are normalized by dividing the obtained simulation time with the number of test cases in the test suite. In addition, Table A.5 summarizes the results of the statistical analysis performed for each case study for the simulation time metric. As for RQ1, where the selected algorithms are compared with RS, as it can be shown, all the algorithms outperformed RS for all the case studies at all the three test levels for every single artificial problem with the exception of Greedy. This last algorithm performed similar to RS in terms of the simulation time for the DC engine case study. However, for the rest of case studies, Greedy outperformed RS. On average, from Table 8.3, it can be seen that the selected algorithms outperformed RS up to 37.74 % in terms of the simulation time.

RQ2 aims at answering which of the algorithms showed better performance. For the case of simulation time, local search algorithms performed better than global search algorithms. As shown in Table 8.3, the average improvements between both local search algorithms is not very high (the higher average improvement was of 5.97% for the AVM with respect to Greedy for the ACC case study). Taking the statistical analysis into account, which is shown in Table A.5, the AVM algorithm performed the best in the ACC, tank and the DC engine case studies, whereas Greedy performed best in the UAV case study. In this last case study, AVM also outperformed both global search algorithms (i.e., RWGA and WBGA). Moreover, for those cases where AVM fared best, Greedy also outperformed both global search algorithms for the ACC as well as the tank case studies, although during the DC engine case study both global search algorithms outperformed Greedy.

Notice that for this metric we did not address RQ3, since a larger test suite always implies a larger test execution time.

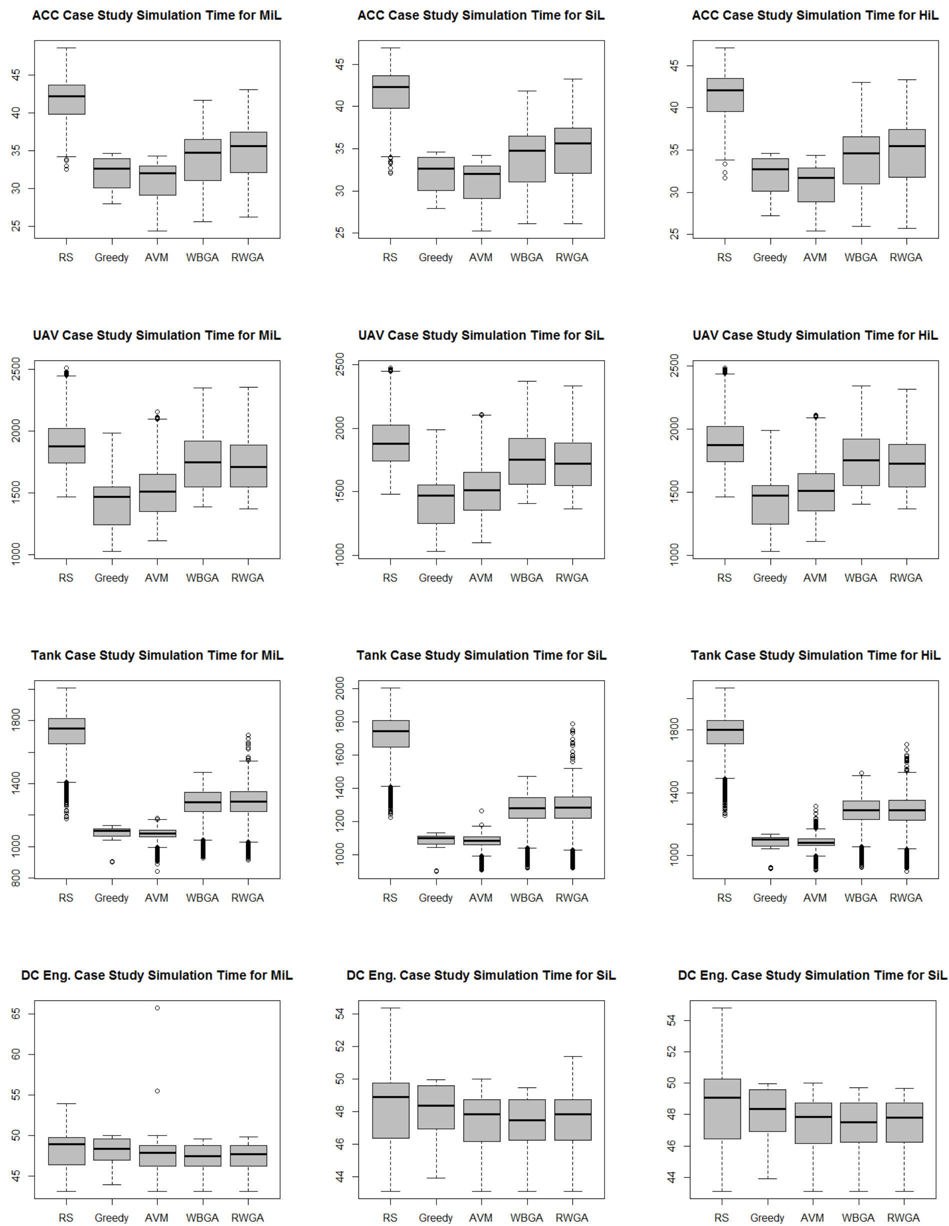



Figure 8.5: Distribution of the normalized simulation time (i.e., average simulation time per test case) for all the artificial problems of each of the case studies

### 8.4.4 Functional Requirements Covering Time (FRCT)

Figure 8.6 depicts the distribution of the FRCT for all the artificial problems of each case study, while Table A.6 reports the summary for the statistical results when compared each pair of the selected algorithms. RQ1 aims to answer whether the problem to solve was non-trivial by comparing the selected algorithms with RS. As for the statistical analysis reported in Table A.6, in general, except for Greedy, the selected algorithms were outperformed by RS for most of the artificial problems, with the exception of the tank case study. This means that the problem of reducing the time for testing functional requirements is trivial in this context. However, it is positive that still Greedy performs better than RS.

As for RQ2, Greedy performed best in most of the case studies for most of the test levels, except for the ACC case study and the MiL and SiL test levels. It is noteworthy the average improvement of the FRCT of Greedy with respect to the rest of algorithms for the DC engine case study, which on average, it improved the rest of algorithms in around 87.80%, as reported in Table 8.3. In addition, unlike for previous evaluation metric, AVM performed worst when compared to the rest of algorithms. This might be because AVM gives more importance to the rest of objectives. However, as shown in Figure 8.6 and reported in Table 8.3, the differences between AVM and both global search algorithms in terms of the FRCT is not very high.

RQ3 evaluates the scalability of the selected algorithms. Table A.7 reports the Speaman's rank correlation for the FRCT metric with respect to the test suite size. Results vary for each algorithm depending on the case study as well as the product. For instance in the ACC case study, the Spearman's rank correlation $\rho$ showed a negative correlation with statistical significance for all the algorithms in both p4 and p5 products. It is noteworthy that since these two products were more complex than the rest, their number of functional requirements was higher, as shown in Table 8.2. For the rest of case studies, in general Greedy showed a negative $\rho$ for most of the products, which means that its performance increases when the test suite size increases. In addition, except for the DC engine case study, the AVM algorithm also showed negative correlation in most of the cases, except for non-complex products (typically p1). As for the global search algorithms, in general their performance decreased with the test suite size in most of the products and case studies, with the exception of the ACC case study.

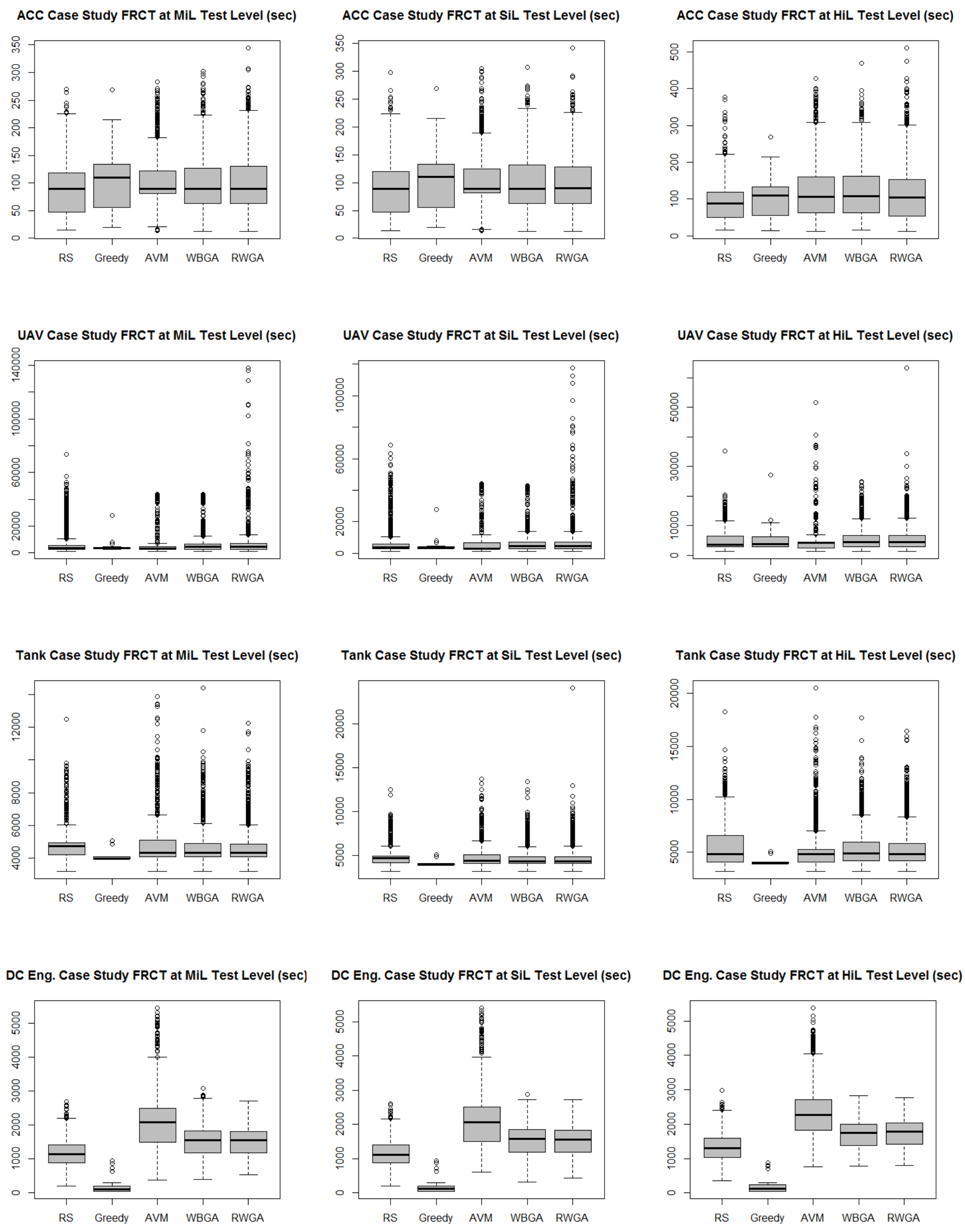



Figure 8.6: Distribution of the FRCT for all the artificial problems of each case study

### 8.4.5 Non-Functional Requirements Covering Time (NFRCT)

Figure 8.7 depicts the distribution of the results for the NFRCT and Table A.8 reports the summary for the statistical results when compared each pair of the selected algorithms. The first RQ compares the selected algorithms with RS. For the ACC and Tank case studies, the selected algorithms outperformed RS. For the UAV case study, AVM showed similar results as RS, whereas the rest of algorithms showed worst results than RS. Conversely, Greedy outperformed RS in the DC engine case study, while the rest of algorithms showed worse performance than RS. On average, Greedy was the algorithm showing highest average improvement percentage when compared with RS, as shown in Table 8.3, improving the RS algorithm on average between 14.99% and 86.31%.

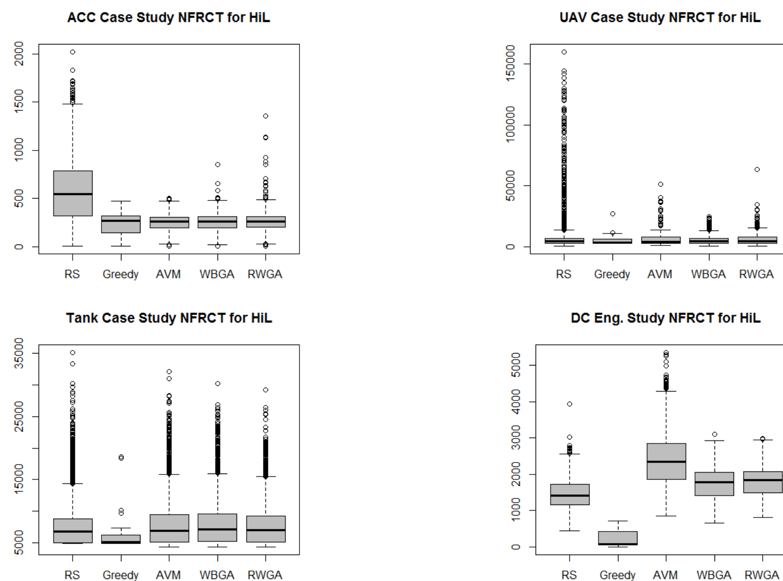



Figure 8.7: Distribution of the NFRCT for all the artificial problems of each case study

The second RQ evaluates which of the selected algorithms fared best. According to the statistical analysis, which is summarized in Table A.8, in this case, results vary depending on the case study. Greedy showed the best performance for the ACC and the DC engine case studies. AVM outperformed the rest of algorithms in the UAV case study. Finally, RWGA performed better than the rest in the Tank case study. In addition, from Table 8.3, it can be appreciated that as for the average percentage of the improvement, Greedy was the algorithm showing the best performance in terms of the NFRCT metric. In fact, it is important to highlight the results for the DC engine case study, where it improved the rest of the algorithms in around 88.78%.

RQ3 evaluates the scalability of the approach. Table A.9 reports the results of the Spearman's rank correlation for the NFRCT metric based on the test suite size. On the one hand, except for the tank case study, RS as well as both global search algorithms (i.e., WBGA and RWGA), showed a performance decrement as the test suite size increases. On the other hand, Greedy improved with the test suite size for all products of the tank and DC Engine case studies, as well as in most of the products of the ACC and UAV case studies. Finally, the AVM algorithm increased its performance with the test suite size in all products of the ACC and tank case studies, as well as in most of the UAV case study's products. However, in the DC engine case study the performance of the AVM algorithm decreased as the test suite size increased.

## 8.5 Discussion of the Results

We summarize the results of the empirical evaluation and what they tell us about the research questions.

### 8.5.1 Answer to RQ1

The first RQ refers to the comparison of the proposed algorithms with RS to assess that the problem to solve is not trivial. RS has been selected as the comparison baseline for several similar studies (e.g., [WAG15, WAG13, WBA$^+$14, AWSE16a, PSS$^+$16]). Generally in RQ 1, for most of the metrics, the selected algorithms outperformed RS. The most striking exceptions were for the Functional Requirements Covering Time in the ACC and UAV case studies. In these cases, RS outperformed the rest of the algorithms. The reasons for this might be that either the rest of algorithms focused on other objectives, or for these case studies there were not a lot of functional requirements and thus, the problem to solve was non-trivial. Moreover, for the remaining evaluation metrics in these case studies, the rest of the algorithms outperformed RS. Thus, we can answer the first RQ as follows:

*Based on the experimental results and the statistical tests of our study we can conclude that the test case prioritization for simulation-based testing of CPS product lines is a non-trivial problem and thus, search algorithms are recommended to be used.*

### 8.5.2 Answer to RQ2

RQ 2 aimed at answering which of the selected search algorithms was the most appropriate to solve the test case prioritization problem for configurable CPSs. Generally in

RQ 2, the selected local search algorithms (i.e., AVM and Greedy) outperformed the selected global search algorithms (i.e., WBGA and RWGA). As for the fault revealing capability, which was measured by the FDT and APFD metrics, AVM outperformed the rest of the algorithms in two case studies using the FDT and in three case studies using the APFD metric, whereas Greedy outperformed the rest of algorithms in two out of four case studies using the FDT. For the APFD metric, WBGA outperformed the rest of metrics in one case study. As for the simulation time, the AVM algorithm outperformed the rest of the algorithms in three out of four case studies, whereas Greedy was the best algorithm for the simulation time in the UAV case study. As for the RCT metrics, for functional requirements Greedy outperformed the rest for the DC engine as well as the tank case studies. Regarding non-functional case studies, Greedy outperformed the rest of metrics for the ACC and the DC engine case studies, whereas the AVM algorithm outperformed the rest for the UAV case study. Finally, RWGA outperformed the rest for the tank case study.

The reason that local search algorithms (i.e., AVM and Greedy) outperformed global search algorithms (e.g., WBGA) can be explained that optimal solutions of our problem may exist in a local search space and thereby local search algorithms are more efficient as compared with global search algorithms in terms of finding optimal solutions within a limited number of fitness evaluations (e.g., 50,000 in our case). However, when the number of evaluating fitness is increased, the performance of global search algorithms may be improved. Nevertheless, it is worth mentioning that the time to run the global search algorithms will also be increased with the growth of fitness evaluations.

These results are consistent with a previous study performed in the context of this thesis [AWSE16b]. In the conference version, only two objectives were taken into account: (1) the test execution time and (2) the FDC. We measured the performance of the algorithms with the FDT metric, which is also used in this chapter. Results of our conference version paper showed that local search algorithms were better than global search ones. Specifically, Greedy performed best with shorter test suites while AVM performed best with longer test suites [AWSE16b]. Thus, from the results of our experiment in this chapter, and also supported by the results published in our previous work [AWSE16b], we can answer the second RQ as follows:

*Results can vary depending on different case studies. However, local search algorithms performed better than global search algorithms. Generally, for faster fault detection and for simulation time reduction, AVM performed better, whereas for faster requirements testing Greedy outperformed the rest.*

### 8.5.3 Answer to RQ3

The last RQ aimed at assessing the scalability of the proposed search algorithms. To answer this RQ we applied the Spearman's rank correlation test, where the correlation between the selected evaluation metrics with respect to the number of test cases in the test suite was measured. Results varied depending on the case study and algorithms. From the previous RQs, we extracted the best algorithms for solving the test case prioritization problem for our context. On the one hand, for the FDT metric, the AVM algorithm showed the best performance. For this metric, the performance of the algorithm increased with the test suite size for 44 out of 57 cases. Moreover, for the rest of the cases, the performance of the AVM algorithm did not decrease with statistical significance (i.e., p-value $> 0.05$). On the other hand, for the Requirements Covering Time, both functional and non-functional, Greedy showed the best performance. As for the FRCT metric, Greedy showed a performance increase in 43 out of 57 cases. Regarding the NFRCT metric, the performance of the Greedy algorithm increased in 16 out of 19 cases as the test suite size increased. This means that the selected algorithms are scalable for their purpose. From the results of our experiments, we can answer the RQ 3 as follows:

*The increment of test cases showed a positive impact on the performance of the AVM algorithm for solving the test case prioritization approach when finding faults as quick as possible (FDT and APFD metrics). In addition, the increment of test cases showed a positive impact on the performance of the Greedy algorithm for covering Functional and Non-Functional Requirements (FRCT and NFRCT metrics). Thus, we can conclude that the selected search algorithms along with our defined fitness function are scalable to address the test case prioritization problems with increasing complexity.*

## 8.6 Threats to Validity

We summarize the threats to validity of our empirical evaluation as follows:
**Internal validity:** *Internal validity* threats exist when the results can be influenced by internal factors (e.g., parameters of the algorithms) [ABHPW10]. An internal validity threat of our study could be that the configurations related to global search algorithms (e.g., population size, crossover rate, etc.) were not changed. However, the selected settings are in accordance with the common guidelines in the literature and other studies related to the application of search algorithms to testing [WAG15, AB11]. Another internal validity threat refers to the injected faults. In our study we have

employed 20 mutants, but there might be cases with more faults. Notice that in our context, each mutant includes not only the software, but also the physical layer of the CPS, what makes the execution of the simulation time consuming. In addition, some studies have discovered that the subsumed mutants are one of the threats of this technique to assess the quality of test suites [PHH+16]. We have tried to mitigate this threat by distributing different types of faults (i.e., different mutation operators) distributed across the whole system (i.e., the physical and cyber layer of the CPSs).

**Conclusion Validity:** A *conclusion validity* threat in most of the evaluations where search algorithms are employed involves the random variations of the results [WAG15]. To reduce this threat, we divided the experiment in different artificial problems and repeated the execution of each algorithm for each artificial problem 50 times to account for random variations. Moreover, we compared the results of the algorithms by applying statistical analysis.

**Construct Validity:** The *construct validity* threat is that the measures used are not comparable across the algorithms. However, we used the same stopping criterion for all the algorithms, i.e., the number of fitness evaluations (50,000 times), which is a comparable measure across all the algorithms.

**External Validity:** External validity threats appear when the outcome of results are influenced by external factors [ABHPW10]. An *external validity* threat with any experiment is related to the generalization of results. We used four case studies, which might not be enough to ensure that our results can be extrapolated to all CPS product lines. However, to reduce this threat we used four case studies from different domains with different sizes and different characteristics to assure a sufficient degree of heterogeneity. In addition, one of the case studies was a real-world industrial case study.

## 8.7 Related Work

Many studies in product line engineering testing have addressed the test case prioritization problem. Some of them compared different prioritization criteria [SSRC14a, SSPRC15]. Others employed search-based algorithms [PSS+16, HPLT14] or statistical testing [DPC+14, DPC+15]. However, notice that their prioritization approach is at the domain engineering level. In their context, a test case is a valid product of the product line. Conversely, in our case, we focus at the application engineering level, where a test case is a set of signals stimulating the Cyber-Physical System Under Test (CPSUT).

An approaches focused on test case prioritization at the application engineering

level for product lines. Specifically, Wang et al. proposed a multi-objective approach for test case prioritization of Software Product Lines (SPLs), employing three different weigh-based search algorithms. Our approach builds upon this works by: (1) applying it in the configurable CPS engineering context, which faces several particularities such as the multi-level simulation-based testing (MiL, SiL and HiL test levels), reduction of simulation time by reducing the initialization time of test cases based on the test prioritization or testing functional and non-functional requirements, and (3) evaluating the performance of several algorithms using simulation and mutation testing.

In section 3.3.2 different search-based approaches for test case prioritization were highlighted (e.g., [MRE02, WSKR06, ZHG+09, HFM15]). When compared to the current test case prioritization techniques, our approach faces some differences. Notice that we aim at prioritizing test cases for CPSs employing simulation, which requires multi-level testing (i.e., MiL, SiL and HiL test levels), and each of the test levels have their corresponding test objectives (e.g., the HiL test level includes non-functional requirements coverage). The test cases employed in this study are designed for testing configurable CPS products at the system engineering level. These test cases take from seconds to minutes to be executed, unlike unit testing, where the test execution time of each test case is in the order of some milliseconds [AIB10]. In addition, the types of test cases we use are reactive test cases, which have some particularities, such as the varying test execution time depending on the previously prioritized test case. This is something that, to the best of our knowledge, is not considered in other test case prioritization methods; this allows for the reduction of the overall simulation time. Moreover, we evaluated the performance of the selected algorithms using simulation and mutation testing. Notice that the use of mutation testing in this context is especially challenging and expensive due to the fact that the physical layer, which is typically modeled with complex mathematical models, has to be simulated in each generated mutant, which makes the simulation time very long. Lastly, we integrated the test case prioritization approach with FeatureIDE [TKB+14], so that when a specific product configuration is selected to test, its requirements are automatically selected. This allows for a systematic and fully automated test prioritization for configurable CPSs.

## 8.8 Conclusion and Future Work

This chapter proposes a search-based test case prioritization approach that optimizes the process of testing configurable CPSs by reducing the fault detection time, simulation time and requirements covering time. To this end, we defined a fitness function that employed five objectives. Four different case studies were used to compare the

performance of different algorithms. Based on the results of empirical evaluation, we observed that the selected four search algorithms outperformed RS. Moreover, we discovered that the local search algorithms (i.e., AVM and Greedy) performed better than the global search algorithms (WBGA and RWGA). Specifically, the AVM algorithm showed the best performance for fault detection and reduction of simulation time, whereas the Greedy algorithm performed better to reduce the requirements covering time. The proposed search-based test case prioritization approach has been integrated within our framework developed in this dissertation for testing configurable CPSs employing simulation-based testing.

The empirical evaluation of Pareto-based multi-objective search algorithms remains as future work. We already have launched some preliminary experiments of two well known multi-objective search algorithms (Non-dominated Sorting Genetic Algorithm II (NSGA-II) and Strength Pareto Evolutionary Algorithm 2 (SPEA2)). However, Pareto-based multi-objective search algorithms return a set of solutions and evaluating them with mutation testing is extremely expensive. In the future, we foresee to perform the evaluation of Pareto-based multi-objective algorithms with several clusters.

# Part IV

# Debugging

Chapter 9

# Debugging Product Lines

Configurable systems testing is challenging mainly due to the potentially huge number of products under test. Most of the research on this field focuses on making testing affordable by selecting a representative subset of products to be tested. However, once the tests are executed and some failures revealed, debugging is a cumbersome and time consuming task due to difficulty to localize and isolate the faulty features in the configurable system. In this chapter, we propose a debugging approach for the localization of bugs in configurable systems. The proposed approach works in two steps. First, the features of the configurable system are ranked according to their *suspiciousness* (i.e., likelihood of being faulty) using spectrum-based localization techniques. Then, a novel fault isolation approach is used to generate valid products of minimum size containing the most suspicious features, helping to isolate the cause of failures.

For the evaluation of our approach, we compared ten suspiciousness techniques on nine Software Product Lines (SPLs) of different sizes. The results reveal that three of the techniques (Tarantula, Kulcynski and Ample2) stand out over the rest, showing a stable performance with different types of faults and product suite sizes. By using these metrics, faults were localized by examining between 0.1% and 14.4% of the feature sets. Our results show that the proposed approach is effective at locating bugs in configurable systems, serving as a helpful complement for the numerous approaches for testing SPLs.

## 9.1 Introduction

Developing high-quality software requires not only effective testing methods to uncover failures, but also debugging techniques to locate and fix the bugs that trigger them. Debugging is mostly a manual process where testers must identify the defective code using techniques such as tracing, memory dumps or step-by-step execution. More sophisticated techniques include Spectrum-Based Fault Localization (SBFL), which

ranks code components (e.g., statements) according to their probability of having faults, so-called *suspiciousness* [AZGvG09, HRS+00, XCKX13, WGL+16].

Debugging configurable systems, such as SPLs or configurable Cyber-Physical Systems (CPSs), is challenging due to the difficulty to find and isolate the faulty features in the configurable system. Also, even if a suspicious feature or set of features are detected, it might still be difficult to generate small valid products (i.e., satisfying the constraints of the feature model) where the failure is reproduced and the defective assets can be pinpointed. In this context, the recent advances on SPL testing contrast with the poor support for SPL debugging, which remains as a manual and time-consuming endeavor.

In this Chapter, we propose an approach to configurable systems debugging. The approach works in two steps. First, the outcomes of testing (test coverage and test outputs) are used to rank features according to their probability of having faults, so-called *suspiciousness score*. The suspiciousness score of each feature (or set of features) is calculated using SBFL techniques adapted for the SPL domain. Then, a fault isolation approach is proposed to generate, by automatically analyzing the feature model, products of minimum size containing the most suspicious features, in order to facilitate the isolation of the failure causes. The proposed method mainly works at the feature model level, which abstracts the complexity of the underlying implementations such as the use of different programming languages or the combination of hardware and software features, e.g., CPSs [ASEZ17]. For the evaluation of the approach, we compared ten state-of-the-art SBFL techniques on nine SPLs of different sizes with simulated faults. Results reveal that SBFL performs well at locating faults in SPLs. More specifically, we found that three of the techniques under evaluation (Kulcynski2, Tarantula and Ample2) stand out over the rest, being able to localize the bugs by examining between 0.1% and 14.4% of the feature sets.

This chapter is structured as follows. Section 9.2 presents our approach for fault localization in highly configurable systems (e.g., SPLs) and the fault isolation algorithm. An empirical evaluation of our approach is performed in Section 9.3. Section 9.4 highlights the main issues that threatens our empirical evaluation. Section 9.5 positions our work with the current literature. Section 9.6 concludes the study and highlights future work.

## 9.2 Fault Localization Approach

In this section, we present a two-step approach for locating bugs in highly configurable systems. First, SBFL techniques are used to calculate the suspiciousness of each

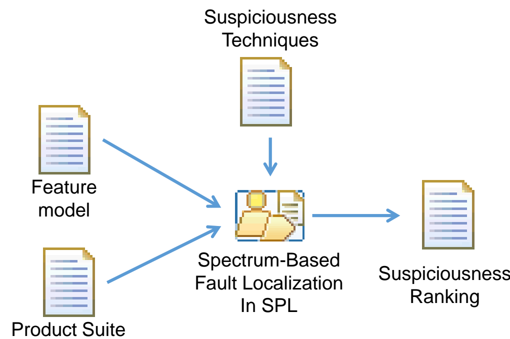



Figure 9.1: Overview of our approach on SBFL for highly configurable systems

feature set based on the testing outcomes, namely code coverage data and testing results (passes and failures). Second, the obtained suspiciousness scores are processed by a novel fault isolation approach to generate the smallest valid product containing the faulty feature set, helping to isolate the cause of the failure, and thus the bug causing it.

We may recall that this contribution focuses on debugging and not testing. Thus, we assume the existence of a product suite (e.g., pairwise suite) and their corresponding testing results, obtained using any state-of-the-art testing technique, (e.g., methods presented in previous chapters). In what follows, our approach is described in detail, including the overall methodology for its application.

### 9.2.1 Spectrum-based fault localization in SPLs

We propose to adapt SBFL techniques to measure the suspiciousness score of each feature set in a SPL. Figure 9.1 depicts the overview of the approach from a black-box perspective. Our approach receives a Feature Model and a product suite as inputs, and it returns a ranking of all the feature sets in the configurable system, ordered by their suspiciousness value in descendent order, according to a given suspiciousness technique, e.g., Tarantula. The process to calculate the suspiciousness scores and to break ties in the final ranking is detailed next.

#### Constructing the coverage matrix and error vector

Based on the SBFL theory (explained in Section 3.4.2), we consider the SPL products under test as the test cases, and the feature sets as the components where faults must be located. As an example, consider the feature model in Figure 2.2 and the product suite in Table 2.1. Table 9.1 depicts the coverage matrix, where the products under test are placed in columns, and the feature sets are listed in rows (note that a bug is

simulated in the feature `MP3`). For the sake of simplicity, only feature sets composed of one or two features are considered, although the approach could be generalized to feature sets of any size. In the example, only some feature pairs are shown to keep this chapter at a reasonable size. For each product under test (i.e., $P_1$, $P_2$,..., $P_8$), a cell is marked with "•" if it contains the feature set of the row. Additionally, the final row depicts the error vector, that is, the test outcome of each product, either successful ("S") or failed ("F").

Based on the information collected in the coverage matrix and the error vector, the suspiciousness score of each feature set can be calculated using any of the state-of-the-art suspiciousness techniques proposed in the literature [XCKX13]. To this purpose, we propose a slight modification of the meaning of the classical notation used in SBFL formulas, where test cases are replaced by *products* and components are replaced by *feature sets*, namely:

| | |
|---|---|
| $N_{CF}$ | number of failed products that cover a feature set. |
| $N_{UF}$ | number of failed products that do not cover a feature set. |
| $N_{CS}$ | number of successful products that cover a feature set. |
| $N_{US}$ | number of successful products that do not cover a feature set. |
| $N_C$ | total number of products that cover a feature set. |
| $N_U$ | total number of products that do not cover a feature set. |
| $N_S$ | total number of successful products. |
| $N_F$ | total number of failed products. |

Table 9.1: An example showing the suspiciousness value computed using the Tarantula technique in the Mobile Phone SPL

| ID | Feature Set | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ | $P_8$ | $N_{CF}$ | $N_{CS}$ | $N_F$ | $N_S$ | Suspiciousness | Ranking |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $F_1$ | MobilePhone | • | • | • | • | • | • | • | • | 4 | 4 | 4 | 4 | 0.5 | 5 |
| $F_2$ | Screen | • | • | • | • | • | • | • | • | 4 | 4 | 4 | 4 | 0.5 | 5 |
| $F_3$ | Calls | • | • | • | • | • | • | • | • | 4 | 4 | 4 | 4 | 0.5 | 5 |
| $F_4$ | High resolution | • | | | • | • | | | • | 2 | 2 | 4 | 4 | 0.5 | 6 |
| $F_5$ | Basic | | | | | | • | • | | 1 | 1 | 4 | 4 | 0.5 | 6 |
| $F_6$ | Colour | | • | • | | | | | | 1 | 1 | 4 | 4 | 0.5 | 6 |
| $F_7$ | GPS | | | • | | • | | | | 1 | 1 | 4 | 4 | 0.5 | 6 |
| $F_8$ | Media | | • | | • | • | • | | • | 4 | 1 | 4 | 4 | 0.8 | 3 |
| $F_9$ | Camera | | | | • | • | | | • | 2 | 1 | 4 | 4 | 0.75 | 4 |
| $F_{10}$ | MP3 (**BUG**) | | • | | • | • | • | | | 4 | 0 | 4 | 4 | 1 | 1 |
| $F_{11}$ | GPS-Colour | | | • | | | | | | 0 | 1 | 4 | 4 | 0 | 7 |
| $F_{12}$ | MP3-Colour | | • | | | | | | | 1 | 0 | 4 | 4 | 1 | 2 |
| Execution results | | S | F | S | F | F | F | S | S | | | | | | |

Table 9.1 shows the values of $N_{CF}$, $N_{CS}$, $N_F$ and $N_S$ for each feature set. Based on this information, the suspiciousness of each feature set using *Tarantula* is

depicted in the column “Suspiciousness”, followed by the position of each feature set in the ranking. As illustrated, the feature sets `MP3` (where a fault was seeded) and `MP3-Colour` are placed at the top of the ranking, followed by `Media` and `Camera`, with a suspiciousness score of 0.8 and 0.75 respectively. The rest of single features have a suspiciousness score of 0.5 according to Tarantula. Finally, the feature set `GPS-Colour` has a suspiciousness score of 0.

#### Breaking ties

The last column in Table 9.1 indicates the suspiciousness ranking of each feature set. As illustrated, the suspiciousness score of some feature sets are identical. We have taken three different strategies to break ties:

- *Core features:* If a core feature is faulty, all products will fail, and thus, for some techniques (e.g., Tarantula) all the feature sets will have the same suspiciousness score. If this occurs, our SBFL approach places core features at the top of the suspiciousness ranking. Notice that this does not happen with all techniques (e.g., Wong).

- *Feature interactions:* Faults in isolated features may distort feature groups suspiciousness scores. Take as an example the simulated fault in `MP3`. All feature sets including the feature `MP3` will fail, which will result, for some techniques (e.g., Tarantula), in all feature sets including `MP3` having the same suspiciousness score, e.g., the Tarantula scores of `MP3` and `MP3-Colour` in Table 9.1 are equal. Under this scenario, when a feature set $S$ has the same suspiciousness that any of its feature subsets $S' \subset S$, then $S'$ is ranked over $S$.

- *Parental relations:* If a parent feature is faulty, all the products containing one ore more of its subfeatures will also be faulty, since parent and child features must appear together in products. Hence, for instance, a bug in the feature `Media` would make all the products including any of its child features to fail, that is, those including `Camera`, `MP3`, or both. To address this issue, when a parent feature has the same suspiciousness score as its child features, the parent feature is ranked first.

All ties obtained after applying the previous strategies are broken randomly. We remark, however, that other strategies would also be feasible and studying their effectiveness remains for future work.

### 9.2.2 Fault isolation

Even if we have a list of the most suspicious feature sets, it could still be challenging to find a product, hopefully as small as possible, where the fault can be easily located. This is the goal of techniques like delta-debugging [ZH02], which aims to generate minimal inputs inducing the failure in the program under test. Based on this idea, in this section, we present a debugging approach for the isolation of bugs in highly configurable systems. The goal is to generate a minimal product, in terms of number of features, where the fault(s) can be easily located. For the generation of the product, we leveraged advanced tools for the automated analysis of feature models. More specifically, we used the analysis operations on feature models integrated into the tool SPLAR [MBC09a].

The overall overview of the approach for generating the minimal product is depicted in Figure 9.2. The debugging approach receives a suspicious feature set ($FS$), a feature model (with a set of features $F$), and the failing product being debugged ($P$) as inputs. Then, a partial configuration is created in three steps, namely: (1) unselect the features that are not part of the product being debugged, (2) select the core features ($C$), and (3) select the features in the suspicious feature set (out of the remaining features). Formally, let $S$ and $R$ be the sets of selected and removed features in the partial configuration respectively. The partial configuration is defined as follows.

$$\begin{aligned} \forall f \in F \bullet \; & f \notin P \;\Rightarrow\; f \in R \wedge \\ & f \in C \;\Rightarrow\; f \in S \wedge \\ & f \in FS \;\Rightarrow\; f \in S \end{aligned} \tag{9.1}$$

The partial configuration is then provided as an input to the propagate operation, which generates a minimal valid product including the suspicious feature set. This operation (also called dependency analysis operation [BSRC10]) receives a partial configuration as input, and it automatically selects and unselects the necessary features to create a valid product according to the constraints of the model (if such product exists). For example, suppose that we run the propagate analysis operation on the selected features {`GPS`, `Camera`}. The operation would propagate the decisions returning the product {`Mobile Phone`, `Calls`, `Screen`, `High resolution`, `Media`, `Camera`, `GPS`}. Notice that the product includes the core features, plus the features `Media` and `High resolution` (both required by `Camera`). It is noteworthy that the minimal product generated is composed of a subset of the features

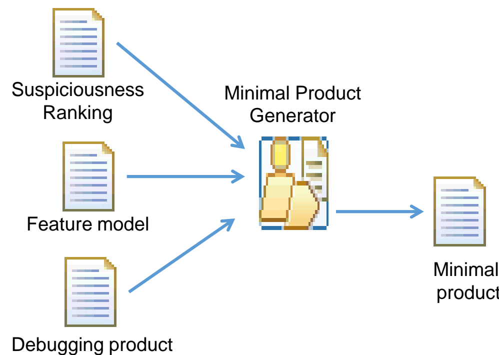



Figure 9.2: Overview of our approach for fault isolation in highly configurable systems

in the product being debugged, and thus no new features are considered, which could result in unexpected results, e.g., new faults being introduced.

Continuing with the previous example, let us assume that the feature `MP3` has the highest suspiciousness score, and P5 = {`MobilePhone`, `Screen`, `Calls`, `High resolution`, `Media`, `Camera`, `MP3`, `GPS`} is the faulty product being debugged. A partial configuration would be created by unselecting the features not contained in the product (`Colour`, `Basic`), selecting the core features (`Mobile phone`, `Calls`, `Screen`), and selecting the suspicious feature set (`MP3`). This configuration would be then provided as input to the propagate operation, which would return the following product {`Mobile phone`, `Calls`, `Screen`, `High resolution`, `Media`, `MP3`}. Note that the features `Media` and `High resolution` are automatically selected, whereas the feature `GPS` is not selected. We may remark that the products generated by the propagate operation are always minimal, i.e., only those features strictly necessary to make a valid product are selected. Therefore, the debugger is provided with the smallest product including the suspicious feature set, contributing to reduce the effort required to locate the bug.

### 9.2.3 Methodology

Figure 9.3 depicts the overall methodology to apply our SPL debugging approach. First, the suspiciousness scores of each feature sets are calculated based on the coverage information and test results, as explained in Section 9.2.1. Then, for each faulty product, the most suspicious feature set is selected and a minimal product is generated and tested. We reiterate that the tests can be performed using any state-of-the-art testing technique and it is out of the scope of this study. If the test outcome is successful, the next most suspicious feature set is selected and another minimal product

is generated. Conversely, if the product fails, the suspicious feature set is reported to the engineer to fix it. This process is repeated until all faults have been fixed. Notice that every time a faulty product is selected, the tests must be executed again to confirm that the product is still buggy, since the faults could have been fixed while debugging previous products. Finally, it is noteworthy that the calculation of the suspiciousness scores is only performed once, unlike related approach where it is calculated every time a bug is fixed [LWG+17]. Although this may affect the accuracy of our approach, we believe that this is a sensible strategy for highly configurable systems where re-executing all the tests is usually very costly [WAG13, WBA+14, ASEZ17].

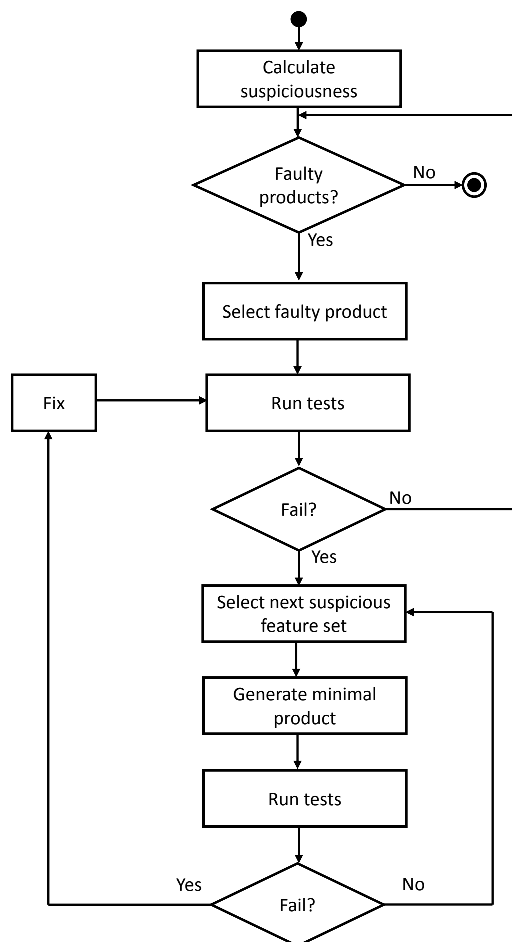



Figure 9.3: Overview of the methodology for SPL fault isolation

## 9.3 Empirical Evaluation

This section empirically evaluates the proposed debugging approach.

### 9.3.1 Research questions

In order to evaluate the effectiveness of feature-based SBFL in SPLs we aim to answer the following Research Questions (RQs):

**RQ1:** *What is the effectiveness of different state-of-the-art suspiciousness techniques at isolating the causes of failures in highly configurable systems?*

**RQ2:** *How the size of the product suite affects the performance of the techniques under study?*

**RQ3:** *How the number and type of faults (single or interaction) affect the performance of the techniques under study?*

### 9.3.2 Experimental design

#### Subject models and product suites

We selected nine feature models representing SPLs of different sizes for the evaluation. Seven of the models were taken from the SPLOT repository [MBC09a]. Additionally, we used the feature model of the Drupal framework, a realistic case study to evaluate variability testing techniques proposed by Sanchez et al. [SSRC14b]. Additionally, we included the Unmanned Aerial Vehicle (UAV) case study. For each subject model, the SPLCAT tool [Joh17] was used to generate two product suites using 2-wise and 3-wise coverage criteria [JHF12]. Table 9.2 depicts the characteristics of the selected models including number of features, number of Cross-Tree Constraintss (CTCs), total number of products, and number of products in the 2-wise and 3-wise product suites respectively.

| Case Study | Features | CTC | Products | 2-wise | 3-wise |
|---|---|---|---|---|---|
| Drupal V3 | 21 | 9 | 96,768 | 11 | 37 |
| Weather station | 23 | 2 | 1,056 | 14 | 40 |
| Eclipse | 29 | 3 | 983,150 | 17 | 54 |
| Android | 45 | 5 | 36,240 | 18 | 67 |
| UAV | 46 | 4 | 2.3E6 | 22 | 74 |
| Dell Laptop | 47 | 109 | 2,319 | 47 | 142 |
| Arcade | 62 | 35 | 3.3E9 | 18 | 65 |
| HIS | 68 | 4 | 6,400 | 12 | 41 |
| Model transformation | 88 | 0 | 1.6E13 | 28 | 133 |

Table 9.2: Subject feature models

### Fault seeding and test execution

We faced two obstacles in the selection of case studies for the evaluation of our approach. First, we found a lack of case studies with available feature models, source code, and test cases. Second, based on our experience with industrial partners [SEA+17], the execution of test cases in real setting is usually a time-consuming process, which hinders the use of real test cases in a large-scale evaluation as the one required in our paper. To address both obstacles, we resorted to a fault simulator in eight of the subject case studies (where no code nor test cases were available), as previously done in related papers [SSRC14a, EBG12, DPC+14, AHTM+14, AHTL+16]. Additionally, we used a real-world case study with available feature model, source code and test cases (Unmanned Aerial Vehicle), in order to evaluate the approach in realistic settings.

For the simulation of faults (in all case studies except UAV), we developed a fault generator to simulate different number and types of faults in the SPL under test. The fault generator is based on the one proposed by Ensan et al. [EBG12] and it has been used in several works to evaluate the fault detection rate of SPL test suites (e.g., [EBG12, SSRC14a, BEG12]). The fault generator simulates faults in single features as well as faults caused by the interaction of two features. More specifically, our generator receives a feature model as an input and returns a random list of faulty feature sets as an output. For instance, the following list simulates two faults in the SPL in Figure 2.2: {{`Colour`}, {`GPS`, `MP3`}}, a fault in the feature `Colour` and another fault caused by the interaction of the features `GPS` and `MP3`.

In addition to the fault simulator, we developed a test system to simulate the test outcomes of each product using a simple oracle: if a product contains any of the features labeled as faulty, the execution of the product is classified as failed, otherwise it is classified as successful. This is an intuitive approach that assumes that the test cases of each product are good enough to reveal failures in the products under test. Note that this is a key requirement for the application of SBFL: if test cases are not able to identify failures, they will certainly not be helpful in identifying faults. Both, the fault simulator and the test system, have been previously used in the literature [SSRC14a].

As for the UAV case study, the experiments were performed employing a Simulink model in charge of simulating the UAV. We employed a test suite composed of 120 test cases. A test case in our case was a set of signals stimulating the inputs of the System Under Test (SUT) over a specific amount of time. The test execution time for each test case lasts from 30 seconds to 3000 seconds.[1] Furthermore, we employed

[1]Notice that this is the simulated test execution time

mutation testing to simulate faults. Mutation testing was employed since it has been demonstrated to be a good substitute of real faults [JJI+14]. For each fault in a specific feature set, a mutant was created, performing the mutation in one of the assets of that feature sets. This mutant was later selected when a product included the faulty feature set. We employed the mutation operators proposed by Hanh et al. for Simulink models [HBT16]. To speed up the evaluation process, we prioritized the test cases with an additional greedy algorithm that used historical data of the test cases. This algorithm demonstrated to be effective in a previous work at detecting faults as fast as possible [ASEZ17]. Since SBFL only uses information whether the test execution passed or failed, once the test cases detected a fault, the test execution was stopped with the aim of speeding up the evaluation process.

### Suspiciousness techniques

We assessed the effectiveness of ten state-of-the-art suspiciousness techniques for the isolation of faults in SPLs. The chosen techniques were Tarantula, Ochiai, Dstar, Naish2, Wong and Russel-Rao, as proposed in [LLT15]. We also included Kulcynski2, Arithmetic-mean, Ample and M2, as they showed promising results in preliminary experiments [XCKX13]. The algebraic form of the chosen techniques are shown in Table 9.3 using the notation presented in Section 9.2.1. In the Dstar technique's formula, the * is an exponent of $N_{CF}$. We set * equal to 2 based on the original paper [WDGL14] and other relevant ones (e.g., [PCJ+17]).

### Evaluation metrics

The following metrics were used to measure the effectiveness of the approach.

*Percentage of examined features (EXAMF).* The EXAM score is one of the most common metrics to evaluate the effectiveness of fault location techniques [WGL+16]. It is calculated as the number of statements examined with respect to the total number of statements in the program. In our approach, the number of statements examined could be intuitively substituted by the number of feature sets examined, and the total number of statements by the total number of features sets. Given a product $p$ being debugged and a faulty feature set $f$, we propose a variant of the EXAM score, called *EXAMF*, calculated as follows:

$$EXAMF(p, f) = \frac{NF_f}{NF_p} \times 100\% \tag{9.2}$$

Table 9.3: Algebraic form of the suspiciousness techniques under evaluation

| Technique | Equation |
|---|---|
| Ample2 | $\lvert\frac{N_{CF}}{N_{CF}+N_F} - \frac{N_{CS}}{N_{CS}+N_{US}}\rvert$ |
| Arithmetic mean | $\frac{2(N_{CF} \times N_{US} - N_{UF} \times N_{CS})}{(N_{CF}+N_{CS}) \times (N_{US}+N_{UF}) + (N_{CF}+N_{UF}) \times (N_{CS}+N_{US})}$ |
| Dstar | $\frac{(N_{CF})^*}{(N_F - N_{CF}) + N_{CS}}$ |
| Kulcynski2 | $\frac{N_{CF}}{N_{UF}+N_{CS}}$ |
| M2 | $\frac{N_{CF}}{N_{CF}+N_S-N_{CS}+2(N_F-N_{CF}+N_{CS})}$ |
| Naish2 | $N_{CF} - \frac{N_{CS}}{N_{CS}+(N_S-N_{CS})+1}$ |
| Ochiai | $\frac{N_{CF}}{\sqrt{N_F(N_{CF}+N_{CS})}}$ |
| Russel-Rao | $\frac{N_{CF}}{N_{CF}+N_F-N_{CF}+N_{CS}+N_S-N_{CS}}$ |
| Tarantula | $\frac{N_{CF}/N_F}{N_{CS}/N_S+N_{CF}/N_F}$ |
| Wong | $N_{CF}$ |

Where $NF_f$ is the number of feature sets examined to isolate the fault in f, and $NF_p$ is the total number of feature sets in $p$. Since we are aiming at faults caused by a single feature or interaction between two features, $NF_p$ is equal to all the valid possible combinations of one or two of the features of $p$. This was calculated using the SPLCAT tool. The lower the EXAMF score is, the more effective is the technique.

As an example, consider a fault in the feature `MP3`, and P5 = {`MobilePhone`, `Screen`, `Calls`, `High resolution`, `Media`, `Camera`, `MP3`, `GPS`} the faulty product being debugged. Let us suppose that `GPS` is the most suspicious feature and `MP3` the second most suspicious feature, according to a certain technique. Accordingly, the debugger would examine first the `GPS` feature, proceeding later to examine the `MP3` feature. Considering that the total number of valid feature sets (i.e., single features and pairs of features) in P5 is 28, this metric is calculated as $EXAMF(P5, \{MP3\}) = (2/28) \times 100 = 7.14$. This means that 7.14% of the feature sets in P5 had to be examined in order to locate the fault in `MP3`.

The EXAMF metric measures the effectiveness of a fault localization technique at detecting a single fault. In the cases were several faults are present, the effectiveness of each fault localization technique was evaluated as the average EXAMF score. Thus, the *average EXAMF* score for multiple faulty feature sets $F$ in a product $p$ that is being debugged is calculated as follows:

$$\frac{\sum_{i=1}^{|F|} EXAMF(p, F_i)}{|F|} \quad (9.3)$$

As an example, let us suppose two faults in the `MP3` and `GPS` features, and P5 the faulty product being debugged. Let us suppose that 4 feature sets were examined before isolating the fault in `MP3`, and 5 feature sets were checked before isolating the bug in `GPS`, i.e., $EXAMF(P5, \{MP3\}) = (4/28) \times 100 = 14.2$ and $EXAMF(P5, \{GPS\}) = (5/28) \times 100 = 17.8$. The average EXAMF is calculated as $(14.2 + 17.8)/2 = 16$. That is, 16% of the feature sets need to be examined on average to locate each faulty feature set in P5.

### Experiments

In order to answer our research questions, we performed five independent experiments with different number and types of simulated faults. Each experiment was conducted on the subject models depicted in Table 9.2 assessing the effectiveness of the ten suspiciousness techniques depicted in Table 9.3. Table 9.4 shows the number of simulated faults in single and pairs of features in each experiment. As proposed by Sanchez et al. [SSRC14a], the maximum number of faults in each model was set to n/10, being $n$ the number of features in the SPL. For the fifth experiment, where faults due to single features and interaction of two features are combined, the distribution of the simulated faults was the same for both type of faults, as proposed in [SSRC14a]. For each experiment and case study, five different distributions of faults were randomly generated, so-called *test scenarios*, in order to calculate averages. In total, 40 different test scenarios were run on each experiment and product suite: 8 case studies $\times$ 5 test scenarios.

| Experiment | Single faults | Interaction faults |
|---|---|---|
| 1 | 1 | 0 |
| 2 | [2, n/10] | 0 |
| 3 | 0 | 1 |
| 4 | 0 | [2, n/10] |
| 5 | [1, n/5] | [1, n/5] |

Table 9.4: Types of faults simulated in each experiment (n = number of features in the SPL)

### 9.3.3 Experimental results

#### Experiment 1: a fault in a single feature

This experiment aims at evaluating the approach when the SPL has one fault in a single feature. Tables 9.5 and 9.6 report the EXAMF values of each suspiciousness technique under evaluation on the data collected from the pairwise and 3-wise product suites respectively. The best value on each column is highlighted in boldface. We reiterate that the shown values are the average of five different scenarios with a randomly simulated fault on each of them. The EXAMF score ranged between 0.07% and 7.6% for the pairwise product suite, and between 0.07% and 8.43% for the 3-wise suite. That is, both product suites yielded similar results, with only slight differences in favor of the pairwise suite. This means that having more test data information was not necessarily helpful in this experiment.

The performance of all the techniques was consistent in all the case studies, and in both product suites. The faulty feature was successfully ranked as the most suspicious feature in 100% of the test scenarios for Ample2, Dstar, Kulcynski2, M2, and Ochiai, i.e., these were the techniques showing the best performance. In the case of Tarantula, the most suspicious feature was ranked first in 97.5% (78 out of 80) of the test scenarios. The technique performing worst was Arithmetic mean, followed by Naish2, Russel-Rao, and Wong.

Table 9.5: EXAMF scores obtained using the pairwise product suite in Experiment 1. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | **1.24** | **1.20** | **1.04** | **0.30** | **0.31** | **0.77** | **0.14** | **0.07** | **0.12** | **0.58** |
| Arithmetic M. | 7.64 | 2.14 | 4.2 | 1.27 | 2.62 | 3.86 | 0.97 | 0.64 | 2.06 | 2.82 |
| Dstar | **1.24** | **1.20** | **1.04** | **0.30** | **0.31** | **0.77** | **0.14** | **0.07** | **0.12** | **0.58** |
| Kulcynski2 | **1.24** | **1.20** | **1.04** | **0.30** | **0.31** | **0.77** | **0.14** | **0.07** | **0.12** | **0.58** |
| M2 | **1.24** | **1.20** | **1.04** | **0.30** | **0.31** | **0.77** | **0.14** | **0.07** | **0.12** | **0.58** |
| Naish2 | 1.73 | **1.20** | 1.94 | **0.30** | 0.36 | 1.71 | 0.24 | 0.16 | 0.34 | 0.88 |
| Ochiai | **1.24** | **1.20** | **1.04** | **0.30** | **0.31** | **0.77** | **0.14** | **0.07** | **0.12** | **0.58** |
| Russel-Rao | 1.73 | **1.20** | 1.94 | **0.30** | 0.36 | 1.71 | 0.24 | 0.16 | 0.34 | 0.88 |
| Tarantula | **1.24** | **1.20** | 1.17 | **0.30** | **0.31** | **0.77** | **0.14** | **0.07** | **0.12** | 0.59 |
| Wong | 1.73 | **1.20** | 1.94 | **0.30** | 0.36 | 1.71 | 0.24 | 0.16 | 0.34 | 0.88 |
| Mean | 2.03 | 1.29 | 1.64 | 0.34 | 0.56 | 1.36 | 0.25 | 0.15 | 0.38 | 0.90 |

#### Experiment 2: multiple faults in single features

This experiment evaluates the approach when the SPL contains multiple faults in two or more single features. Tables 9.7 and 9.8 show the average EXAMF values of each technique under evaluation on the data collected from the pairwise and 3-wise product suites respectively. As illustrated, the results obtained with the 3-wise suite

Table 9.6: EXAMF scores obtained using the 3-wise product suite in Experiment 1. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | **1.92** | **1.11** | **0.91** | **0.29** | **0.31** | **0.76** | **0.13** | **0.07** | **0.10** | **0.62** |
| Arithmetic M. | 8.43 | 1.83 | 3.83 | 1.31 | 2.83 | 3.82 | 1.00 | 0.61 | 1.98 | 2.85 |
| Dstar | **1.92** | **1.11** | **0.91** | **0.29** | **0.31** | **0.76** | **0.13** | **0.07** | **0.10** | **0.62** |
| Kulcynski2 | **1.92** | **1.11** | **0.91** | **0.29** | **0.31** | **0.76** | **0.13** | **0.07** | **0.10** | **0.62** |
| M2 | **1.92** | **1.11** | **0.91** | **0.29** | **0.31** | **0.76** | **0.13** | **0.07** | **0.10** | **0.62** |
| Naish2 | 2.48 | **1.11** | 1.73 | **0.29** | 0.36 | 1.71 | 0.24 | 0.16 | 0.28 | 0.93 |
| Ochiai | **1.92** | **1.11** | **0.91** | **0.29** | **0.31** | **0.76** | **0.13** | **0.07** | **0.10** | **0.62** |
| Russel-Rao | 2.48 | **1.11** | 1.73 | **0.29** | 0.36 | 1.71 | 0.24 | 0.16 | 0.28 | 0.93 |
| Tarantula | **1.92** | **1.11** | 0.99 | **0.29** | **0.31** | **0.76** | **0.13** | **0.07** | **0.10** | 0.63 |
| Wong | 2.48 | **1.11** | 1.73 | **0.29** | 0.36 | 1.71 | 0.24 | 0.16 | 0.28 | 0.93 |
| Mean | 2.74 | 1.18 | 1.46 | 0.39 | 0.58 | 1.35 | 0.25 | 0.15 | 0.34 | 0.94 |

(between 0.10% and 8.89%) were slightly better than those obtained with the pairwise suite (between 0.22% and 12.38%). More specifically, the EXAMF values of the 3-wise suite outperformed those of the pairwise suite in 59 out of the 80 measures (10 techniques x 8 case studies). This means that the use of more test data improved the performance of the fault isolation techniques in this particular experiment.

Overall, the technique performing best with both product suites was Tarantula, followed by Kulcynski2, and Ample2. Conversely, Russel-Rao, Wong and Naish2, which showed exactly the same results in all case studies, resulted in the techniques with worst performance in this experiment.

Table 9.7: EXAMF scores obtained using the pairwise product suite in Experiment 2. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | 3.27 | 1.37 | 6.26 | 0.90 | 0.77 | 1.11 | 1.27 | 0.60 | 1.42 | 1.88 |
| Arithmetic M. | 7.32 | 2.95 | 6.26 | 1.10 | 2.36 | 2.79 | 0.98 | 0.35 | 1.32 | 2.82 |
| Dstar | 3.74 | 1.41 | 11.50 | 3.00 | 3.42 | 2.04 | 2.98 | 0.89 | 4.99 | 3.78 |
| Kulcynski2 | **3.10** | **1.21** | 3.74 | 0.40 | **0.46** | **0.98** | **0.49** | **0.22** | 0.39 | 1.22 |
| M2 | 3.74 | 1.71 | 12.06 | 3.16 | 3.97 | 2.28 | 3.04 | 0.89 | 5.60 | 4.05 |
| Naish2 | 4.59 | 2.01 | 12.38 | 3.33 | 4.57 | 2.54 | 3.10 | 0.94 | 5.79 | 4.36 |
| Ochiai | 3.58 | 1.41 | 10.13 | 2.48 | 2.69 | 1.40 | 2.66 | 0.89 | 3.89 | 3.23 |
| Russel-Rao | 4.59 | 2.01 | 12.38 | 3.33 | 4.57 | 2.54 | 3.10 | 0.94 | 5.79 | 4.36 |
| Tarantula | **3.10** | **1.21** | **3.59** | **0.37** | **0.46** | **0.98** | **0.49** | **0.22** | **0.36** | **1.20** |
| Wong | 4.59 | 2.01 | 12.38 | 3.33 | 4.57 | 2.54 | 3.10 | 0.94 | 5.79 | 4.36 |
| Mean | 4.16 | 2.73 | 9.07 | 2.14 | 2.78 | 1.92 | 2.12 | 0.69 | 3.53 | 3.12 |

### Experiment 3: fault in a feature interaction

This experiment evaluates the approach under the presence of one fault due to the interaction of two features. Tables 9.9 and 9.10 show the mean EXAMF values of each technique over the five test scenarios. As in the previous experiment, the results obtained with the 3-wise suite were significantly better than those obtained with the

Table 9.8: EXAMF scores obtained using the 3-wise product suite in Experiment 2. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | **2.1** | **1.10** | 1.67 | 0.72 | 0.62 | 1.12 | 1.14 | 0.34 | 0.80 | 1.06 |
| Arithmetic M. | 8.55 | 2.80 | 6.00 | 1.23 | 2.21 | 2.85 | 0.91 | 0.38 | 1.08 | 2.89 |
| Dstar | 3.66 | 1.20 | 7.14 | 2.65 | 2.66 | 2.23 | 2.91 | 1.22 | 4.41 | 3.12 |
| Kulcynski2 | 2.50 | **1.10** | 1.33 | 0.34 | **0.29** | 0.95 | 0.38 | **0.10** | 0.22 | 0.80 |
| M2 | 3.93 | 1.26 | 7.72 | 2.88 | 3.00 | 2.40 | 3.03 | 1.25 | 4.72 | 3.36 |
| Naish2 | 4.13 | 1.66 | 8.89 | 3.09 | 3.36 | 2.60 | 3.10 | 1.27 | 5.24 | 3.71 |
| Ochiai | 2.61 | **1.10** | 5.37 | 2.20 | 1.91 | 1.47 | 2.62 | 1.11 | 3.55 | 2.43 |
| Russel-Rao | 4.13 | 1.66 | 8.89 | 3.09 | 3.36 | 2.60 | 3.10 | 1.27 | 5.24 | 3.71 |
| Tarantula | 2.50 | **1.10** | **1.21** | **0.31** | **0.29** | **0.95** | **0.38** | **0.10** | **0.20** | **0.78** |
| Wong | 4.13 | 1.66 | 8.89 | 3.09 | 3.36 | 2.60 | 3.10 | 1.27 | 5.24 | 3.71 |
| Mean | 3.82 | 1.46 | 5.71 | 1.96 | 2.11 | 1.98 | 2.07 | 0.83 | 3.07 | 2.56 |

pairwise suite. More specifically, the EXAMF values of the 3-wise suite outperformed those of the pairwise suite in 73 out of the 80 measures. Interestingly, the mean EXAMF values were significantly higher (up to 47.57%) than those observed in the previous experiments, which suggests that, as expected, locating bugs caused by the interaction of features is harder than isolating bugs in single features. Also, analogously to Experiment 1, where a single fault was also simulated, the performance of the techniques was consistent across all the case studies showing identical conclusions for both product suites. More specifically, the techniques performing best were Ample2, Dstar, Kulcynski2, M2, and Ochiai, all of them with the same average score. Conversely, the technique Arithmetic mean performed significantly bad in comparison with the rest of techniques, with a mean score over 21% with both product suites.

Table 9.9: EXAMF scores obtained using the pairwise product suite in Experiment 3. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | **2.2** | **1.24** | **1.22** | **0.27** | **0.3** | **0.97** | **0.32** | **0.16** | **0.31** | **0.77** |
| Arithmetic M. | 36.41 | 17.05 | 47.35 | 13.59 | 13.57 | 15.7 | 13.34 | 5.34 | 22.37 | 20.52 |
| Dstar | **2.2** | **1.24** | **1.22** | **0.27** | **0.3** | **0.97** | **0.32** | **0.16** | **0.31** | **0.77** |
| Kulcynski2 | **2.2** | **1.24** | **1.22** | **0.27** | **0.3** | **0.97** | **0.32** | **0.16** | **0.31** | **0.77** |
| M2 | **2.2** | **1.24** | **1.22** | **0.27** | **0.3** | **0.97** | **0.32** | **0.16** | **0.31** | **0.77** |
| Naish2 | 13.89 | 5.28 | 23.2 | 3.16 | 4.98 | 11.59 | 2.44 | 2.19 | 6.51 | 8.14 |
| Ochiai | **2.2** | **1.24** | **1.22** | **0.27** | **0.3** | **0.97** | **0.32** | **0.16** | **0.31** | **0.77** |
| Russel-Rao | 13.89 | 5.28 | 23.2 | 3.16 | 4.98 | 11.59 | 2.44 | 2.19 | 6.51 | 8.14 |
| Tarantula | 4.93 | 4.2 | 3.27 | 1.29 | 0.65 | 1.66 | 2.56 | 0.34 | 1.06 | 2.21 |
| Wong | 13.89 | 5.28 | 23.2 | 3.16 | 4.98 | 11.59 | 2.44 | 2.19 | 6.51 | 8.14 |
| Mean | 9.40 | 4.32 | 12.63 | 2.57 | 3.07 | 5.70 | 2.48 | 1.30 | 4.45 | 5.10 |

#### Experiment 4: multiple faults in feature interactions

This experiment aims to evaluate our approach when the SPL has multiple faults caused by feature interactions. Tables 9.11 and 9.12 show the average EXAMF values

Table 9.10: EXAMF scores obtained using the 3-wise product suite in Experiment 3. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | **1.42** | **1.05** | **0.73** | **0.21** | **0.26** | **0.71** | **0.20** | **0.07** | **0.10** | **0.53** |
| Arithmetic M. | 38.86 | 17.73 | 47.57 | 12.43 | 13.89 | 15.80 | 14.58 | 5.47 | 22.9 | 21.03 |
| Dstar | **1.42** | **1.05** | **0.73** | **0.21** | **0.26** | **0.71** | **0.20** | **0.07** | **0.10** | **0.53** |
| Kulcynski2 | **1.42** | 1.05 | **0.73** | **0.21** | **0.26** | **0.71** | **0.20** | **0.07** | **0.10** | **0.53** |
| M2 | **1.42** | **1.05** | **0.73** | **0.21** | **0.26** | **0.71** | **0.20** | **0.07** | **0.10** | **0.53** |
| Naish2 | 5.94 | 3.14 | 5.30 | 2.01 | 1.68 | 6.35 | 1.49 | 1.16 | 1.19 | 3.14 |
| Ochiai | **1.42** | **1.05** | 0.73 | **0.21** | **0.26** | **0.71** | **0.20** | **0.07** | **0.10** | **0.53** |
| Russel-Rao | 5.94 | 3.14 | 5.30 | 2.01 | 1.68 | 6.35 | 1.49 | 1.16 | 1.19 | 3.14 |
| Tarantula | **1.42** | 1.86 | **0.73** | **0.21** | **0.26** | **0.71** | 0.47 | **0.07** | **0.10** | 0.65 |
| Wong | 5.94 | 3.14 | 5.30 | 2.01 | 1.68 | 6.35 | 1.49 | 1.16 | 1.19 | 3.14 |
| Mean | 6.52 | 3.43 | 6.78 | 1.97 | 2.05 | 3.91 | 2.05 | 0.94 | 2.71 | 3.37 |

obtained in each of the case studies for five different test scenarios. As in the previous two experiments, the techniques showed significantly better performance with the 3-wise suite compared to the pairwise suite. This improvement was significant in the case of Tarantula where the overall average EXAMF value decreased from 5.55% with the pairwise suite to 0.99% with the 3-wise suite. Overall, the EXAMF values of the 3-wise suite outperformed those of the pairwise suite in 58 out of the 80 measures. It is also noteworthy that the average EXAMF scores in this experiment are noticeably higher than in the previous ones. This suggests that isolating multiple interaction faults imposes a significantly hard problem for the techniques under evaluation.

From the results, it is observed that Tarantula is the most effective technique to isolate multiple interaction faults, achieving the lowest average EXAMF value in 7 out of the 8 case studies with both test suites. Conversely, Arithmetic mean was the technique that showed the worst performance, followed by Russel-Rao, Naish2 and Wong.

Table 9.11: EXAMF scores obtained using the pairwise product suite in Experiment 4. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | **8.93** | 3.93 | 14.47 | 2.35 | 3.69 | 2.38 | 4.93 | 1.49 | 5.15 | 5.25 |
| Arithmetic M. | 44.69 | 19.86 | 52.54 | 11.52 | 17.44 | 14.96 | 10.90 | 5.39 | 23.23 | 22.28 |
| Dstar | 13.06 | 7.39 | 27.70 | 4.97 | 6.17 | 8.85 | 7.14 | 3.40 | 15.06 | 10.42 |
| Kulcynski2 | 14.92 | 5.19 | 14.49 | 3.23 | 2.27 | 1.79 | 3.69 | 1.48 | 5.38 | 5.83 |
| M2 | 13.19 | 6.14 | 27.57 | 5.89 | 6.9 | 5.23 | 7.91 | 3.42 | 15.29 | 10.17 |
| Naish2 | 23.13 | 13.01 | 37.55 | 8.72 | 12.45 | 9.53 | 9.86 | 4.20 | 18.56 | 15.22 |
| Ochiai | 11.91 | 4.51 | 23.26 | 3.52 | 3.78 | 1.96 | 5.99 | 2.97 | 11.16 | 7.67 |
| Russel-Rao | 23.13 | 13.01 | 37.55 | 8.72 | 12.45 | 9.53 | 9.86 | 4.20 | 18.56 | 15.22 |
| Tarantula | 12.96 | **3.73** | **13.78** | **2.10** | **1.75** | **1.66** | **3.66** | **1.45** | **5.09** | **5.13** |
| Wong | 23.13 | 13.01 | 37.55 | 8.72 | 12.45 | 9.53 | 9.86 | 4.20 | 18.56 | 15.22 |
| Mean | 18.90 | 8.98 | 28.65 | 5.97 | 7.93 | 6.54 | 7.38 | 3.22 | 13.60 | 11.34 |

Table 9.12: EXAMF scores obtained using the 3-wise product suite in Experiment 4. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | **2.16** | 1.28 | 2.03 | 0.96 | 0.69 | 1.43 | 2.36 | 0.47 | 1.56 | 1.44 |
| Arithmetic M. | 45.80 | 20.39 | 51.16 | 12.26 | 11.61 | 16.30 | 11.02 | 5.46 | 22.98 | 21.89 |
| Dstar | 6.14 | 2.50 | 18.51 | 5.44 | 4.27 | 3.37 | 10.95 | 3.73 | 14.09 | 7.67 |
| Kulcynski2 | 3.06 | 1.29 | 2.11 | 0.83 | 0.55 | 0.87 | 0.98 | 0.25 | 0.39 | 1.15 |
| M2 | 11.6 | 4.63 | 23.00 | 7.08 | 5.41 | 4.22 | 11.66 | 4.19 | 15.14 | 9.66 |
| Naish2 | 18.33 | 7.63 | 28.92 | 8.95 | 6.85 | 6.59 | 12.36 | 4.92 | 12.37 | 13.06 |
| Ochiai | 2.18 | 1.54 | 7.71 | 2.78 | 2.22 | 1.28 | 8.03 | 2.86 | 4.19 | 4.44 |
| Russel-Rao | 18.33 | 7.63 | 28.92 | 8.95 | 6.85 | 6.59 | 12.36 | 4.92 | 12.37 | 13.06 |
| Tarantula | 2.17 | **1.10** | **1.68** | **0.71** | **0.43** | **0.87** | **0.85** | **0.22** | **0.29** | **0.93** |
| Wong | 18.33 | 7.63 | 28.92 | 8.95 | 6.85 | 6.59 | 12.36 | 4.92 | 16.81 | 12.37 |
| Mean | 12.81 | 5.56 | 19.30 | 5.69 | 4.57 | 4.81 | 8.29 | 3.19 | 11.40 | 8.40 |

### Experiment 5: faults in single features and feature interactions

This experiment assessed the proposed approach in SPLs containing faults in single features as well as faults due to the interaction of two features. Tables 9.13 and 9.14 show the average EXAMF values obtained in this experiment for the nine case studies. As in the previous experiments, the overall performance of most techniques was better when using the 3-wise suite than when using the pairwise suite. More specifically, the EXAMF values of the 3-wise suite outperformed those of the pairwise suite in 54 out of the 80 measures. In contrast to the previous experiments, the results with each suite revealed slight differences, although they overall agree that the techniques performing best were Tarantula, Kulcynski2 and Ample2. Conversely, and in line with the previous experiments, the technique showing the worst performance is Arithmetic mean, followed by Russel-Rao, Naish2 and Wong.

Table 9.13: EXAMF scores obtained using the pairwise product suite in Experiment 5. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | 11.29 | **1.55** | 4.66 | **1.55** | 0.97 | 1.79 | **2.13** | 1.28 | 2.71 | 3.10 |
| Arithmetic M. | 15.42 | 8.16 | 16.47 | 6.15 | 4.75 | 4.92 | 4.48 | 1.28 | 5.42 | 7.44 |
| Dstar | 10.21 | 3.68 | 9.94 | 4.41 | 3.99 | 3.15 | 4.71 | 1.85 | 7.47 | 5.48 |
| Kulcynski2 | **9.45** | 1.85 | 4.71 | 2.01 | 0.82 | 1.09 | 2.26 | **0.73** | **1.94** | **2.76** |
| M2 | 10.17 | 3.54 | 9.73 | 4.44 | 3.86 | 2.75 | 4.94 | 1.89 | 8.00 | 5.48 |
| Naish2 | 9.70 | 4.57 | 13.31 | 5.36 | 5.04 | 4.05 | 5.3 | 1.98 | 8.73 | 6.44 |
| Ochiai | 11.68 | 2.95 | 7.92 | 3.31 | 2.37 | 1.85 | 4.02 | 1.83 | 5.98 | 4.65 |
| Russel-Rao | 9.70 | 4.57 | 13.31 | 5.36 | 5.04 | 4.05 | 5.30 | 1.98 | 8.73 | 6.44 |
| Tarantula | 12.06 | 1.85 | **4.39** | 2.01 | **0.82** | **0.96** | 2.26 | **0.73** | **1.94** | 3.00 |
| Wong | 9.70 | 4.57 | 13.31 | 5.36 | 5.04 | 4.05 | 5.30 | 1.98 | 8.73 | 6.44 |
| Mean | 10.94 | 3.73 | 9.77 | 4.00 | 3.27 | 2.87 | 4.07 | 1.55 | 5.96 | 5.12 |

Table 9.14: EXAMF scores obtained using the 3-wise product suite in Experiment 5. Best values on each column are highlighted in boldface

| Technique | Drupal V3 | Weather St. | Eclipse | Android | UAV | Dell L. | Arcade | HIS | Model T. | Mean |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | 2.46 | **1.27** | 1.99 | **0.89** | 0.5 | 1.22 | **1.5** | 0.74 | 1.69 | 1.36 |
| Arithmetic M. | 13.91 | 7.83 | 18.29 | 6.02 | 5.59 | 5.43 | 5.46 | 1.33 | 5.26 | 7.68 |
| Dstar | 4.92 | 3.05 | 8.13 | 4.51 | 3.61 | 3.52 | 5.45 | 2.45 | 8.24 | 4.87 |
| Kulcynski2 | **2.11** | **1.27** | 1.97 | 1.08 | **0.38** | 0.91 | 1.58 | **0.33** | **0.75** | 1.15 |
| M2 | 5.93 | 3.73 | 9.31 | 4.61 | 3.89 | 3.03 | 5.66 | 2.56 | 8.56 | 5.25 |
| Naish2 | 7.58 | 4.22 | 12.2 | 5.14 | 4.44 | 3.97 | 5.85 | 2.61 | 9.26 | 6.14 |
| Ochiai | 3.57 | 1.72 | 4.08 | 3.21 | 2.2 | 1.49 | 4.86 | 2.07 | 6.40 | 3.28 |
| Russel-Rao | 7.58 | 4.22 | 12.2 | 5.14 | 4.44 | 3.97 | 5.85 | 2.61 | 9.26 | 6.14 |
| Tarantula | 2.35 | **1.27** | **1.72** | 1.08 | **0.38** | **0.78** | 1.58 | **0.33** | **0.75** | **1.14** |
| Wong | 7.58 | 4.22 | 12.2 | 5.14 | 4.44 | 3.97 | 5.85 | 2.61 | 9.26 | 6.14 |
| Mean | 5.80 | 3.28 | 8.21 | 3.68 | 2.99 | 2.83 | 4.36 | 1.76 | 5.94 | 4.31 |

### Statistical Analysis

Results of the performed experiments were analyzed by means of statistical analysis. Specifically, for each experiment of each case study, each pair of the metrics were analyzed with a post-hoc analysis employing the Kruskal-Wallis test [VD98], which is a non-parametric method. This returned a p-value for each pair of metrics. The p-value indicates whether there is a statistically significant difference between two different SBFL techniques or not. As the statistical significance level was set to 99%, we considered that there was statistical significance between two different techniques when the p-value $< 0.01$. When the p-value of the Kruskal-Wallis test returned a value below 0.01, the Vargha and Delaney test was employed to obtain the $\hat{A}_{12}$ value [AB11, VD00]. The $\hat{A}_{12}$ value determines the difference between two techniques and see which of the two techniques is better.

Tables 9.15 and 9.16 summarize the results for the statistical analysis related to the performed experiments for the pairwise and 3-wise suites. These tables indicate the number of times, out of 40 (5 experiments $\times$ 8 case studies), in which the technique in the row outperformed the technique in the column with statistical significance (i.e., p-value $< 0.01$ and the $\hat{A}_{12}$ in favor of the technique in the row). After the statistical analysis, it can be appreciated that the best metric was Kulcynski2. In fact, this metric was not statistically outperformed by any of the other metrics. However, the rest of metrics were outperformed by Kulcynski2 at least in one of the experiments for both, the pairwise and 3-wise suite.

Apart from Kulcynski2, two techniques can be considered as valid ones as compared to the rest for solving the fault localization problem in SPLs: Tarantula and Ample2. Kulcynski2 statistically outperformed Tarantula only in one test scenario for each of the product suites, whereas it statistically outperformed Ample2 in one test scenario for the pairwise suite and in five test scenarios for the 3-wise suite.

Table 9.15: Summary of the Results for the Statistical Analysis for the pairwise suite

| | Ample2 | Arithmetic | Dstar | Kulcynski2 | M2 | Naish2 | Ochiai | Russel-Rao | Tarantula | Wong |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | – | 36 | 13 | 0 | 12 | 28 | 8 | 29 | 1 | 29 |
| Arithmetic | 0 | – | 3 | 0 | 3 | 6 | 2 | 6 | 0 | 6 |
| Dstar | 0 | 28 | – | 0 | 0 | 9 | 0 | 9 | 1 | 9 |
| Kulcynski2 | 1 | 39 | 12 | – | 12 | 29 | 9 | 29 | 1 | 29 |
| M2 | 0 | 27 | 0 | 0 | – | 9 | 0 | 9 | 1 | 9 |
| Naish2 | 0 | 7 | 0 | 0 | 0 | – | 0 | 0 | 0 | 0 |
| Ochiai | 0 | 32 | 1 | 0 | 0 | 16 | – | 16 | 1 | 16 |
| Russel-Rao | 0 | 7 | 0 | 0 | 0 | 0 | 0 | – | 0 | 0 |
| Tarantula | 1 | 33 | 12 | 0 | 13 | 22 | 9 | 21 | – | 21 |
| Wong | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | – |

Table 9.16: Summary of the Results for the Statistical Analysis for the 3-wise suite

| | Ample2 | Arithmetic | Dstar | Kulcynski2 | M2 | Naish2 | Ochiai | Russel-Rao | Tarantula | Wong |
|---|---|---|---|---|---|---|---|---|---|---|
| Ample2 | – | 42 | 23 | 0 | 24 | 40 | 17 | 40 | 1 | 40 |
| Arithmetic | 0 | – | 5 | 0 | 5 | 7 | 4 | 7 | 0 | 7 |
| Dstar | 0 | 34 | – | 0 | 1 | 23 | 0 | 23 | 1 | 23 |
| Kulcynski2 | 6 | 45 | 20 | – | 24 | 39 | 17 | 39 | 1 | 39 |
| M2 | 0 | 33 | 0 | 0 | – | 17 | 0 | 17 | 1 | 17 |
| Naish2 | 0 | 23 | 0 | 0 | 0 | – | 0 | 0 | 0 | 0 |
| Ochiai | 0 | 37 | 10 | 0 | 12 | 35 | – | 35 | 1 | 35 |
| Russel-Rao | 0 | 23 | 0 | 0 | 0 | 0 | 0 | – | 0 | 0 |
| Tarantula | 6 | 45 | 22 | 0 | 23 | 38 | 17 | 38 | – | 38 |
| Wong | 0 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | – |

### 9.3.4 Discussion

We now summarize the results and what they tell us about the research questions.

#### RQ1: Effectiveness of different suspiciousness techniques

The results of the experiments and the corresponding statistical analysis of the data reveal that the techniques Kulcynski, Ample2 and Tarantula are the most effective suspiciousness techniques for fault isolation in SPLs. It is remarkable that these three technique showed a very stable performance with different types of faults and suite sizes. In contrast, the results of Ochiai, Dstar and M2 were more sensitive to the type of faults, and diverged significantly among the different experiments. The techniques Arithmetic mean, Russel-Rao, Naish2, and Wong performed badly in all experiments. In the light of these results, RQ1 is answered as follows:

*Different suspiciousness techniques may perform very differently in the context of SPLs. Based on the results of our study, the most effective suspiciousness techniques are Kulcynski2, Tarantula and Ample2. Conversely, the techniques Arithmetic mean, Wong, Russel-Rao and Naish2 perform badly and they should be avoided.*

### RQ2: Size of the suite

The results obtained with the 3-wise suite were consistently better when compared with those obtained with the pairwise suite. The only exception was Experiment 1 where both suites yielded similar results. We suspect that this was due to the simplicity of the problem, which made both suites to obtain the optimal result easily. Overall, however, the experimental results were expected and in line with the theory behind SBFL, which states that the accuracy of the techniques is better as the the size of the test suite increases. Based on our results, RQ2 is answered as follows:

*The accuracy of the fault localization techniques gets better as the number of products in the suite increases.*

### RQ3: Types and Number of Faults

The experimental results show that isolating a single fault (Experiments 1 and 3) is significantly easier than isolating multiple faults (Experiments 2, 4, and 5). This was expected because multiple faults may interfere among them making the results of the suspiciousness metrics less accurate. The results also suggest that detecting multiple interaction faults (Experiment 4) is significantly harder than detecting multiple single and interaction faults, either in isolation (Experiment 2) or combined (Experiment 5). In the view of these results, RQ3 is answered as follows:

*The number and type of faults have a strong impact in the effectiveness of the suspiciousness techniques. Isolating single faults is significantly easier than locating multiple bugs. Locating multiple bugs caused by the interaction among different features is the hardest scenario.*

## 9.4 Threats to validity

The factors that could have influenced our work are summarized in the following internal and external validity threats.

**Internal validity**: The number of simulated faults on each feature model could introduce a bias in our evaluation. To mitigate this threat, we experimented with different amounts of simulated faults, up to a maximum of 10% with respect to the number of features, as proposed in [SSRC14a]. Similarly, it could be the case that simulated faults affect differently to different types of features, or that the debugging approach performs differently on products of different sizes. To address these threats, we created five different test scenarios with different simulated faults and two different product suites on each case study. Finally, another threat is related to the developed test system simulator, which assumes that test cases and test oracles are always capable of differentiating a faulty product from a non-faulty one. We reiterate, however, that a key requirement for the successful application of SBFL is that test cases are able to reveal the faults to be located. To mitigate this threat, we also evaluated our approach using a real-world case study with real test cases and mutation testing. The results are consistent with those obtained using simulated faults.

**External validity**: As mentioned in Section 9.2.1, if a core feature is faulty, all products will fail, and thus the results of suspiciousness techniques will not be accurate enough to locate the bug. This is an intrinsic problem of SBFL techniques which depends on the existence of successful and failing tests to identify the suspicious components. To alleviate this threat, when all the products in the product suite fail, core features are placed at the top of the suspiciousness ranking. As another limitation, we considered faults in single features and faults caused by the interaction between two features, as these are common types of faults in software programs [KKLH09]. Thus, evaluating the effectiveness of the approach at isolating faults caused by the interaction among three or more features remains for future work.

A problem with any empirical evaluation is related with the generalization of the results. We used nine case studies, which might not be enough to conclude that some techniques are better than others. To mitigate this threat, we chose case studies from different domains with different sizes and characteristics to assure a sufficient degree of heterogeneity.

**Conclusion validity**: A possible conclusion validity threat could be the configuration for the Dstar technique. Notice that this technique can be adjusted by setting the *, which is the exponent of $N_{CF}$. To reduce this threat, we set * to 2 based on previous studies [PCJ+17].

## 9.5 Related Work

In this section, we overview those works closely related to our approach in the fields of SPL testing, SBFL and fault isolation.

Lopez-Herrejon et al. conducted a systematic mapping study on Combinatorial Interaction Testing (CIT) for SPLs [LHFRE15]. They found that a majority of the papers focus on deriving products from variability models (typically a FM) using pairwise testing [PSK+10, POS+12, CDS08]. Similarly to those papers, we leverage the tools for the automated analysis of feature models. In particular, we propose to use the propagate analysis operation, typically used during product configuration, to generate minimal products including the suspicious feature set, easing the isolation of faults. In contrast with previous work, however, this study focuses on debugging, not testing, and thus our approach does not aim to reveal failures, but to locate the bugs that trigger them. Similarly, a number of papers addressed the problem of product prioritization in SPLs [SSRC14a, SSPRC15, LHLE15, PSS+16, WBA+14, AHTM+14, AHTL+16] for reordering the products derived from a feature model according to different criteria (e.g., complexity of products). In our case, rather than prioritizing products or test cases, we prioritize feature sets according to their suspiciousness score, calculated using state-of-the-art SBFL techniques.

In addition to product prioritization, our fault isolation approach also shares similarities with delta modeling [CHS10]. Delta modeling is an approach for the SPL automated product derivation [CHS10]. It consists of having a core product with a set of features as a basis [CHS10]. Later, to derive new products, different delta operations are applied to the core product [CHS10, LLL+15]. These delta operations consists of (1) adding new features, (2) removing features and (3) modifying features. Our algorithm adds suspicious features to the core features of the SPL and, subsequently, a propagation function adds required features in order to have a valid product. These operations can be considered as part of delta modeling approach since our algorithm has an initial product composed of the SPL core features. The algorithm is designed this way so that the propagation function increases efficiency. Otherwise, every time the propagation function is called, the core features would be added to derive a valid product.

To the best of our knowledge, SBFL has been applied in the SPL context only in a recent study [LWG+17]. Li et al. proposed a search-based approach that generates application engineering level test cases that can be easily reused between different products of the SPL [LWG+17]. Their approach integrates fault localization techniques with the aim to generate more effective test cases in locating bugs. However,

unlike in this study, they apply SBFL at the code level, whereas we propose the application of SBFL at the feature level to isolate feature sets containing faults.

Yilmaz et al. [YCP04, YCP06] focused on the generation and scheduling of configurations in configurable software (e.g., Linux) for efficient fault characterization. To this end, they proposed two kinds of covering arrays, namely, fixed-strength covering arrays and variable-strength covering arrays [YCP06]. Their empirical evaluation focuses on how different covering arrays perform in fault localization with two case studies. As expected, they found that higher strength covering arrays performed better than lower strength ones. In contrast with their approach, we propose a SBFL approach to locale faulty feature sets in SPLs following a model-based approach (using feature models). Additionally, we assess how different SBFL techniques perform in different test scenarios (i.e., different amount and types of faults, with different product suites). We think, however, that both approaches could be complementary: using their covering array algorithms to generate and prioritize product suites of different strengths, and allow for a faster fault localization in SPLs. Exploring this idea remains for future work.

As mentioned in Section 3.4.2, several empirical studies have been carried out to assess the performance between different SBFL techniques [PCJ+17, AZVG07, LTL13, WDGL14, JH05]. The subject programs of previous studies have always been the source code of program with different languages (e.g., C or Java). In contrast, we propose the application of SBFL in SPLs at the feature level, a context in which to the best of our knowledge, this technique has never been applied. We provided an empirical evaluation that compared ten different suspiciousness techniques in different fault scenarios, across nine case studies of different complexities. Unlike in previous studies, where Dstar has been found to be one of the best techniques, together with Ochiai, we found that in our context the best techniques are Tarantula, Ample2 and Kulcynski2. Moreover, we complement the use of SBFL in the SPL context with a fault isolation algorithm that provides the debugger with the smallest product to help isolate the faulty feature sets.

Many studies have proposed different techniques for pinpointing faults in computer programs. Simplifying large test cases that produce a fault by removing irrelevant details is the core idea of Delta Debugging [ZH02], a well-known fault isolation technique. Our fault isolation approach builds upon delta debugging by employing an incremental approach to build the minimal product (i.e., the core features are taken as a baseline, and the most suspicious feature sets are included to form the minimal product as possible) instead of a decremental approach (i.e., isolating fault by making the input space smaller). Moreover, our approach is designed for the SPL context, a

context where debugging has centered little attention. This SPL context faces several idiosyncrasies, such as the use of feature models to manage the variability and the use of reasoning techniques (e.g., SPLAR) to derive valid products.

## 9.6 Conclusion

In this chapter we presented a debugging approach for SPLs using SBFL techniques. Based on the features included on each product under test and the test outcomes, it is possible to identify which feature sets were involved in a failure, and which ones did not, narrowing the search for the faulty feature set that made the execution fail. As a result, feature sets are ranked according to their suspiciousness score, assisting debuggers on the localization of bugs. Additionally, we propose to exploit the techniques for the automated analysis of feature models to generate minimal valid products containing the suspicious feature sets, contributing to reduce the effort required to isolate and locate faults. We empirically evaluated our approach by comparing the effectiveness of ten SBFL techniques on nine case studies. Results show that the approach is effective, with the techniques Tarantula, Kulcynski2 and Ample2 showing a good and stable performance with different number and types of faults. We also found that the effectiveness of the technique increases with the number of products under test. This work complements the extensive corpus of papers on SPL testing, and paves the path for new contributions on fault localization in SPLs.

# Part V

# Final Remarks

Chapter 10

# Conclusion

This chapter concludes the thesis. Specifically, Section 10.1 summarizes the contributions, discusses the validation of the hypotheses and highlights the main limitations of the proposed solutions. Section 10.2 discusses a set of lessons learned we extracted from the thesis. Finally, short and mid-term future work are exposed in Section 10.3.

## 10.1 Summary of the Contributions

Testing configurable Cyber-Physical Systems (CPSs) is challenging, mainly due to the high amount of configurations that they can be set to. Developing CPSs prototypes is expensive, and as a result, simulation-based testing has been envisioned as an efficient means for testing CPSs [BNSB16]. In this dissertation, different solutions were proposed to cost-effectively test configurable CPSs in an automated manner.

The first contribution of the dissertation corresponds to a tool supported methodology that automatically generates a test system for each of the products that must be tested in a configurable CPS employing the tool Simulink. The prototypical version of the tool employs feature models to manage the variability of both, the configurable CPS and the test system. This allows for a systematic generation of each test system instance in an automated manner within a few seconds. Furthermore, it reduces the error proneness. However, an investment of time must be employed by domain engineers to develop the test feature model and all the test assets (e.g., test oracles).

The second contribution of this thesis corresponds to a multi-objective search approach to generate a test suite composed of reactive test cases that can be employed to test CPSs. Four cost-effectiveness measures are defined and corresponding crossover and mutation operators are developed. Furthermore, the algorithm returns test suites in a prioritized manner considering the similarity of test cases, which would allow for a faster fault detection when historical data is not available. Results indicated that Non-dominated Sorting Genetic Algorithm II (NSGA-II) was the best algorithm for solving the proposed problem.

To further optimize the test process of configurable CPSs, the third contribution consisted of a test case selection method based on weight-based search algorithms that cost-effectively selects test cases for testing configurations of the configurable system at different test levels (i.e., Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL)). Corresponding test case selection objectives were defined for each of the test levels. An empirical evaluation with two case studies suggested that the Genetic Algorithm (GA) was the best algorithm for solving this problem.

Apart from test selection, test case prioritization was also investigated to cost-effectively test configurable CPSs, which is considered the fourth contribution of this dissertation. In this case we also employed weight-based search algorithms for this purpose. As in the case of test case selection, for each of the test levels, specific test objectives were defined. An empirical evaluation with four case studies suggested that local search algorithms are better than global search algorithms for solving the test case prioritization problem. Specifically, the Alternating Variable Method (AVM) algorithm showed the best performance when reducing the fault detection time and the simulation time, whereas Additional Greedy was the best algorithm for reducing the time to cover functional and non-functional requirements.

A problem when testing configurable systems, such as Software Product Lines (SPLs) or configurable CPSs is that locating faults is extremely difficult. To solve this problem, we adapted Spectrum-Based Fault Localization (SBFL) to the product line engineering context and proposed a fault isolation method. We performed an empirical evaluation with eight case studies and compared ten SBFL state-of-the-art techniques. The empirical evaluation suggested that the proposed method is appropriate for locating bugs in configurable systems. Furthermore, three of the selected techniques (i.e., Kulcynski2, Tarantula and Ample2) performed better than the rest.

Overall, this thesis proposes methods to systematically and cost-effectively test configurable CPSs. We believe that the proposed methods advance the current practice in automation, optimization and debugging of configurable systems, with an emphasis on CPSs.

### 10.1.1 Hypotheses Validation

We stated four research hypotheses in Section 4.2. This section analyses each of the contributions and argues whether the stated hypotheses can be validated.

### First hypothesis

The first hypothesis is stated as follows: “The use of variability models (e.g., feature models) helps the systematic generation of test system instances for testing configurable CPSs, which is faster than a manual generation”. To test this hypothesis we proposed a methodology supported by a tool, named ASTERYSCO. Specifically, as variability modeling notation we employed features models, which were later processed by ASTERYSCO to generate test systems in Simulink. We evaluated the tool employing the case study presented in Section 4.4.1, which involves the Simulink model of an Unmanned Aerial Vehicle (UAV). This case study was the largest available in terms of Simulink blocks. For the evaluation, we compared the differences of generating test system instances with our methodology and a manual approach. The main drawback of employing our proposed method involves that an investment of time is required by test engineers to develop the test feature model and the domain test assets. However, once developed these items, the following steps were much faster than manual test system generation. In fact, we generated the test system instance of manually derived 10 configurations of different sizes and automatically derived (by a pairwise algorithm) 28 configurations. The generation of the generic test system was about 3.85 seconds, while the time required by ASTERYSCO to instantiate the generic test system into a configuration-specific test system was around 3 seconds on average. Considering this, it can be assumed that the stated first hypothesis has been validated.

### Second hypothesis

The second hypothesis is stated as follows: “The use of multi-objective search algorithms permits the generation of cost-effective test cases in CPSs testing”. To test this hypothesis, we proposed a test generation approach based on multi-objective search algorithms to generate reactive test cases. The evaluation was performed with four CPSs of different characteristics, and five search algorithms were assessed, taking Random Search (RS) as baseline algorithm. The proposed search algorithms outperformed in general RS. Furthermore, among the selected five multi-objective search algorithms, NSGA-II was the best one. In fact, in terms of the Hypervolume (HV) quality indicator, on average, NSGA-II outperformed RS in 49.25 %. As for individual objectives, NSGA-II outperformed RS in all the objectives and case studies. Specifically, on average, requirements coverage was improved in 51.74%, test execution time in 61.96%, similarity in 29.86% and prioritization-aware similarity in 30.65%. Considering the performed empirical evaluation, we can conclude that the stated second hypothesis

has been validated.

### Third hypothesis

The third hypothesis is stated as follows: "The use of search algorithms permits optimization of the test process of configurable CPSs employing simulation by selecting and prioritizing test cases". To test this hypothesis we proposed two methods: (1) a test case selection method and (2) a test case prioritization method. Both of them were based on weight-based search algorithms. Since simulation-based testing involves three test levels, corresponding test objectives were proposed for each of the test levels. As for test case selection, two case studies were employed in an empirical evaluation with 75 artificial problems in total. Three search algorithms were assessed, GA as a representative of global search algorithms and additional greedy as well as AVM as representative of local search algorithms. As a baseline algorithm, RS was selected. The empirical evaluation suggested that in general GA performed better than the rest of algorithms, managing to reduce the overall test execution time by 80% as compared with RS, while maintaining the overall test quality. As for test case prioritization, four case studies were employed in an empirical evaluation with 570 artificial problems. In this case, two local search algorithms (AVM and additional greedy) and two global search algorithms (Weight-Based Genetic Algorithm (WBGA) and Randomly-Weighted Genetic Algorithm (RWGA)) were employed, whereas RS was taken as a baseline algorithm. In general, local search algorithms managed to perform better than global search algorithms. For faster fault detection and simulation time, AVM performed best, whereas for requirements covering time reduction Additional Greedy performed best. On average, the selected algorithms improved the faults detection time in up to 60.66% as compared with RS, the simulation time in up to 37.74%, the functional requirements covering time in up to 84.32% and the non-functional requirements covering time in up to 86.31%. Considering the empirical evaluation of both, test case selection and test case prioritization approaches, we believe that the third hypothesis has been validated.

### Fourth hypothesis

The fourth hypothesis is stated as follows: "The adaption of SBFL techniques to the product line engineering context permits the localization and isolation of faulty features". To test this hypothesis we have adapted SBFL to the product line engineering context and proposed an algorithm that returns products of minimal size, which allows for the efficient isolation of faulty features. We empirically evaluated the approach on 8 product line models from the SPLOT repository [MBC09b] and compared 10 state-

of-the-art SBFL techniques. We found that three of them (Tarantula, Kulcynski2 and Ample2) performed in general better than the remaining ones. The hardest scenario for locating bugs in configurable systems was when multiple feature interaction faults exist. For that case, for product suites derived following a pairwise approach, on average, Tarantula required examining 5.55% of feature sets, Kulcynski2 6.27% and Ample2 5.45%. These values were further improved with product suites derived following a 3-wise approach as well as with other types of faults. We believe that these values are quite good for locating bugs in configurable systems, and thus, it can be concluded that the fourth hypothesis has been validated.

### 10.1.2 Limitations of the Proposed Solutions

This section discusses some of the limitations that the proposed solutions might have when applying them in practice. One such limitation can be the selected variability modeling notation. In our case, we selected feature modeling due to its wide use in industry [BRN+13]. Specifically, we employed FeatureIDE [TKB+14], since it is an open source, highly intuitive and quite powerful feature modeling tool. Nevertheless, researchers have found limitations in feature modeling for managing variability of CPS product lines [SYAL16]. To this end, new variability notations for CPSs have been envisioned [KNK+17]. When using FeatureIDE we did not find any limitation, with the exception of that related to parameterizable variables, which can be managed directly with MATLAB files. It is important to highlight that in our case, we have employed MATLAB/Simulink for modeling and simulation of CPSs. However, we have to take into account the limitations of feature models and be ready to evolve our tools to be able to handle new variability modeling notation proposals.

The proposed tools and methods have been implemented in MATLAB, and we have used Simulink for modeling and simulating CPSs. This decision was made due to the wide use in industry of MATLAB/Simulink, which is becoming a prevalent modeling language for CPSs [MNBB16]. However, other tools such as Dymola or Modelica are also being employed to simulate CPSs, and current trends are moving towards co-simulation, which allows the interconnection of different tools. In fact, in Mondragon University, we have recently developed co-simulation tools for the vertical transport domain [SAM+17] as well as for intelligent buildings [ELM+17]. While the proposed test case selection and prioritization approaches could be easily adapted for using them in a co-simulation environment, ASTERYSCO, which is employed to generate test system instances in Simulink, could require further efforts to adapt it. To ensure a future adaption of ASTERYSCO either, for another simulation tool or for a co-simulation environment, we have tried to correctly document all the code of the

tool in order to be as maintainable as possible.

As for the proposed multi-objective test generation approach, we employed four objectives to cost-effectively generate and prioritize test suites composed of reactive test cases. While the cost function was handled by the test execution time objective, the effectiveness of test suites was measured by (1) the requirements coverage and (2) two similarity functions. These similarity functions were defined based on previous works that demonstrated that dissimilar test cases have a higher likelihood for detecting faults [FPCY16, HB10, HAB13]. Thus, we have assumed when evaluating the approach that a more dissimilar test suite is better at finding faults. However, to ensure this, further empirical evaluation is required including mutation testing, which is a technique that has been demonstrated as a good substitute for real faults [JJI+14]. However, unfortunately, an empirical evaluation with mutation testing was unfeasible in the context of this study, basically due to resource problems. As explained before, mutation testing of CPSs is highly expensive because the physical layer is computationally very costly to simulate. Furthermore, our approach returned a set of solutions in a pareto-front. Evaluating the whole pareto-front would require a lot of computational effort, which we foresee to handle in the future by employing several clusters for the evaluation of the fault detection capability of the proposed test case generation approach.

A similar issue happened for both, the test case selection and prioritization approaches. In these cases, we could not evaluate multi-objective search algorithms due to the cost they would require when evaluating the whole pareto-front employing mutation testing. Furthermore, when evaluating solutions for the HiL test level, a real HiL environment could not be employed, basically because these environments employ real-time emulation of the system, and it would be too costly to execute all the algorithm runs in such environments. Instead, we simulated them employing a SiL environment. The execution of test cases at a HiL test environment would require some adaptations, such as a connection between the computer in charge of selecting and prioritizing test cases and the hardware platform executing test cases.

Another limitation of the test case selection and prioritization metrics involves the historical database. Notice that this database includes historical data involving the Fault Detection Capability (FDC) of each test case, which is obtained by testing other configurations. At the beginning of the testing process, when there have not been execution of test cases, there is no data related to the FDC. To overcome this problem, the test case generation approach presented in Chapter 5 also considers test prioritization based on test similarities. Furthermore, to initialize the FDC of test cases, initially, mutation testing can be employed, i.e., a configuration can be derived

and some mutants generated for this configuration. Later, test cases can be executed in order to observe which ones detect and which not each of the mutants. This would enable to initially assess the quality of each test case.

For the debugging part, it is important to highlight that in order this technique to be effective, both, test cases and test oracles have to be good at detecting faults. Otherwise, information can be missed when building the coverage matrix and calculating the suspiciousness scores of each feature set. Unfortunately, this is an issue that cannot be always ensured. Furthermore, we must be ready to adapt SBFL to other variability notations (e.g., Clafer [BDA+15], SimPL [BYBS13], etc.).

## 10.2 Lessons Learned

This section summarizes lessons learned from the research carried out during this Ph.D. thesis. These lessons can be employed as a guidelines either, by researchers or industrial practitioners.

- *Simulation-based testing is an efficient and practical method for testing configurable CPSs:* Advantages of simulation-based testing include (1) execution of larger test suites, (2) easiness to integrate test oracles for the automated validation and (3) replication of safety-critical functionalities that in a non-virtual environment are extremely expensive to reproduce. In the last few years, several research publications have shown the increasing trend of simulation-based testing in industry, especially in the automotive domain [MNB+15, MNBB16, MNB17, BANBS16]. Highly reputed researchers have recently envisioned the use of simulation-based testing along with search-based methods as efficient means for testing CPSs [BNSB16]. In this thesis we have focused on CPSs that are highly configurable. In this case, the use of simulation-based testing is highly important, especially because many configurations need to be tested and thus, automation and optimization is highly recommended for the systematic validation of these systems. Thus, we highly recommend practitioners from industry to involve simulation when testing their configurable CPSs.

- *Investment in the domain engineering level eases the systematic test and validation processes of configurable CPSs:* The philosophy of product line engineering relies on investing a higher amount of time by domain engineers developing variability-handling assets to ease application engineers setting up a specific configuration. In this thesis we have demonstrated that this philosophy can be extrapolated to configurable CPSs testing. The use of a test feature model incorporating informa-

tion related to the test assets permits fully automating the process of test system generation, test case generation, test case selection and prioritization. We highly recommend practitioners to involve a variability management notation when testing their configurable systems. In fact, this philosophy has been already acquired by Orona, which is one of our industrial partners, to automatically test highly configurable refactored code [SEA+17].

- *Search-Based Software Engineering (SBSE) approaches are effective at solving CPSs testing problems:* One of the lessons learned in this thesis is that SBSE methods (e.g., test case generation, test case selection and test case prioritization) are effective for solving CPSs methods. In fact, we have tested multi-objective algorithms for test generation, and weight-based search algorithms (both, global and local search algorithms) for test case selection and prioritization. These problems are complex to solve, as demonstrated in empirical evaluations, where RS was taken as a baseline algorithm. Furthermore, while this thesis was being developed, several publications appeared both, in well reputed journals as well as conferences, showing effectiveness of SBSE methods for testing CPSs (e.g., [MNB+15, MNBB16, MNB17, BANBS16]).

- *SBFL is effective at the software level, but also at the feature level of configurable systems:* We adapted the technique SBFL to the product line engineering context to localize faults in configurable systems. We have shown that SBFL is not only good at localizing bugs in software, as demonstrated by many studies (e.g., [HRS+00, AZVG07, AZGvG09, WDGL14, WGL+16, PCJ+17]), but also at the feature level of configurable systems. This could permit not only localizing software faults but also others, such as faults due to drivers, sensors, communication systems, as well as feature interaction faults, which are typical SPL faults [SSRC14a, KKLH09].

- *Artificial problems permit reducing threats to validity and assessing the scalability of problems in empirical evaluations related to CPS engineering:* One typical external validity threat for software engineering studies involves the use of few case studies. This threat is often difficult to mitigate since there are not always available case studies. In our empirical evaluations, we have divided the problems in several artificial problems by, for instance, using different amount of test cases, employing different types and number of faults, different product configurations, etc. This permits comparing different algorithms and techniques at different situations, in addition to assessing the scalability of the approaches (e.g., how the techniques work with more test cases). Not only that, but also, we have found that dividing the

evaluation in several problems is typically well seen by reviewers when trying to publish papers.

## 10.3 Perspectives and Future Work

In this section we summarize the short and medium term objectives to complement this work from three perspectives (i.e., industry transfer, application of the proposed methods in specific domains and further research).

### 10.3.1 Industry Transfer

The research performed by the Engineering School of Mondragon is industry oriented. We are currently in contact with several industrial companies to transfer them the proposed methods. Hitachi is currently evaluating ASTERYSCO, our test system generation tool presented in Chapter 5 to apply it in their automotive systems. We have already done a meeting with Orona, which is one of the most powerful lifting companies in Europe, to present them our test case selection and prioritization approaches. Alerion technologies, which is a small company constructing UAVs, also showed interest in the test case generation and prioritization approaches presented in this thesis; we plan to transfer them our methods during the TESTOMAT project[1] to perform the automated validation of their UAVs. Furthermore, in the scope of the TESTOMAT project, Ulma Embedded Solution will be involved in adding tool support to our test case prioritization algorithms for testing UAVs in an agile framework.

### 10.3.2 Application of the Proposed Methods in Specific Domains

The proposed methods in this dissertation are generic for any configurable CPS. However, it is worth mentioning that each system has its own particularities. For instance, Matinnejad et al. focused on testing CPSs from controllers perspective [MNB$^+$15], where test cases are generated with the objective of violating controllers specifications. Ben Abdessalem et al. employed multi-objective search algorithms to generate test cases for automated driving vehicles [BANBS16]. In this case, the test case generation is guided towards finding collisions between the vehicle and objects (e.g., pedestrians). These particularities can be also considered and integrated within the proposed methods in this dissertation.

[1] http://front4.itea3.eu/project/testomatproject.html

In the future we seek to adapt some of the methods proposed in this thesis within the context of the Horizon 2020 HiFi-Elements project. In this project, electrical vehicles at system and component level will be tested. We foresee to integrate the proposed test case generation, selection and prioritization methods considering particularities of electrical vehicles and their components.

### 10.3.3 Further Research

Further research as well as new developments can be performed to complement this work.

As for short-term future work, an ongoing Ph.D. project is continuing with this work, which aims at developing other optimization methods for testing configurable CPSs. Preliminary results have been published at the SPLC 2017 conference [MASE17], where an iterative test allocation approach is proposed based on search algorithms to optimally test configurable CPSs.

As highlighted in the limitations sections, despite feature models not showing important problems in our case for modeling CPSs in Simulink, other works have claimed limitations of the feature modeling technique for the CPS engineering context [SYAL16]. Future directions in the SPL engineering community envision novel variability modeling methods for CPSs [KNK+17]. As a mid-term future work, we plan to develop new parsers to integrate other variability modeling notations in our tool chain.

Regarding the first contribution of this thesis, involving the automatic generation of test system instances, in the future, we would like to adapt our tool-supported methodology for other simulation tools (e.g., Modelica) as well as co-simulation environments (e.g., Building Controls Virtual Test Bed (BCVTB)). Adapting ASTERYSCO to the BCVTB environment would allow for the distributed validation of configurable CPSs, which is an idea that we would like to implement.

As for the second contribution, which involves the automatic generation of reactive test cases for testing CPSs, in the future we would like to do a more comprehensive empirical evaluation. This would include more case studies, more search algorithms, other search objectives (e.g., white-box coverage) as well as mutation testing for the evaluation of the fault revealing ability of the generated test suites. Furthermore, the requirements coverage function is developed for each of the study subjects with a script function. We would like to investigate a formal approach to link reactive behaviors of the system with functional requirements to automatically generate these scripts and make the process of generating test cases more systematic.

Regarding the third and fourth contribution, due to resource problems, we were unable to use pareto-based search algorithms in our evaluation of the test case selection and prioritization approaches. This is because when evaluating the fault revealing capability of our methods we employed mutation testing, which it has been shown as a good substitute of real faults [JJI+14]. However, notice that in our context, when generating mutants of the CPSs models, both, the physical and the cyber layer had to be simulated. The physical layer of these systems is usually modeled with complex mathematical models, and executing simulations of so many mutants was extremely time consuming. In the medium-term future we would like to test pareto-based algorithms for selecting and prioritizing test cases for the CPS context. Furthermore, as stated in the limitations section, one drawback of our test case selection and prioritization approach involves the initial startup of the test history to obtain the FDC metric. As a short-term objective, we are planning to empirically evaluate black-box metrics for test case selection and prioritization that would not require a historical database.

Finally, regarding the fifth contribution, which involves debugging configurable systems, we would like to theoretically derive new SBFL techniques that better fit into the scope of product line engineering. Furthermore, with the derived techniques we would like to expand on the empirical evaluation by incorporating larger product line models.

# Bibliographic References


[AAPV09] Luciano C Ascari, Lucilia Y Araki, Aurora RT Pozo, and Silvia R Vergilio. Exploring machine learning techniques for fault localization. In *Test Workshop, 2009. LATW'09. 10th Latin American*, pages 1–6. IEEE, 2009.

[AB11] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 1–10. IEEE, 2011.

[Abb15] Houssam Y Abbas. *Test-based falsification and conformance testing for cyber-physical systems*. PhD thesis, 2015.

[ABHPW10] Shaukat Ali, Lionel C Briand, Hadi Hemmati, and Rajwinder Kaur Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering*, 36(6):742–762, 2010.

[AF13] Andrea Arcuri and Gordon Fraser. Parameter tuning or default values? an empirical investigation in search-based software engineering. *Empirical Software Engineering*, 18(3):594–623, 2013.

[AF14] Andrea Arcuri and Gordon Fraser. On the effectiveness of whole test suite generation. In *International Symposium on Search Based Software Engineering*, pages 1–15. Springer, 2014.

[AHF$^+$14] Houssam Abbas, Bardh Hoxha, Georgios Fainekos, Jyotirmoy V Deshmukh, James Kapinski, and Koichi Ueda. Wip abstract: Conformance testing as falsification for cyber-physical systems. In *2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2014.

[AHF+17] M Moein Almasi, Hadi Hemmati, Gordon Fraser, Andrea Arcuri, and Jānis Benefelds. An industrial evaluation of unit test generation: Finding real faults in a financial application. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*, pages 263–272. IEEE Press, 2017.

[AHFU14] Houssam Abbas, Bardh Hoxha, Georgios Fainekos, and Koichi Ueda. Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on*, pages 1–6. IEEE, 2014.

[AHTL+16] Mustafa Al-Hajjaji, Thomas Thüm, Malte Lochau, Jens Meinicke, and Gunter Saake. Effective product-line testing using similarity-based product prioritization. *Software & Systems Modeling*, pages 1–23, 2016.

[AHTM+14] Mustafa Al-Hajjaji, Thomas Thüm, Jens Meinicke, Malte Lochau, and Gunter Saake. Similarity-based prioritization in software product-line testing. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 197–206, New York, NY, USA, 2014. ACM.

[AIB10] Andrea Arcuri, Muhammad Zohaib Iqbal, and Lionel Briand. Black-box system testing of real-time embedded systems using random and search-based testing. In *Proceedings of the 22Nd IFIP WG 6.1 International Conference on Testing Software and Systems*, ICTSS'10, pages 95–110, Berlin, Heidelberg, 2010. Springer-Verlag.

[AKD+10] Shay Artzi, Adam Kiezun, Julian Dolby, Frank Tip, Daniel Dig, Amit Paradkar, and Michael D Ernst. Finding bugs in web applications using dynamic test generation and explicit-state model checking. *IEEE Transactions on Software Engineering*, 36(4):474–494, 2010.

[ALW+17] Shaukat Ali, Hong Lu, Shuai Wang, Tao Yue, and Man Zhang. Uncertainty-wise testing of cyber-physical systems. *Advances in Computers*, 2017.

[ARM16] Arend Aerts, Michel A. Reniers, and Mohammad Reza Mousavi. Model-based testing of cyber-physical systems. In *Cyber-Physical*

*Systems : Foundations, Principles and Applications*, pages 287–304. Cyber-Physical Systems: Foundations, Principles and Applications, 2016.

[ASE14a] Aitor Arrieta, Goiuria Sagardui, and Leire Etxeberria. A comparative on variability modelling and management approaches in simulink for embedded systems. In *V Jornadas de Computación Empotrada*, number 26-33 in JCE 2014, 2014.

[ASE14b] Aitor Arrieta, Goiuria Sagardui, and Leire Etxeberria. A configurable test architecture for the automatic validation of variability-intensive cyber-physical systems. In *VALID 2014: The Sixth International Conference on Advances in System Testing and Validation Lifecycle*, pages 79–83, 2014.

[ASE14c] Aitor Arrieta, Goiuria Sagardui, and Leire Etxeberria. Towards the automatic generation and management of plant models for the validation of highly configurable cyber-physical systems. In *Proceedings of 2014 IEEE 19th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8, 2014.

[ASE15a] Aitor Arrieta, Goiuria Sagardui, and Leire Etxeberria. Test control algorithms for the validation of cyber-physical systems product lines. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 273–282, New York, NY, USA, 2015. ACM.

[ASE15b] Aitor Arrieta, Goiuria Sagardui, and Leire Etxeberria. Variability in test systems: Review and challenges. In *International Conference on Advances in System Testing and Validation Lifecycle, VALID*, 2015.

[ASEZ17] Aitor Arrieta, Goiuria Sagardui, Leire Etxeberria, and Justyna Zander. Automatic generation of test system instances for configurable cyber-physical systems. *Software Quality Journal*, 25(3):1041–1083, 2017.

[ATF09] Wasif Afzal, Richard Torkar, and Robert Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[AWM+17] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. Search-based test case generation for cyber-physical systems. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pages 688–697, 2017.

[AWSE16a] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. Search-based test case selection of cyber-physical system product lines for simulation-based validation. In *Proceedings of the 20th International Systems and Software Product Line Conference*, pages 297–306, 2016.

[AWSE16b] Aitor Arrieta, Shuai Wang, Goiuria Sagardui, and Leire Etxeberria. Test case prioritization of configurable cyber-physical systems with weight-based search algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 1053–1060, New York, NY, USA, 2016. ACM.

[AY15] Shaukat Ali and Tao Yue. U-test: Evolving, modelling and testing realistic uncertain behaviours of cyber-physical systems. In *8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015*, pages 1–2, 2015.

[AYZ16] Shaukat Ali, Tao Yue, and Man Zhang. Tackling uncertainty in cyber-physical systems with automated testing. *Ada User Journal*, 37(4), 2016.

[AZGvG09] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J.C. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780 – 1792, 2009. SI: {TAIC} {PART} 2007 and {MUTATION} 2007.

[AZVG07] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007*, pages 89–98. IEEE, 2007.

[BANBS16] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing advanced driver assistance systems using multi-objective search and neural networks. In *Proceedings of the 31st*

*IEEE/ACM International Conference on Automated Software Engineering*, pages 63–74, 2016.

[Bat05] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines Conference (SPLC)*, volume 3714 of *Lecture Notes in Computer Sciences*, pages 7–20. Springer–Verlag, 2005.

[BCVP11] Pierre-Jean Bristeau, François Callou, David Vissière, and Nicolas Petit. The navigation and control technology inside the ar.drone micro uav, 2011.

[BDA+15] Kacper Bak, Zinovy Diskin, Michal Antkiewicz, Krzysztof Czarnecki, and Andrzej Wasowski. Clafer: unifying class and feature modeling. *Software & Systems Modeling*, pages 1–35, 2015.

[BEG12] Ebrahim Bagheri, Faezeh Ensan, and Dragan Gasevic. Grammar-based test generation for software product line feature models. In *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '12, pages 87–101, Riverton, NJ, USA, 2012. IBM Corp.

[Beh12] Razieh Behjati. *A Model-Based Approach to the Software Configuration of Integrated Control Systems*. PhD thesis, University of Oslo, 2012.

[BHM+10] Angelo Brillout, Nannan He, Michele Mazzucchi, Daniel Kroening, Mitra Purandare, Philipp Rümmer, and Georg Weissenbacher. Mutation-based test case generation for simulink models. In *Formal Methods for Components and Objects*, pages 208–227. Springer, 2010.

[BK08] Eckard Bringmann and Andreas Kramer. Model-based testing of automotive systems. In *Proceedings of the 1st International Conference on Software Testing, Verification and Validation*, pages 485 – 493, Lillehammer, Norway, 2008.

[BLL07] Lionel C Briand, Yvan Labiche, and Xuetao Liu. Using machine learning to support debugging with tarantula. In *Software Reliability, 2007. ISSRE'07. The 18th IEEE International Symposium on*, pages 137–146. IEEE, 2007.

[BNR03] Thomas Ball, Mayur Naik, and Sriram K Rajamani. From symptom to cause: localizing errors in counterexample traces. In *ACM SIGPLAN Notices*, volume 38, pages 97–105. ACM, 2003.

[BNSB16] Lionel Briand, Shiva Nejati, Mehrdad Sabetzadeh, and Domenico Bianculli. Testing the untestable: Model testing of complex software-intensive systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 789–792. ACM, 2016.

[BRN+13] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Kryzsztof Czarnecki, and Andrzej Wasowski. A survey of variability modeling in industrial practice. In *Variability Modelling of Software-intensive Systems (VaMoS)*, pages 7:1–7:8, 2013.

[Bro12] J Brownlee. *Cleverl Algorithms: Nature-Inspired Programming Recipes*. lulu.com, 2012.

[BSRC10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615 – 636, 2010.

[BT+15] Nguyen Thanh Binh, Khuat Thanh Tung, et al. A novel test data generation approach based upon mutation testing by using artificial immune system for simulink models. In *Knowledge and Systems Engineering*, pages 169–181. Springer, 2015.

[BTWV15] Loli Burgueno, Javier Troya, Manuel Wimmer, and Antonio Vallecillo. Static fault localization in model transformations. *IEEE Transactions on Software Engineering*, 41(5):490–506, 2015.

[BX16] Benjamin Busjaeger and Tao Xie. Learning for test prioritization: an industrial case study. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 975–980. ACM, 2016.

[BYBS13] Razieh Behjati, Tao Yue, Lionel Briand, and Bran Selic. Simpl: A product-line modeling methodology for families of integrated control systems. *Information and Software Technology*, 55(3):607 – 629, 2013. Special Issue on Software Reuse and Product Lines.

[CDFR08] Peggy Cellier, Mireille Ducassé, Sébastien Ferré, and Olivier Ridoux. Formal concept analysis enhances fault localization in software. *Lecture Notes in Computer Science*, 4933:273–288, 2008.

[CDFR11] Peggy Cellier, Mireille Ducassé, Sébastien Ferré, and Olivier Ridoux. Multiple fault localization with data mining. In *SEKE*, pages 238–243, 2011.

[CDS08] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Trans. Softw. Eng.*, 34(5):633–650, 2008.

[CGF+17] Jose Campos, Yan Ge, Gordon Fraser, Marcello Eler, and Andrea Arcuri. An empirical evaluation of evolutionary algorithms for test suite generation. In *Symposium on Search-Based Software Engineering*, 2017.

[CGS04] Sagar Chaki, Alex Groce, and Ofer Strichman. Explaining abstract counterexamples. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 73–82. ACM, 2004.

[CHS10] Dave Clarke, Michiel Helvensteijn, and Ina Schaefer. Abstract delta modeling. In *Proceedings of the Ninth International Conference on Generative Programming and Component Engineering*, GPCE '10, pages 13–22, New York, NY, USA, 2010. ACM.

[CJKO01] David W Corne, Nick R Jerram, Joshua D Knowles, and Martin J Oates. Pesa-ii: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 283–290. Morgan Kaufmann Publishers Inc., 2001.

[CLM04] Tsong Yueh Chen, Hing Leung, and IK Mak. Adaptive random testing. In *ASIAN*, volume 4, pages 320–329. Springer, 2004.

[CM13] Cagatay Catal and Deepti Mishra. Test case prioritization: A systematic mapping study. *Software Quality Journal*, 21(3):445–478, September 2013.

[CN01] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, August 2001.

[dCMMCDA14] Ivan do Carmo Machado, John D McGregor, Yguaratã Cerqueira Cavalcanti, and Eduardo Santana De Almeida. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology*, 56(10):1183–1199, 2014.

[DJ14] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, part i: Solving problems with box constraints. *IEEE Trans. Evolutionary Computation*, 18(4):577–601, 2014.

[DLPW08] Christian Dziobek, Joachim Loew, Wojciech Przystas, and Jens Weiland. Functional variants handling in simulink models. Technical report, Mathworks, 2008.

[DLSV11] Patricia Derler, Edward A. Lee, and Alberto Sangiovanni-Vincentelli. Modeling cyber-physical systems. *Proceedings of the IEEE (special issue on CPS)*, 100(1):13 – 28, January 2011.

[DN11] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760 – 771, 2011.

[DPAM02] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[DPC+14] Xavier Devroey, Gilles Perrouin, Maxime Cordy, Pierre-Yves Schobbens, Axel Legay, and Patrick Heymans. Towards statistical prioritization for software product lines testing. In *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, page 10. ACM, 2014.

[DPC+15] Xavier Devroey, Gilles Perrouin, Maxime Cordy, Hamza Samih, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. Statistical prioritization for software product line testing: an experience report. *Software & Systems Modeling*, pages 1–19, 2015.

[DSLL13] Michael Dukaczewski, Ina Schaefer, Remo Lachmann, and Malte Lochau. Requirements-based delta-oriented spl testing. In *4th International Workshop on Product LinE Approaches in Software Engineering, PLEASE 2013*, pages 49 – 52, San Francisco, CA, United states, 2013.

[EBA+11] Alireza Ensan, Ebrahim Bagheri, Mohsen Asadi, Dragan Gasevic, and Yevgen Biletskiy. Goal-oriented test case selection and prioritization for product line feature models. In *Proceedings of the 2011 Eighth International Conference on Information Technology: New Generations*, ITNG '11, pages 291–298, Washington, DC, USA, 2011. IEEE Computer Society.

[EBG12] Faezeh Ensan, Ebrahim Bagheri, and Dragan Gašević. Evolutionary search-based test generation for software product line feature models. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering*, CAiSE'12, pages 613–628, Berlin, Heidelberg, 2012. Springer-Verlag.

[ELM11] John C. Eidson, Edward A. Lee, and Slobodan Matic. Distributed real-time software for cyber-physical systems. *Proceedings of the IEEE*, 100:45–59, 2011.

[ELM+17] Leire Etxeberria, Felix Larrinaga, Urtzi Markiegi, Aitor Arrieta, and Goiuria Sagardui. Enabling co-simulation of smart energy control systems for buildings and districts. In *IEEE 22nd Conference on Emerging Technologies and Factory Automation (ETFA2017)*, pages 1–4, 2017.

[ER11] Emelie Engström and Per Runeson. Software product line testing–a systematic mapping study. *Information and Software Technology*, 53(1):2–13, 2011.

[ERP14] Sebastian Elbaum, Gregg Rothermel, and John Penix. Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*, pages 235–245. ACM, 2014.

[ERS10] Emelie Engström, Per Runeson, and Mats Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, 2010.

[EYHB15] Michael G. Epitropakis, Shin Yoo, Mark Harman, and Edmund K. Burke. Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 234–245, New York, NY, USA, 2015. ACM.

[FA11] Gordon Fraser and Andrea Arcuri. Evolutionary generation of whole test suites. In *Quality Software (QSIC), 2011 11th International Conference on*, pages 31–40. IEEE, 2011.

[FA13] Gordon Fraser and Andrea Arcuri. Whole test suite generation. *IEEE Transactions on Software Engineering*, 39(2):276–291, 2013.

[fam13a] FaMa Tool Suite. http://www.isa.us.es/fama/, Accessed November 2013.

[FAM13b] Gordon Fraser, Andrea Arcuri, and Phil McMinn. Test suite generation with memetic algorithms. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1437–1444. ACM, 2013.

[FPCY16] Robert Feldt, Simon M. Poulding, David Clark, and Shin Yoo. Test set diameter: Quantifying the diversity of sets of test cases. In *2016 IEEE International Conference on Software Testing, Verification and Validation, ICST 2016, Chicago, IL, USA, April 11-15, 2016*, pages 223–233, 2016.

[FRC81] Kurt Fischer, Farzad Raji, and Andrew Chruscicki. A methodology for retesting modified software. In *Proceedings of the National Telecommunications Conference B-6-3*, pages 1–6, 1981.

[Fre14] FreeRTOS. Freertos. http://www.freertos.org/, 2014.

[GCKS06] Alex Groce, Sagar Chaki, Daniel Kroening, and Ofer Strichman. Error explanation with distance metrics. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(3):229–247, 2006.

[GHS92] Rajiv Gupta, Mary Jean Harrold, and Mary Lou Soffa. An approach to regression testing using slicing. In *Software Maintenance, 1992. Proceerdings., Conference on*, pages 299–308. IEEE, 1992.

[GLJZ12] Liang Gong, David Lo, Lingxiao Jiang, and Hongyu Zhang. Diversity maximization speedup for fault localization. In *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on*, pages 30–39. IEEE, 2012.

[GR04] D Greer and G Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243 – 253, 2004.

[GSB10] Andreas Griesmayer, Stefan Staber, and Roderick Bloem. Fault localization using a model checker. *Software Testing, Verification and Reliability*, 20(2):149–173, 2010.

[HAB13] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology*, 22(1):6:1–6:42, 2013.

[Har11] Mark Harman. Making the case for morto: Multi objective regression test optimization. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pages 111–114. IEEE, 2011.

[HB10] Hadi Hemmati and Lionel Briand. An industrial investigation of similarity measures for model-based test case selection. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 141–150. IEEE, 2010.

[HBT16] Le Thi My Hanh, Nguyen Thanh Binh, and Khuat Thanh Tung. A novel fitness function of metaheuristic algorithms for test data generation for simulink models based on mutation analysis. *Journal of Systems and Software*, 120(C):17–30, 2016.

[HFM15] H. Hemmati, Z. Fang, and M. V. Mäntylä. Prioritizing manual test cases in traditional and rapid release environments. In *Proceedings of the 8th International Conference on Software Testing, Verification and Validation (ICST'15)*, pages 1–10, 2015.

[HJK+14] Mark Harman, Yue Jia, Jens Krinke, William B Langdon, Justyna Petke, and Yuanyuan Zhang. Search based software engineering for software product line engineering: a survey and directions for future work. In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 5–18. ACM, 2014.

[HLL+16] Robert M Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. Sip: Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(2):17, 2016.

[HMZ12] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, December 2012.

[HN+88] Robert Hecht-Nielsen et al. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988.

[HPH+16] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. Comparing white-box and black-box test prioritization. In *Proceedings of the 38th International Conference on Software Engineering*, pages 523–534. ACM, 2016.

[HPHT15] Christopher Henard, Mike Papadakis, Mark Harmany, and Yves Le Traon. Combining multi-objective search and constraint solving for configuring large scale software product lines. In *37th International Conference on Software Engineering (ICSE'15)*, pages 517–528, 2015.

[HPLT14] Christopher Henard, Mike Papadakis, and Yves Le Traon. Mutation-based generation of software product line test configurations. In *International Symposium on Search Based Software Engineering*, pages 92–106. Springer, 2014.

[HPP+14] C Henard, M Papadakis, G Perrouin, J Klein, P Heymans, and Y Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Trans. Software Eng.*, 40(7):650–670, 2014.

[HRS+00] Mary Jean Harrold, Gregg Rothermel, Kent Sayre, Rui Wu, and Liu Yi. An empirical investigation of the relationship between spectra

differences and regression faults. *Software Testing Verification and Reliability*, 10(3):171–194, 2000.

[HS88] Mary Jean Harrold and ML Souffa. An incremental approach to unit testing during maintenance. In *Software Maintenance, 1988., Proceedings of the Conference on*, pages 362–367. IEEE, 1988.

[HWRS08] Mats PE Heimdahl, Michael W Whalen, Ajitha Rajan, and Matt Staats. On mc/dc and implementation structure: An empirical study. In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 5–B. IEEE, 2008.

[JH05] James A Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 273–282. ACM, 2005.

[JH11] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on*, 37(5):649–678, 2011.

[JHF12] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC '12, pages 46–55, New York, NY, USA, 2012. ACM.

[JJI+14] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 654–665. ACM, 2014.

[Joh17] Martin Johansen. Software product line covering array tool, 2017.

[KB11] Robert Könighofer and Roderick Bloem. Automated error localization and correction for imperative programs. In *Formal Methods in Computer-Aided Design (FMCAD), 2011*, pages 91–100. IEEE, 2011.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, 1990.

[KFK14] Aaron Kane, Thomas E. Fuhrman, and Philip Koopman. Monitor based oracles for cyber-physical system testing: Practical experience report. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*, pages 148–155, 2014.

[KIJT17] Muhammad Khatibsyarbini, Mohd Adham Isa, Dayang NA Jawawi, and Rooster Tumeng. Test case prioritization approaches in regression testing: A systematic literature review. *Information and Software Technology*, 2017.

[KKLH09] Rick Kuhn, Raghu Kacker, Yu Lei, and Justin Hunter. Combinatorial software testing. *Computer*, 42:94–96, 2009.

[KKT07] Bogdan Korel, George Koutsogiannakis, and Luay H Tahat. Model-based test prioritization heuristic methods and their evaluation. In *Proceedings of the 3rd international workshop on Advances in model-based testing*, pages 34–43. ACM, 2007.

[KKT08] Bogdan Korel, George Koutsogiannakis, and Luay H Tahat. Application of system models in regression test suite prioritization. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 247–256. IEEE, 2008.

[KM09] R Krishnamoorthi and SA Sahaaya Arul Mary. Factor oriented requirement coverage based system test case prioritization of new and regression test cases. *Information and Software Technology*, 51(4):799–808, 2009.

[KMS15] Joseph Kempka, Phil McMinn, and Dirk Sudholt. Design and analysis of different alternating variable searches for search-based software testing. *Theoretical Computer Science*, 605:1–20, 2015.

[KNK+17] Jacob Krüger, Sebastian Nielebock, Sebastian Krieter, Christian Diedrich, Thomas Leich, Gunter Saake, Sebastian Zug, and Frank Ortmeier. Beyond software product lines: Variability modeling in cyber-physical systems. In *Proceedings of the 21st International*

*Systems and Software Product Line Conference (SPLC 2017) - Vision Track*, 2017.

[KSM+15] Eric Knauss, Miroslaw Staron, Wilhelm Meding, Ola Söder, Agneta Nilsson, and Magnus Castell. Supporting continuous integration by code-churn based test selection. In *Proceedings of the 2nd International Workshop on Rapid Continuous Software Engineering (RCoSE'15)*, pages 19–25. IEEE Press, 2015.

[KTH05] Bogdan Korel, Luay Ho Tahat, and Mark Harman. Test prioritization using system models. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 559–568. IEEE, 2005.

[KW04] Daniel Köb and Franz Wotawa. Introducing alias information into model-based debugging. In *Proceedings of the 16th European Conference on Artificial Intelligence*, pages 833–837. IOS Press, 2004.

[KWW09] Peter M. Kruse, Joachim Wegener, and Stefan Wappler. A highly configurable test system for evolutionary black-box testing of embedded systems. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1545–1552, New York, NY, USA, 2009. ACM.

[LBB15] Z. Lu, M. M. Bezemer, and J. F. Broenink. Model-driven design of simulation support for the terra robot software tool suite. In *Communicating Process Architectures 2015*, pages 143 – 158, 2015.

[Leh00] E. Lehmann. Time partition testing: A method for testing dynamic functional behaviour. In *Proceedings of the European Software Test Congress*, 2000.

[LFN+17] Remo Lachmann, Michael Felderer, Manuel Nieke, Sandro Schulze, Christoph Seidl, and Ina Schaefer. Multi-objective black-box test case selection for system testing. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1311–1318. ACM, 2017.

[LHFRE15] Roberto E Lopez-Herrejon, Stefan Fischer, Rudolf Ramler, and Alexander Egyed. A first systematic mapping study on combinatorial interaction testing for software product lines. In *Software*

*Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, pages 1–10. IEEE, 2015.

[LHH07] Zheng Li, Mark Harman, and Robert M Hierons. Search algorithms for regression test case prioritization. *IEEE Transactions on software Engineering*, 33(4):225–237, 2007.

[LHLE15] Roberto E. Lopez-Herrejon, Lukas Linsbauer, and Alexander Egyed. A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology*, 61:33 – 51, 2015.

[LKL12] Jihyun Lee, Sungwon Kang, and Danhyung Lee. A survey on software product line testing. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 31–40. ACM, 2012.

[LLL+15] Remo Lachmann, Sascha Lity, Sabrina Lischke, Simon Beddig, Sandro Schulze, and Ina Schaefer. Delta-oriented test case prioritization for integration testing of software product lines. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 81–90, New York, NY, USA, 2015. ACM.

[LLNB17] Bing Liu, Lucia Lucia, Shiva Nejati, and Lionel Briand. Improving fault localization for simulink models using search-based testing and prediction models. In *24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2017)*, 2017.

[LLSG12] S. Lity, M. Lochau, I. Schaefer, and U. Goltz. Delta-oriented model-based spl regression testing. In *3rd International Workshop on Product LinE Approaches in Software Engineering, PLEASE 2012*, pages 53 – 6, Piscataway, NJ, USA, 2012.

[LLT15] Tien-Duy B. Le, David Lo, and Ferdian Thung. Should i follow this fault localization tool's output? *Empirical Software Engineering*, 20(5):1237–1274, 2015.

[LMP16] Qi Luo, Kevin Moran, and Denys Poshyvanyk. A large-scale empirical comparison of static and dynamic test case prioritization

techniques. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 559–570. ACM, 2016.

[LS15] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. Lee and Seshia, 2 edition, 2015.

[LTL13] Tien-Duy B Le, Ferdian Thung, and David Lo. Theory and practice, do they match? a case with spectrum-based fault localization. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 380–383. IEEE, 2013.

[LTMHT14] Khuat Thanh Le Thi My Hanh and Nguyen Thanh Binh Tung. Mutation-based test data generation for simulink models using genetic algorithm and simulated annealing. *International Journal of Computer and Information Technology*, 3(04):763–771, 2014.

[LWG+17] Xuelin Li, W. Eric Wong, Ruizhi Gao, Linghuan Hu, and Shigeru Hosono. Genetic algorithm-based test generation for software product line with the integration of fault localization techniques. *Empirical Software Engineering*, pages 1–51, 2017.

[LYAZ16a] Hong Lu, Tao Yue, Shaukat Ali, and Li Zhang. Integrating search and constraint solving for nonconformity resolving recommendations of system product line configuration. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 57–68, 2016.

[LYAZ16b] Hong Lu, Tao Yue, Shaukat Ali, and Li Zhang. Model-based incremental conformance checking to enable interactive product configuration. *Information and Software Technology*, 72:68–89, 04/2016 2016.

[MASE17] Urtzi Markiegi, Aitor Arrieta, Goiuria Sagardui, and Leire Etxeberria. Search-based product line fault detection allocating test cases iteratively. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume A*, SPLC '17, pages 123–132, New York, NY, USA, 2017. ACM.

[Mat15] Reza Matinnejad. Simulink fault patterns. Technical report, 2015.

[Mat16] MathWorks. http://es.mathworks.com/discovery/cyber-physical-systems.html, March 2016.

[MBC09a] M. Mendonca, M. Branco, and D. Cowan. S.P.L.O.T.: Software Product Lines Online Tools. In *Companion to the 24th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 761–762, Orlando, Florida, USA, October 2009. ACM.

[MBC09b] Marcílio Mendonça, Moises Branco, and Donald Cowan. S.p.l.o.t. - software product lines online tools. In *24th ACM SIGPLAN Conference on object oriented programming systems languages and applications - OOPSLA Companion*, page 761, Orlando, Florida, USA, 10/2009 2009. ACM Press, ACM Press.

[MBED12] Wolfgang Mueller, Markus Becker, Ahmed Elfeky, and Anthony DiPasquale. Virtual prototyping of cyber-physical systems. In *Asia and South Pacific Design Automation Conference*, pages 219 – 226, 2012.

[McM04] Phil McMinn. Search-based software test data generation: a survey. *Software testing, Verification and reliability*, 14(2):105–156, 2004.

[MGP08] B. Magro, J. Garbajosa, and J. Perez. A software product line definition for validation environments. In *12th International Software Product Line Conference (SPLC)*, pages 45 – 54, Piscataway, NJ, USA, 2008.

[MGS13] Dusica Marijan, Arnaud Gotlieb, and Sagar Sen. Test case prioritization for continuous regression testing: An industrial case study. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance (ICSM'13)*, pages 540–543. IEEE Computer Society, 2013.

[MGS+17] Morten Mossige, Arnaud Gotlieb, Helge Spieker, Hein Meling, and Mats Carlsson. Time-aware test case execution scheduling for cyber-physical systems. In *International Conference on Principles and Practice of Constraint Programming*, pages 387–404. Springer, Cham, 2017.

[MH97] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[MH15] Anila Mjeda and Mike Hinchey. Requirement-centric reactive testing for safety-related automotive software. In *2015 IEEE/ACM 2nd International Workshop on Requirements Engineering and Testing*, Florence, Italy, 2015.

[Mje13] Anila Mjeda. *Standard-Compliant Testing for Safety-Related Automotive Software*. PhD thesis, University of Limeric, 2013.

[MK16] Phil McMinn and Gregory M Kapfhammer. Avmf: An open-source framework and implementation of the alternating variable method. In *International Symposium on Search Based Software Engineering*, pages 259–266. Springer, 2016.

[MM16] Morteza Mohaqeqi and Mohammad Reza Mousavi. Sound test-suites for cyber-physical systems. In *Theoretical Aspects of Software Engineering (TASE), 2016 10th International Symposium on*, pages 42–48. IEEE, 2016.

[MMT14] Morteza Mohaqeqi, Mohammad Reza Mousavi, and Walid Taha. Conformance testing of cyber-physical systems: A comparative study. *ECEASST*, 70:1–16, 2014.

[MNB+15] Reza Matinnejad, Shiva Nejati, Lionel Briand, Thomas Bruckmann, and Claude Poull. Search-based automated testing of continuous controllers: Framework, tool support, and case studies. *Information and Software Technology*, 57:705 – 722, 2015.

[MNB17] Reza Matinnejad, Shiva Nejati, and Lionel C. Briand. Automated testing of hybrid simulink/stateflow controllers: Industrial case studies. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 938–943, New York, NY, USA, 2017. ACM.

[MNBB16] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. Automated test suite generation for time-continuous simulink models. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 595–606, New York, NY, USA, 2016. ACM.

[MRE02] Alexey G. Malishevsky, Gregg Rothermel, and Sebastian Elbaum. Modeling the cost-benefits tradeoffs for regression testing tech-

niques. In *In Proceedings of the International Conference on Software Maintenance*, pages 204–213, 2002.

[MS76] Webb Miller and David L. Spooner. Automatic generation of floating-point test data. *IEEE Transactions on Software Engineering*, (3):223–226, 1976.

[MS02] Wolfgang Mayer and Markus Stumptner. Modeling programs with unstructured control flow for debugging. In *Australian Joint Conference on Artificial Intelligence*, pages 107–118. Springer, 2002.

[MS06] Ghassan Misherghi and Zhendong Su. Hdd: Hierarchical delta debugging. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 142–151, New York, NY, USA, 2006. ACM.

[MS07] Wolfgang Mayer and Markus Stumptner. Abstract interpretation of programs for model-based debugging. In *IJCAI*, pages 471–476, 2007.

[MSB+14] Pieter J. Mosterman, David Escobar Sanabria, Enes Bilgin, Kun Zhang, and Justyna Zander. Automating humanitarian missions with a heterogeneous fleet of vehicles. *Annual Reviews in Control*, 38(2):259–270, 2014.

[MSR14] Christian Manz, Michael Schulze, and Manfred Reichert. An approach to detect the origin and distribution of software defects in an evolving cyber-physical system. In *Workshop on Emerging Idears and Trends in Engineering of Cyber-Physical Systems (EITEC '14)*, April 2014.

[MSW00] Cristinel Mateis, Markus Stumptner, and Franz Wotawa. Modeling java programs for diagnosis. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 171–175. IOS Press, 2000.

[MSWW02] Wolfgang Mayer, Markus Stumptner, Dominik Wieland, and Franz Wotawa. Can ai help to improve debugging substantially? debugging experiences with value-based models. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 417–421. IOS Press, 2002.

[MZ16] Pieter J. Mosterman and Justyna Zander. Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems. *Software & Systems Modeling*, 15(ISSN 1619-1366):5–16, 2016.

[NAW+08] Syeda Nessa, Muhammad Abedin, W Eric Wong, Latifur Khan, and Yu Qi. Software fault localization using n-gram analysis. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 548–559. Springer, 2008.

[NdCMM+11] Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, John D McGregor, Eduardo Santana De Almeida, and Silvio Romero de Lemos Meira. A systematic mapping study of software product lines testing. *Information and Software Technology*, 53(5):407–423, 2011.

[NYA+13] Kunming Nie, Tao Yue, Shaukat Ali, Li Zhang, and Zhiqiang Fan. Constraints: The core of supporting automated product configuration of cyber-physical systems. In *ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems*, pages 370–387, 2013.

[OZML11] Sebastian Oster, Ivan Zorcic, Florian Markert, and Malte Lochau. Moso-polite - tool support for pairwise and model-based software product line testing. In *VaMoS*, pages 79–82, 2011.

[PCJ+17] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D. Ernst, Deric Pang, and Benjamin Keller. Evaluating and improving fault localization. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 609–620, Piscataway, NJ, USA, 2017. IEEE Press.

[PHH+16] Mike Papadakis, Christopher Henard, Mark Harman, Yue Jia, and Yves Le Traon. Threats to the validity of mutation-based test assessment. In *International Symposium on Software Testing and Analysis (ISSTA'16)*, pages 354–365, 2016.

[Pik15] Piketec. Tpt - model-based testing of embedded control systems, July 2015.

[PKT17] Annibale Panichella, Fitsum Kifetew, and Paolo Tonella. Automated test case generation as a many-objective optimisation problem with

dynamic selection of the targets. *IEEE Transactions on Software Engineering*, 2017.

[PMB+12] A. Polzer, D. Merschen, G. Botterweck, A. Pleuss, J. Thomas, B. Hedenetz, and S. Kowalewski. Managing complexity and variability of a model-based embedded software product line. *Innovations and Systems and Software Engineering*, 8(1):35 – 49, 2012.

[POS+12] Gilles Perrouin, Sebastian Oster, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. Pairwise testing for software product lines: Comparison of two approaches. *Software Quality Journal*, 20(3-4):605–643, 2012.

[PPG09] Beatriz Pérez, Macario Polo, and Ignacio García. Model-driven testing in software product lines. In *Proceedings of the 2009 IEEE International Conference on Software Maintenance (ICSM 2009)*, pages 511 – 514, 2009.

[PPP09] Beatriz Pérez, Macario Polo, and Mario Piattini. Towards an automated testing framework to manage variability using the uml testing profile. In *AST*, pages 10–17, 2009.

[PS14] Pure-Systems. pure::variants. http://www.pure-systems.com, July 2014.

[PSK+10] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. Automated and scalable t-wise test case generation strategies for software product lines. In *2010 Third international conference on software testing, verification and validation*, pages 459–468. IEEE, 2010.

[PSS+16] José A Parejo, Ana B Sánchez, Sergio Segura, Antonio Ruiz-Cortés, Roberto E. Lopez-Herrejon, and Alexander Egyed. Multi-objective test case prioritization in highly configurable systems: A case study. *Journal of Systems and Software*, pages –, 2016.

[PWAY16] Dipesh Pradhan, Shuai Wang, Shaukat Ali, and Tao Yue. Search-based cost-effective test case selection within a time budget: An empirical study. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 1085–1092, New York, NY, USA, 2016. ACM.

[RBDL97] Thomas Reps, Thomas Ball, Manuvir Das, and James Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. In *Proceedings of the 6th European SOFTWARE ENGINEERING Conference Held Jointly with the 5th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC '97/FSE-5, pages 432–449, New York, NY, USA, 1997. Springer-Verlag New York, Inc.

[RCC+12] John Regehr, Yang Chen, Pascal Cuoq, Eric Eide, Chucky Ellison, and Xuejun Yang. Test-case reduction for c compiler bugs. In *ACM SIGPLAN Notices*, volume 47, pages 335–346. ACM, 2012.

[Rob10] William Robinson. A roadmap for comprehensive requirements modeling. *Computer*, 43(5):64–72, 2010.

[RSB+13] Rakesh Rana, Miroslaw Staron, Christian Berger, Jörgen Hansson, Martin Nilsson, and Fredrik Törner. Increasing efficiency of iso 26262 verification and validation by combining fault injection and mutation testing with model based development. In *ICSOFT*, pages 251–257, 2013.

[RUCH99] Gregg Rothermel, Roland H Untch, Chengyun Chu, and Mary Jean Harrold. Test case prioritization: An empirical study. In *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*, pages 179–188. IEEE, 1999.

[RUCH01] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, 27(10):929–948, 2001.

[RWSH08] Ajitha Rajan, Michael Whalen, Matt Staats, and Mats Heimdahl. Requirements coverage as an adequacy measure for conformance testing. *Formal Methods and Software Engineering*, pages 86–104, 2008.

[SAM+17] Goiuria Sagardui, Joseba Agirre, Urtzi Markiegi, Aitor Arrieta, Carlos Fernando Nicolás, and Jose María Martín. Multiplex: A co-simulation architecture for elevators validation. In *Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), 2017 IEEE International Workshop of*, pages 1–6. IEEE, 2017.

[SCQ09] Hema Srikanth, Myra B. Cohen, and Xiao Qu. Reducing field failures in system configurable software: Cost-based prioritization. In *Proceedings of the 20th International Symposium on Software Reliability Engineering (ISSRE'09)*, pages 61–70. IEEE Computer Society, 2009.

[SEA+17] Goiuria Sagardui, Leire Etxeberria, Joseba Agirre, Aitor Arrieta, Carlos Fernando Nicolás, and Jose María Martín. A configurable validation environment for refactored embedded software: an application to the vertical transport domain. In *ISSRE 2017 (Industry Track): IEEE International Symposium on Software Reliability Engineering*, 2017.

[SH09] H. Shokry and M. Hinchey. Model-based verification of embedded software. *Computer*, 42(4):53 – 59, 2009.

[SMP10] Vanessa Stricker, Andreas Metzger, and Klaus Pohl. Avoiding redundant testing in application engineering. In *Software Product Lines: Going Beyond*, pages 226–240. Springer, 2010.

[SPH16] Federica Sarro, Alessio Petrozziello, and Mark Harman. Multi-objective software effort estimation. In *Proceedings of the 38th International Conference on Software Engineering*, pages 619–630. ACM, 2016.

[SR05] Mats Skoglund and Per Runeson. A case study of the class firewall regression test selection technique on a large scale distributed software system. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 10–pp. IEEE, 2005.

[SSPRC15] Ana B Sánchez, Sergio Segura, José A Parejo, and Antonio Ruiz-Cortés. Variability testing in the wild: the drupal case study. *Software & Systems Modeling*, pages 1–22, 2015.

[SSRC14a] Ana B. Sánchez, S. Segura, and Antonio Ruiz-Cortés. A comparison of test case prioritization criteria for software product lines. In *IEEE International Conference on Software Testing, Verification, and Validation*, pages 41–50, 2014.

[SSRC14b] Ana B. Sánchez, Sergio Segura, and Antonio Ruiz-Cortés. The drupal framework: A case study to evaluate variability testing tech-

niques. In *8th International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, 2014.

[SWO05] Hema Srikanth, Laurie Williams, and Jason Osborne. System test case prioritization of new and regression test cases. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 10–pp. IEEE, 2005.

[SYAL16] Safdar Aqeel Safdar, Tao Yue, Shaukat Ali, and Hong Lu. Evaluating variability modeling techniques for supporting cyber-physical system product line engineering. In *International Conference on System Analysis and Modeling*, pages 1–19. Springer International Publishing, 2016.

[SZF15] Maria Spichkova, Anna Zamansky, and Eitan Farchi. Towards a human-centred approach in modelling and testing of cyber-physical systems. In *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*, pages 847–851. IEEE, 2015.

[TB17] Hong-Linh Truong and Luca Berardinelli. Testing uncertainty of cyber-physical systems in iot cloud infrastructures: combining model-driven engineering and elastic execution. In *Proceedings of the 1st ACM SIGSOFT International Workshop on Testing Embedded and Cyber-Physical Systems, Santa Barbara, CA*, pages 10–14, 2017.

[TH02] Steffen Thiel and Andreas Hein. Systematic integration of variability into product line architecture design. In *SPLC*, pages 130–153, 2002.

[TKB+14] Thomas Thuem, Christian Kastner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 79:70 – 85, 2014.

[VD98] András Vargha and Harold D Delaney. The kruskal-wallis test and stochastic homogeneity. *Journal of Educational and Behavioral Statistics*, 23(2):170–192, 1998.

[VD00] András Vargha and Harold D Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong.

*Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[VK04] Vijay Vaishnavi and William Kuechler. Design research in information systems, 2004.

[VLOdbH+14] Dimitri Van Landuyt, Steven Op de beeck, Aram Hovsepyan, Sam Michiels, Wouter Joosen, Sven Meynckens, Gjalt de Jong, Olivier Barais, and Mathieu Acher. Towards managing variability in the safety design of an automotive hall effect sensor. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 304–309, New York, NY, USA, 2014. ACM.

[VLW+13] Tanja E. J. Vos, Felix F. Lindlar, Benjamin Wilmes, Andreas Windisch, Arthur I. Baars, Peter M. Kruse, Hamilton Gross, and Joachim Wegener. Evolutionary functional black-box testing in an industrial setting. *Software Quality Journal*, 21(2):259–288, 2013.

[WAG13] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. In *Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, pages 1493 – 1500, Amsterdam, Netherlands, 2013.

[WAG15] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software*, 103(0):370 – 391, 2015.

[WAGL16] Shuai Wang, Shaukat Ali, Arnaud Gotlieb, and Marius Liaaen. A systematic test case selection methodology for product lines: results and insights from an industrial case study. *Empirical Software Engineering*, pages 1–37, 2016.

[Wan15] Shuai Wang. *Systematic Product Line Testing: Methodologies, Automation, and Industrial Application*. PhD thesis, University of Oslo, 2015.

[WAY+16] Shuai Wang, Shaukat Ali, Tao Yue, Yan Li, and Marius Liaaen. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 631–642, 2016.

[WBA+14] Shuai Wang, David Buchmann, Shaukat Ali, Arnaud Gotlieb, Dipesh Pradhan, and Marius Liaaen. Multi-objective test prioritization in software product line testing: An industrial case study. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 32–41, New York, NY, USA, 2014. ACM.

[WDG+12] W Eric Wong, Vidroha Debroy, Richard Golden, Xiaofeng Xu, and Bhavani Thuraisingham. Effective software fault localization using an rbf neural network. *IEEE Transactions on Reliability*, 61(1):149–169, 2012.

[WDGL14] W Eric Wong, Vidroha Debroy, Ruizhi Gao, and Yihao Li. The dstar method for effective software fault localization. *IEEE Transactions on Reliability*, 63(1):290–308, 2014.

[Wei79] Mark David Weiser. Program slices: formal, psychological, and practical investigations of an automatic program abstraction method. 1979.

[Wei84] Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4), 1984.

[WGAL13] Shuai Wang, Arnaud Gotlieb, Shaukat Ali, and Marius Liaaen. Automated test case selection using feature model: An industrial case study. In *MoDELS*, pages 237–253, 2013.

[WGL+16] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software*, 42(8):707–740, August 2016.

[WH08] Michael Wetter and Philip Haves. A modular building controls virtual test bed for the integration of heterogeneous systems. In *Proceedings of the 3rd SimBuild Conference*, pages 69–76, 2008.

[WLT13] Matthias Woehrle, Kai Lampka, and Lothar Thiele. Conformance testing for cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 11(4):84:1–84:23, January 2013.

[WSKR06] Kristen R. Walcott, Mary Lou Soffa, Gregory M. Kapfhammer, and Robert S. Roos. Time-aware test suite prioritization. In *Proceed-*

*ings of the 2006 International Symposium on Software Testing and Analysis*, ISSTA '06, pages 1–12, New York, NY, USA, 2006. ACM.

[WSM02] Franz Wotawa, Markus Stumptner, and Wolfgang Mayer. Model-based debugging or how to diagnose programs automatically. In *IEA/AIE*, pages 746–757. Springer, 2002.

[WSYL11] Jiafu Wan, Hui Suo, Hehua Yan, and Jianqi Liu. A general test platform for cyber-physical systems: Unmanned vehicle with wireless sensor network navigation. *Procedia Engineering*, 24:123 – 127, 2011. International Conference on Advances in Engineering 2011.

[XCKX13] Xiaoyuan Xie, Tsong Yueh Chen, Fei-Ching Kuo, and Baowen Xu. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Trans. Softw. Eng. Methodol.*, 22(4):31:1–31:40, October 2013.

[XWCX13] Xiaoyuan Xie, W Eric Wong, Tsong Yueh Chen, and Baowen Xu. Metamorphic slice: An application in spectrum-based fault localization. *Information and Software Technology*, 55(5):866–879, 2013.

[YCP04] Cemal Yilmaz, Myra B Cohen, and Adam Porter. Covering arrays for efficient fault characterization in complex configuration spaces. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 45–54. ACM, 2004.

[YCP06] Cemal Yilmaz, Myra B Cohen, and Adam A Porter. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering*, 32(1):20–34, 2006.

[YH07] Shin Yoo and Mark Harman. Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 international symposium on Software testing and analysis*, pages 140–150. ACM, 2007.

[YH12] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

[YRW+15] Dongjiang You, Sanjai Rayadurgam, Michael Whalen, Mats PE Heimdahl, and Gregory Gay. Efficient observability-based test generation by dynamic symbolic execution. In *Software Reliability*

*Engineering (ISSRE), 2015 IEEE 26th International Symposium on*, pages 228–238. IEEE, 2015.

[ZC05] Yuan Zhan and John A Clark. Search-based mutation testing for simulink models. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1061–1068. ACM, 2005.

[ZC08] Yuan Zhan and John A Clark. A search-based framework for automatic testing of matlab/simulink models. *Journal of Systems and Software*, 81(2):262–285, 2008.

[ZH02] Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Trans. Softw. Eng.*, 28(2):183–200, February 2002.

[ZHG+09] Lu Zhang, Shan-Shan Hou, Chao Guo, Tao Xie, and Hong Mei. Time-aware test-case prioritization using integer linear programming. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ISSTA '09, pages 213–224, New York, NY, USA, 2009. ACM.

[ZHY13] Lichen Zhang, Jifeng He, and Wensheng Yu. Test case generation from formal models of cyber physical system. *J. Hybrid Inf. Technol*, 6(3):15, 2013.

[ZJC14] Ke Zhai, Bo Jiang, and W. K. Chan. Prioritizing test cases for regression testing of location-based services: Metrics, techniques, and case study. *IEEE Trans. Serv. Comput.*, 7(1):54–67, January 2014.

[ZL07] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

[ZLT+01] Eckart Zitzler, Marco Laumanns, Lothar Thiele, et al. Spea2: Improving the strength pareto evolutionary algorithm. In *Eurogen*, volume 3242, pages 95–100, 2001.

[ZN07] Justyna Zander-Nowicka. Reactive testing and test control of hybrid embedded software. In *Proceedings of the 5th Workshop on System Testing and Validation*, pages 45–62, 2007.

[ZN08] Justyna Zander-Nowicka. *Model-based Testing of Real-Time Embedded Systems in the Automotive Domain*. PhD thesis, Technical University Berlin, 2008.

[ZNSM11] Justyna Zander-Nowicka, Ina Schieferdecker, and Pieter J. Mosterman. *A Taxonomy of Model-Based Testing for Embedded Systems from Multiple Industry Domains*, chapter 1, pages 3–22. Model-Based Testing for Embedded Systems, 2011.

[ZSA+16] Man Zhang, Bran Selic, Shaukat Ali, Tao Yue, Oscar Okariz, and Roland Norgren. Understanding uncertainty in cyber-physical systems: A conceptual model. In *European Conference on Modelling Foundations and Applications*, pages 247–264. Springer, 2016.

[ZZ14] Sai Zhang and Congle Zhang. Software bug localization with markov logic. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 424–427. ACM, 2014.

# Appendices

Appendix A

# Statistical Tests Results for Test Case Prioritization

Table A.1: Summary for the Mann-Whiteney U-Test Statistical Test results for the FDT metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

| | | | | MiL | | | SiL | | | HiL | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Case Study** | **RQ** | **A** | **B** | **+** | **=** | **-** | **+** | **=** | **-** | **+** | **=** | **-** |
| ACC | RQ1 | RS | GREEDY | 26 | 14 | 10 | 27 | 12 | 11 | 18 | 16 | 16 |
| | | RS | AVM | 0 | 7 | 43 | 0 | 5 | 45 | 1 | 27 | 22 |
| | | RS | WBGA | 0 | 14 | 36 | 0 | 10 | 40 | 2 | 31 | 17 |
| | | RS | RWGA | 0 | 9 | 41 | 0 | 12 | 38 | 1 | 32 | 17 |
| | RQ2 | GREEDY | AVM | 1 | 6 | 43 | 0 | 8 | 42 | 11 | 10 | 29 |
| | | GREEDY | WBGA | 6 | 2 | 42 | 5 | 3 | 42 | 11 | 11 | 28 |
| | | GREEDY | RWGA | 5 | 4 | 41 | 5 | 2 | 43 | 11 | 14 | 25 |
| | | AVM | WBGA | 28 | 22 | 0 | 26 | 24 | 0 | 21 | 29 | 0 |
| | | AVM | RWGA | 23 | 26 | 1 | 26 | 24 | 0 | 18 | 32 | 0 |
| | | WBGA | RWGA | 0 | 49 | 1 | 0 | 43 | 7 | 1 | 48 | 1 |
| UAV | RQ1 | RS | GREEDY | 4 | 3 | 43 | 4 | 4 | 42 | 5 | 6 | 39 |
| | | RS | AVM | 3 | 5 | 42 | 1 | 7 | 42 | 3 | 9 | 38 |
| | | RS | WBGA | 1 | 15 | 34 | 1 | 15 | 34 | 3 | 10 | 37 |
| | | RS | RWGA | 1 | 15 | 34 | 1 | 17 | 32 | 0 | 10 | 40 |
| | RQ2 | GREEDY | AVM | 18 | 16 | 16 | 18 | 19 | 13 | 20 | 19 | 11 |
| | | GREEDY | WBGA | 38 | 5 | 7 | 35 | 8 | 6 | 33 | 10 | 7 |
| | | GREEDY | RWGA | 37 | 9 | 4 | 37 | 8 | 5 | 27 | 15 | 8 |
| | | AVM | WBGA | 33 | 16 | 1 | 33 | 13 | 4 | 19 | 26 | 5 |
| | | AVM | RWGA | 34 | 14 | 2 | 31 | 19 | 0 | 15 | 27 | 8 |
| | | WBGA | RWGA | 0 | 49 | 1 | 3 | 44 | 3 | 1 | 42 | 7 |
| TANK | RQ1 | RS | GREEDY | 3 | 7 | 38 | 3 | 8 | 39 | 14 | 15 | 21 |
| | | RS | AVM | 12 | 31 | 7 | 6 | 39 | 5 | 16 | 27 | 7 |
| | | RS | WBGA | 19 | 26 | 5 | 15 | 32 | 3 | 30 | 19 | 1 |
| | | RS | RWGA | 18 | 27 | 5 | 12 | 30 | 8 | 27 | 21 | 2 |
| | RQ2 | GREEDY | AVM | 37 | 9 | 3 | 37 | 8 | 3 | 31 | 5 | 13 |
| | | GREEDY | WBGA | 47 | 3 | 0 | 48 | 2 | 0 | 40 | 7 | 3 |
| | | GREEDY | RWGA | 43 | 3 | 3 | 42 | 5 | 3 | 34 | 11 | 3 |
| | | AVM | WBGA | 10 | 38 | 2 | 13 | 37 | 0 | 17 | 33 | 0 |
| | | AVM | RWGA | 8 | 40 | 2 | 8 | 37 | 5 | 9 | 41 | 0 |
| | | WBGA | RWGA | 0 | 46 | 4 | 0 | 44 | 6 | 0 | 46 | 4 |
| DC eng | RQ1 | RS | GREEDY | 1 | 26 | 13 | 2 | 22 | 16 | 1 | 21 | 18 |
| | | RS | AVM | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | RS | WBGA | 0 | 2 | 38 | 0 | 3 | 37 | 0 | 4 | 36 |
| | | RS | RWGA | 0 | 6 | 34 | 0 | 2 | 38 | 0 | 2 | 38 |
| | RQ2 | GREEDY | AVM | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | GREEDY | WBGA | 0 | 3 | 37 | 0 | 2 | 38 | 0 | 10 | 30 |
| | | GREEDY | RWGA | 0 | 5 | 35 | 0 | 3 | 37 | 0 | 5 | 35 |
| | | AVM | WBGA | 22 | 12 | 6 | 20 | 14 | 6 | 24 | 13 | 3 |
| | | AVM | RWGA | 21 | 12 | 7 | 20 | 16 | 4 | 22 | 13 | 5 |
| | | WBGA | RWGA | 0 | 40 | 0 | 1 | 39 | 0 | 0 | 37 | 3 |

Table A.2: Results for the Spearman's rank correlation, which measures the correlation of the FDT metric with respect to the test suite size. Notice that a negative $\rho$ means an improve in the performance of the algorithm with a larger test suite.

| | | | Product 1 | | Product 2 | | Product 3 | | Product 4 | | Product 5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value |
| ACC | MiL | RS | 0.102 | 0.02189 | 0.098 | 0.02898 | 0.073 | 0.10468 | -0.009 | 0.84038 | 0.091 | 0.04247 |
| | | GREEDY | 0.413 | <0.0000 | 0.255 | <0.0000 | -0.213 | <0.0000 | -0.213 | <0.0000 | -0.213 | <0.0000 |
| | | AVM | -0.209 | <0.0000 | -0.225 | <0.0000 | -0.084 | 0.05981 | -0.118 | 0.00815 | -0.108 | 0.01568 |
| | | WBGA | -0.026 | 0.55910 | -0.032 | 0.48022 | 0.003 | 0.94846 | 0.032 | 0.47356 | 0.023 | 0.60212 |
| | | RWGA | 0.024 | 0.59179 | 0.032 | 0.47333 | 0.107 | 0.01699 | 0.082 | 0.06584 | -0.003 | 0.95185 |
| | SiL | RS | 0.122 | 0.00629 | 0.092 | 0.03874 | 0.110 | 0.01371 | 0.040 | 0.37555 | 0.027 | 0.55174 |
| | | GREEDY | 0.413 | <0.0000 | 0.255 | <0.0000 | -0.213 | <0.0000 | -0.213 | <0.0000 | -0.213 | <0.0000 |
| | | AVM | -0.156 | 0.00045 | -0.216 | <0.0000 | -0.059 | 0.19010 | -0.035 | 0.43311 | -0.021 | 0.64149 |
| | | WBGA | -0.061 | 0.17464 | -0.017 | 0.70785 | -0.059 | 0.19029 | 0.000 | 0.99940 | 0.041 | 0.35958 |
| | | RWGA | 0.002 | 0.96014 | 0.042 | 0.35371 | 0.076 | 0.08921 | 0.134 | 0.00274 | 0.114 | 0.01084 |
| | HiL | RS | -0.092 | 0.03970 | -0.186 | 0.00003 | 0.027 | 0.55312 | -0.056 | 0.21305 | 0.119 | 0.00793 |
| | | GREEDY | 0.596 | <0.0000 | 0.255 | <0.0000 | -0.323 | <0.0000 | -0.213 | <0.0000 | -0.213 | <0.0000 |
| | | AVM | -0.151 | 0.00073 | -0.055 | 0.22292 | -0.078 | 0.08203 | -0.046 | 0.30965 | 0.009 | 0.84489 |
| | | WBGA | -0.039 | 0.38964 | -0.073 | 0.10231 | 0.017 | 0.69939 | -0.011 | 0.81038 | 0.036 | 0.41748 |
| | | RWGA | 0.080 | 0.07283 | -0.027 | 0.54689 | -0.105 | 0.01931 | 0.058 | 0.19921 | 0.133 | 0.00283 |
| UAV | MiL | RS | -0.288 | <0.0000 | 0.005 | 0.90918 | 0.204 | <0.0000 | -0.168 | 0.00016 | -0.094 | 0.03475 |
| | | GREEDY | * | * | * | * | * | * | * | * | * | * |
| | | AVM | -0.331 | <0.0000 | 0.092 | 0.03925 | -0.204 | <0.0000 | -0.520 | <0.0000 | -0.531 | <0.0000 |
| | | WBGA | -0.147 | 0.00095 | -0.002 | 0.96768 | 0.232 | <0.0000 | 0.142 | 0.00147 | 0.006 | 0.88990 |
| | | RWGA | 0.004 | 0.92804 | -0.173 | 0.00010 | 0.201 | 0.00001 | 0.064 | 0.15447 | -0.027 | 0.54200 |
| | SiL | RS | -0.325 | <0.0000 | -0.090 | 0.04474 | 0.217 | <0.0000 | -0.069 | 0.12323 | -0.127 | 0.00457 |
| | | GREEDY | * | * | * | * | * | * | * | * | * | * |
| | | AVM | -0.335 | <0.0000 | -0.002 | 0.96853 | -0.180 | 0.00005 | -0.533 | <0.0000 | -0.526 | <0.0000 |
| | | WBGA | -0.098 | 0.02881 | 0.025 | 0.58394 | 0.252 | <0.0000 | 0.083 | 0.06466 | 0.061 | 0.17393 |
| | | RWGA | -0.109 | 0.01431 | -0.036 | 0.42789 | 0.275 | <0.0000 | 0.071 | 0.11062 | 0.067 | 0.13747 |
| | HiL | RS | -0.294 | <0.0000 | -0.267 | <0.0000 | 0.222 | <0.0000 | 0.118 | 0.00847 | 0.140 | 0.00173 |
| | | GREEDY | * | * | * | * | * | * | * | * | * | * |
| | | AVM | -0.684 | <0.0000 | -0.182 | 0.00004 | 0.027 | 0.54447 | -0.040 | 0.37196 | -0.137 | 0.00218 |
| | | WBGA | -0.201 | 0.00001 | 0.187 | 0.00003 | 0.278 | <0.0000 | 0.069 | 0.12342 | -0.025 | 0.58357 |
| | | RWGA | -0.221 | <0.0000 | 0.019 | 0.67659 | 0.215 | <0.0000 | 0.136 | 0.00236 | 0.017 | 0.69947 |
| TANK | MiL | RS | -0.4293 | <0.0000 | -0.4107 | <0.0000 | -0.4057 | <0.0000 | -0.3495 | <0.0000 | -0.4421 | <0.0000 |
| | | GREEDY | -0.7979 | <0.0000 | -0.7979 | <0.0000 | -0.7979 | <0.0000 | -0.7531 | <0.0000 | -0.7531 | <0.0000 |
| | | AVM | -0.4350 | <0.0000 | -0.2490 | <0.0000 | -0.1702 | 0.00013 | -0.5323 | <0.0000 | -0.2376 | <0.0000 |
| | | WBGA | -0.1969 | 0.00001 | -0.2479 | <0.0000 | -0.2355 | <0.0000 | -0.2734 | <0.0000 | -0.3617 | <0.0000 |
| | | RWGA | -0.2727 | <0.0000 | -0.2255 | <0.0000 | -0.1762 | 0.00007 | -0.3064 | <0.0000 | -0.3320 | <0.0000 |
| | SiL | RS | -0.3833 | <0.0000 | -0.3804 | <0.0000 | -0.3828 | <0.0000 | -0.2770 | <0.0000 | -0.4584 | <0.0000 |
| | | GREEDY | -0.7979 | <0.0000 | -0.7979 | <0.0000 | -0.7979 | <0.0000 | -0.7531 | <0.0000 | -0.7531 | <0.0000 |
| | | AVM | -0.4116 | <0.0000 | -0.2194 | <0.0000 | -0.1817 | 0.00004 | -0.5445 | <0.0000 | -0.2865 | <0.0000 |
| | | WBGA | -0.1909 | 0.00002 | -0.1781 | 0.00006 | -0.2926 | <0.0000 | -0.3027 | <0.0000 | -0.3137 | <0.0000 |
| | | RWGA | -0.1519 | 0.00065 | -0.1867 | 0.00003 | -0.2403 | <0.0000 | -0.2162 | <0.0000 | -0.3413 | <0.0000 |
| | HiL | RS | -0.4609 | <0.0000 | -0.2646 | <0.0000 | -0.2296 | <0.0000 | -0.4880 | <0.0000 | -0.4255 | <0.0000 |
| | | GREEDY | -0.7979 | <0.0000 | -0.1427 | 0.00138 | -0.1427 | 0.00138 | -0.7680 | <0.0000 | -0.7680 | <0.0000 |
| | | AVM | -0.4631 | <0.0000 | -0.3054 | <0.0000 | -0.1392 | 0.00180 | -0.4626 | <0.0000 | -0.4165 | <0.0000 |
| | | WBGA | -0.3027 | <0.0000 | -0.1901 | 0.00002 | -0.2059 | <0.0000 | -0.2890 | <0.0000 | -0.3795 | <0.0000 |
| | | RWGA | -0.2582 | <0.0000 | -0.2390 | <0.0000 | -0.1606 | 0.00031 | -0.3640 | <0.0000 | -0.3782 | <0.0000 |
| DC eng | MiL | RS | 0.1670 | 0.00018 | 0.1211 | 0.00669 | 0.2253 | <0.0000 | 0.2176 | <0.0000 | | |
| | | GREEDY | 0.8909 | <0.0000 | 0.6727 | <0.0000 | 0.9273 | <0.0000 | 0.8667 | <0.0000 | | |
| | | AVM | 0.0529 | 0.23758 | 0.0741 | 0.09783 | 0.0325 | 0.46908 | 0.0058 | 0.89646 | | |
| | | WBGA | 0.6120 | <0.0000 | 0.5979 | <0.0000 | 0.6396 | <0.0000 | 0.5733 | <0.0000 | | |
| | | RWGA | 0.6264 | <0.0000 | 0.5892 | <0.0000 | 0.6264 | <0.0000 | 0.5974 | <0.0000 | | |
| | SiL | RS | 0.1686 | 0.00015 | 0.2008 | 0.00001 | 0.2829 | <0.0000 | 0.3179 | <0.0000 | | |
| | | GREEDY | 0.8909 | <0.0000 | 0.6727 | <0.0000 | 0.9273 | <0.0000 | 0.8667 | <0.0000 | | |
| | | AVM | 0.0438 | 0.32880 | 0.0741 | 0.09808 | 0.0148 | 0.74143 | 0.0464 | 0.30066 | | |
| | | WBGA | 0.6195 | <0.0000 | 0.6074 | <0.0000 | 0.5725 | <0.0000 | 0.5948 | <0.0000 | | |
| | | RWGA | 0.6280 | <0.0000 | 0.6037 | <0.0000 | 0.6153 | <0.0000 | 0.6187 | <0.0000 | | |
| | HiL | RS | 0.2472 | <0.0000 | 0.2078 | <0.0000 | 0.2325 | <0.0000 | 0.2006 | 0.00001 | | |
| | | GREEDY | 0.8909 | <0.0000 | 0.4061 | <0.0000 | 0.5273 | <0.0000 | 0.7576 | <0.0000 | | |
| | | AVM | -0.0513 | 0.25186 | 0.0467 | 0.29755 | -0.0112 | 0.80207 | 0.0324 | 0.47019 | | |
| | | WBGA | 0.5839 | <0.0000 | 0.5532 | <0.0000 | 0.6462 | <0.0000 | 0.5828 | <0.0000 | | |
| | | RWGA | 0.6325 | <0.0000 | 0.6324 | <0.0000 | 0.6597 | <0.0000 | 0.6564 | <0.0000 | | |

Table A.3: Summary for the Mann-Whiteney U-Test Statistical Test results for the APFD metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

| | | | | MiL | | | SiL | | | HiL | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Case Study | RQ | A | B | + | = | - | + | = | - | + | = | - |
| ACC | RQ1 | RS | GREEDY | 50 | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 |
| | | RS | AVM | 0 | 16 | 34 | 0 | 16 | 34 | 4 | 34 | 12 |
| | | RS | WBGA | 0 | 36 | 14 | 1 | 32 | 17 | 18 | 22 | 10 |
| | | RS | RWGA | 0 | 29 | 21 | 0 | 28 | 22 | 9 | 30 | 11 |
| | RQ2 | GREEDY | AVM | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | GREEDY | WBGA | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 1 | 49 |
| | | GREEDY | RWGA | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | AVM | WBGA | 23 | 27 | 0 | 20 | 30 | 0 | 22 | 28 | 0 |
| | | AVM | RWGA | 25 | 22 | 3 | 21 | 28 | 1 | 16 | 34 | 0 |
| | | WBGA | RWGA | 1 | 46 | 3 | 2 | 45 | 3 | 1 | 49 | 0 |
| UAV | RQ1 | RS | GREEDY | 9 | 8 | 33 | 10 | 9 | 31 | 12 | 8 | 30 |
| | | RS | AVM | 5 | 11 | 34 | 5 | 14 | 31 | 6 | 13 | 31 |
| | | RS | WBGA | 2 | 9 | 39 | 1 | 11 | 38 | 4 | 15 | 31 |
| | | RS | RWGA | 1 | 17 | 32 | 4 | 11 | 35 | 4 | 15 | 31 |
| | RQ2 | GREEDY | AVM | 7 | 30 | 13 | 7 | 27 | 16 | 7 | 25 | 18 |
| | | GREEDY | WBGA | 4 | 26 | 19 | 8 | 23 | 19 | 6 | 26 | 18 |
| | | GREEDY | RWGA | 9 | 25 | 16 | 8 | 25 | 17 | 2 | 31 | 17 |
| | | AVM | WBGA | 3 | 28 | 19 | 5 | 31 | 14 | 5 | 31 | 14 |
| | | AVM | RWGA | 5 | 35 | 10 | 4 | 34 | 12 | 3 | 35 | 12 |
| | | WBGA | RWGA | 2 | 48 | 0 | 1 | 47 | 2 | 0 | 49 | 1 |
| TANK | RQ1 | RS | GREEDY | 40 | 7 | 3 | 43 | 3 | 3 | 36 | 10 | 4 |
| | | RS | AVM | 9 | 24 | 17 | 9 | 22 | 19 | 17 | 29 | 4 |
| | | RS | WBGA | 19 | 24 | 7 | 17 | 29 | 4 | 34 | 16 | 0 |
| | | RS | RWGA | 16 | 23 | 11 | 15 | 24 | 11 | 35 | 15 | 0 |
| | RQ2 | GREEDY | AVM | 3 | 8 | 39 | 3 | 7 | 40 | 5 | 13 | 32 |
| | | GREEDY | WBGA | 5 | 12 | 32 | 4 | 14 | 32 | 10 | 18 | 22 |
| | | GREEDY | RWGA | 4 | 9 | 37 | 5 | 10 | 35 | 8 | 15 | 27 |
| | | AVM | WBGA | 23 | 27 | 0 | 23 | 27 | 0 | 18 | 32 | 0 |
| | | AVM | RWGA | 15 | 35 | 0 | 13 | 36 | 1 | 19 | 30 | 1 |
| | | WBGA | RWGA | 0 | 49 | 1 | 0 | 46 | 4 | 0 | 49 | 1 |
| DC eng | RQ1 | RS | GREEDY | 40 | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 |
| | | RS | AVM | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | RS | WBGA | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | RS | RWGA | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | RQ2 | GREEDY | AVM | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | GREEDY | WBGA | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | GREEDY | RWGA | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | AVM | WBGA | 15 | 15 | 10 | 17 | 14 | 9 | 19 | 17 | 4 |
| | | AVM | RWGA | 14 | 17 | 9 | 15 | 15 | 10 | 18 | 15 | 7 |
| | | WBGA | RWGA | 0 | 40 | 0 | 0 | 37 | 3 | 0 | 37 | 3 |

Table A.4: Results for the Spearman's rank correlation test, which measures the correlation of the APFD metric with respect to the test suite size. Notice that a positive $\rho$ means an improve in the performance of the algorithm with a larger test suite.

| | | | Product 1 | | Product 2 | | Product 3 | | Product 4 | | Product 5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value |
| ACC | MiL | RS | 0.892 | <0.0000 | 0.888 | <0.0000 | 0.796 | <0.0000 | 0.841 | <0.0000 | 0.823 | <0.0000 |
| | | GREEDY | 0.976 | <0.0000 | 0.891 | <0.0000 | 0.939 | <0.0000 | 0.939 | <0.0000 | 0.939 | <0.0000 |
| | | AVM | 0.956 | <0.0000 | 0.964 | <0.0000 | 0.889 | <0.0000 | 0.863 | <0.0000 | 0.862 | <0.0000 |
| | | WBGA | 0.923 | <0.0000 | 0.941 | <0.0000 | 0.876 | <0.0000 | 0.854 | <0.0000 | 0.862 | <0.0000 |
| | | RWGA | 0.922 | <0.0000 | 0.936 | <0.0000 | 0.855 | <0.0000 | 0.834 | <0.0000 | 0.853 | <0.0000 |
| | SiL | RS | 0.883 | <0.0000 | 0.903 | <0.0000 | 0.742 | <0.0000 | 0.865 | <0.0000 | 0.836 | <0.0000 |
| | | GREEDY | 0.976 | <0.0000 | 0.891 | <0.0000 | 0.939 | <0.0000 | 0.939 | <0.0000 | 0.939 | <0.0000 |
| | | AVM | 0.947 | <0.0000 | 0.961 | <0.0000 | 0.878 | <0.0000 | 0.870 | <0.0000 | 0.857 | <0.0000 |
| | | WBGA | 0.927 | <0.0000 | 0.934 | <0.0000 | 0.888 | <0.0000 | 0.866 | <0.0000 | 0.841 | <0.0000 |
| | | RWGA | 0.916 | <0.0000 | 0.926 | <0.0000 | 0.872 | <0.0000 | 0.843 | <0.0000 | 0.848 | <0.0000 |
| | HiL | RS | 0.748 | <0.0000 | 0.775 | <0.0000 | 0.782 | <0.0000 | 0.824 | <0.0000 | 0.825 | <0.0000 |
| | | GREEDY | 0.818 | <0.0000 | 0.758 | <0.0000 | 0.915 | <0.0000 | 0.891 | <0.0000 | 0.818 | <0.0000 |
| | | AVM | 0.895 | <0.0000 | 0.789 | <0.0000 | 0.795 | <0.0000 | 0.813 | <0.0000 | 0.812 | <0.0000 |
| | | WBGA | 0.856 | <0.0000 | 0.762 | <0.0000 | 0.739 | <0.0000 | 0.767 | <0.0000 | 0.747 | <0.0000 |
| | | RWGA | 0.844 | <0.0000 | 0.772 | <0.0000 | 0.754 | <0.0000 | 0.768 | <0.0000 | 0.768 | <0.0000 |
| UAV | MiL | RS | 0.892 | <0.0000 | 0.888 | <0.0000 | 0.796 | <0.0000 | 0.841 | <0.0000 | 0.823 | <0.0000 |
| | | GREEDY | 0.976 | <0.0000 | 0.891 | <0.0000 | 0.939 | <0.0000 | 0.939 | <0.0000 | 0.939 | <0.0000 |
| | | AVM | 0.956 | <0.0000 | 0.964 | <0.0000 | 0.889 | <0.0000 | 0.863 | <0.0000 | 0.862 | <0.0000 |
| | | WBGA | 0.923 | <0.0000 | 0.941 | <0.0000 | 0.876 | <0.0000 | 0.854 | <0.0000 | 0.862 | <0.0000 |
| | | RWGA | 0.922 | <0.0000 | 0.936 | <0.0000 | 0.855 | <0.0000 | 0.834 | <0.0000 | 0.853 | <0.0000 |
| | SiL | RS | 0.883 | <0.0000 | 0.903 | <0.0000 | 0.742 | <0.0000 | 0.865 | <0.0000 | 0.836 | <0.0000 |
| | | GREEDY | 0.976 | <0.0000 | 0.891 | <0.0000 | 0.939 | <0.0000 | 0.939 | <0.0000 | 0.939 | <0.0000 |
| | | AVM | 0.947 | <0.0000 | 0.961 | <0.0000 | 0.878 | <0.0000 | 0.870 | <0.0000 | 0.857 | <0.0000 |
| | | WBGA | 0.927 | <0.0000 | 0.934 | <0.0000 | 0.888 | <0.0000 | 0.866 | <0.0000 | 0.841 | <0.0000 |
| | | RWGA | 0.916 | <0.0000 | 0.926 | <0.0000 | 0.872 | <0.0000 | 0.843 | <0.0000 | 0.848 | <0.0000 |
| | HiL | RS | 0.748 | <0.0000 | 0.775 | <0.0000 | 0.782 | <0.0000 | 0.824 | <0.0000 | 0.825 | <0.0000 |
| | | GREEDY | 0.818 | <0.0000 | 0.758 | <0.0000 | 0.915 | <0.0000 | 0.891 | <0.0000 | 0.818 | <0.0000 |
| | | AVM | 0.895 | <0.0000 | 0.789 | <0.0000 | 0.795 | <0.0000 | 0.813 | <0.0000 | 0.812 | <0.0000 |
| | | WBGA | 0.856 | <0.0000 | 0.762 | <0.0000 | 0.739 | <0.0000 | 0.767 | <0.0000 | 0.747 | <0.0000 |
| | | RWGA | 0.844 | <0.0000 | 0.772 | <0.0000 | 0.754 | <0.0000 | 0.768 | <0.0000 | 0.768 | <0.0000 |
| TANK | MiL | RS | 0.936 | <0.0000 | 0.903 | <0.0000 | 0.900 | <0.0000 | 0.916 | <0.0000 | 0.888 | <0.0000 |
| | | GREEDY | 0.964 | <0.0000 | 0.964 | <0.0000 | 0.964 | <0.0000 | 1.000 | <0.0000 | 1.000 | <0.0000 |
| | | AVM | 0.941 | <0.0000 | 0.921 | <0.0000 | 0.935 | <0.0000 | 0.914 | <0.0000 | 0.850 | <0.0000 |
| | | WBGA | 0.896 | <0.0000 | 0.896 | <0.0000 | 0.879 | <0.0000 | 0.877 | <0.0000 | 0.850 | <0.0000 |
| | | RWGA | 0.918 | <0.0000 | 0.884 | <0.0000 | 0.892 | <0.0000 | 0.886 | <0.0000 | 0.858 | <0.0000 |
| | SiL | RS | 0.923 | <0.0000 | 0.895 | <0.0000 | 0.895 | <0.0000 | 0.904 | <0.0000 | 0.890 | <0.0000 |
| | | GREEDY | 0.964 | <0.0000 | 0.964 | <0.0000 | 0.964 | <0.0000 | 1.000 | <0.0000 | 1.000 | <0.0000 |
| | | AVM | 0.938 | <0.0000 | 0.945 | <0.0000 | 0.947 | <0.0000 | 0.915 | <0.0000 | 0.858 | <0.0000 |
| | | WBGA | 0.904 | <0.0000 | 0.886 | <0.0000 | 0.885 | <0.0000 | 0.880 | <0.0000 | 0.845 | <0.0000 |
| | | RWGA | 0.914 | <0.0000 | 0.891 | <0.0000 | 0.895 | <0.0000 | 0.874 | <0.0000 | 0.858 | <0.0000 |
| | HiL | RS | 0.959 | <0.0000 | 0.925 | <0.0000 | 0.923 | <0.0000 | 0.914 | <0.0000 | 0.838 | <0.0000 |
| | | GREEDY | 0.964 | <0.0000 | 0.964 | <0.0000 | 0.964 | <0.0000 | 1.000 | <0.0000 | 1.000 | <0.0000 |
| | | AVM | 0.860 | <0.0000 | 0.870 | <0.0000 | 0.945 | <0.0000 | 0.846 | <0.0000 | 0.850 | <0.0000 |
| | | WBGA | 0.881 | <0.0000 | 0.824 | <0.0000 | 0.839 | <0.0000 | 0.817 | <0.0000 | 0.800 | <0.0000 |
| | | RWGA | 0.886 | <0.0000 | 0.863 | <0.0000 | 0.836 | <0.0000 | 0.839 | <0.0000 | 0.827 | <0.0000 |
| DC eng | MiL | RS | 0.502 | <0.0000 | 0.492 | <0.0000 | 0.387 | <0.0000 | 0.362 | <0.0000 | | |
| | | GREEDY | 0.770 | <0.0000 | 0.891 | <0.0000 | 0.939 | <0.0000 | 0.624 | <0.0000 | | |
| | | AVM | 0.806 | <0.0000 | 0.779 | <0.0000 | 0.828 | <0.0000 | 0.822 | <0.0000 | | |
| | | WBGA | 0.646 | <0.0000 | 0.557 | <0.0000 | 0.615 | <0.0000 | 0.588 | <0.0000 | | |
| | | RWGA | 0.620 | <0.0000 | 0.622 | <0.0000 | 0.653 | <0.0000 | 0.591 | <0.0000 | | |
| | SiL | RS | 0.485 | <0.0000 | 0.441 | <0.0000 | 0.391 | <0.0000 | 0.329 | <0.0000 | | |
| | | GREEDY | 0.770 | <0.0000 | 0.891 | <0.0000 | 0.939 | <0.0000 | 0.624 | <0.0000 | | |
| | | AVM | 0.800 | <0.0000 | 0.796 | <0.0000 | 0.851 | <0.0000 | 0.793 | <0.0000 | | |
| | | WBGA | 0.560 | <0.0000 | 0.578 | <0.0000 | 0.579 | <0.0000 | 0.603 | <0.0000 | | |
| | | RWGA | 0.666 | <0.0000 | 0.650 | <0.0000 | 0.595 | <0.0000 | 0.611 | <0.0000 | | |
| | HiL | RS | 0.295 | <0.0000 | 0.283 | <0.0000 | 0.246 | <0.0000 | 0.247 | <0.0000 | | |
| | | GREEDY | 0.770 | <0.0000 | 0.891 | <0.0000 | 0.939 | <0.0000 | 0.612 | <0.0000 | | |
| | | AVM | 0.865 | <0.0000 | 0.829 | <0.0000 | 0.859 | <0.0000 | 0.797 | <0.0000 | | |
| | | WBGA | 0.558 | <0.0000 | 0.592 | <0.0000 | 0.4933 | <0.0000 | 0.602 | <0.0000 | | |
| | | RWGA | 0.640 | <0.0000 | 0.606 | <0.0000 | 0.596 | <0.0000 | 0.524 | <0.0000 | | |

Table A.5: Summary for the Mann-Whiteney U-Test Statistical Test results for the Simulation Time. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

| Case Study | RQ | A | B | MiL | | | SiL | | | HiL | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | + | = | - | + | = | - | + | = | - |
| ACC | RQ1 | RS | GREEDY | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | RS | AVM | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | RS | WBGA | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | RS | RWGA | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | RQ2 | GREEDY | AVM | 0 | 1 | 49 | 0 | 1 | 49 | 1 | 1 | 48 |
| | | GREEDY | WBGA | 45 | 1 | 4 | 45 | 0 | 5 | 44 | 2 | 4 |
| | | GREEDY | RWGA | 46 | 1 | 3 | 46 | 1 | 3 | 46 | 0 | 4 |
| | | AVM | WBGA | 49 | 1 | 0 | 49 | 1 | 0 | 50 | 0 | 0 |
| | | AVM | RWGA | 50 | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 |
| | | WBGA | RWGA | 47 | 3 | 0 | 47 | 3 | 0 | 42 | 8 | 0 |
| UAV | RQ1 | RS | GREEDY | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | RS | AVM | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | RS | WBGA | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | RS | RWGA | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | RQ2 | GREEDY | AVM | 50 | 0 | 0 | 50 | 0 | 0 | 50 | 1 | 0 |
| | | GREEDY | WBGA | 50 | 0 | 0 | 50 | 0 | 0 | 50 | 2 | 0 |
| | | GREEDY | RWGA | 50 | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 |
| | | AVM | WBGA | 45 | 5 | 0 | 46 | 4 | 0 | 47 | 3 | 0 |
| | | AVM | RWGA | 48 | 2 | 0 | 48 | 2 | 0 | 47 | 3 | 0 |
| | | WBGA | RWGA | 4 | 14 | 32 | 3 | 17 | 30 | 0 | 15 | 35 |
| TANK | RQ1 | RS | GREEDY | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | RS | AVM | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | RS | WBGA | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | | RS | RWGA | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 | 50 |
| | RQ2 | GREEDY | AVM | 10 | 16 | 24 | 11 | 16 | 23 | 18 | 12 | 20 |
| | | GREEDY | WBGA | 50 | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 |
| | | GREEDY | RWGA | 50 | 0 | 0 | 50 | 0 | 0 | 50 | 0 | 0 |
| | | AVM | WBGA | 50 | 0 | 0 | 50 | 0 | 0 | 49 | 1 | 0 |
| | | AVM | RWGA | 50 | 0 | 0 | 50 | 0 | 0 | 48 | 2 | 0 |
| | | WBGA | RWGA | 4 | 45 | 1 | 6 | 42 | 2 | 0 | 48 | 2 |
| DC eng | RQ1 | RS | GREEDY | 18 | 5 | 17 | 19 | 4 | 17 | 16 | 0 | 24 |
| | | RS | AVM | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | RS | WBGA | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | RS | RWGA | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | RQ2 | GREEDY | AVM | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | GREEDY | WBGA | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | GREEDY | RWGA | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | AVM | WBGA | 31 | 9 | 0 | 30 | 10 | 0 | 32 | 8 | 0 |
| | | AVM | RWGA | 33 | 7 | 0 | 34 | 6 | 0 | 31 | 9 | 0 |
| | | WBGA | RWGA | 7 | 33 | 0 | 6 | 34 | 0 | 2 | 31 | 7 |

Table A.6: Summary for the Mann-Whiteney U-Test Statistical Test results for the FRCT metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

| Case Study | RQ | A | B | MiL | | | SiL | | | HiL | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | + | = | - | + | = | - | + | = | - |
| ACC | RQ1 | RS | GREEDY | 26 | 15 | 9 | 23 | 18 | 9 | 23 | 14 | 13 |
| | | RS | AVM | 19 | 30 | 1 | 21 | 29 | 0 | 37 | 5 | 0 |
| | | RS | WBGA | 14 | 36 | 0 | 8 | 42 | 0 | 31 | 7 | 1 |
| | | RS | RWGA | 15 | 35 | 0 | 12 | 38 | 0 | 29 | 12 | 0 |
| | RQ2 | GREEDY | AVM | 16 | 11 | 23 | 19 | 8 | 23 | 28 | 16 | 7 |
| | | GREEDY | WBGA | 13 | 19 | 18 | 13 | 19 | 18 | 32 | 15 | 8 |
| | | GREEDY | RWGA | 13 | 21 | 16 | 11 | 22 | 17 | 29 | 18 | 8 |
| | | AVM | WBGA | 2 | 37 | 11 | 4 | 35 | 11 | 1 | 29 | 7 |
| | | AVM | RWGA | 4 | 36 | 10 | 6 | 34 | 10 | 0 | 36 | 6 |
| | | WBGA | RWGA | 2 | 46 | 2 | 1 | 46 | 3 | 0 | 35 | 2 |
| UAV | RQ1 | RS | GREEDY | 21 | 7 | 21 | 22 | 4 | 23 | 29 | 5 | 16 |
| | | RS | AVM | 24 | 12 | 14 | 25 | 12 | 13 | 19 | 24 | 7 |
| | | RS | WBGA | 32 | 10 | 8 | 30 | 11 | 9 | 29 | 16 | 5 |
| | | RS | RWGA | 30 | 15 | 5 | 30 | 13 | 7 | 28 | 14 | 8 |
| | RQ2 | GREEDY | AVM | 18 | 4 | 28 | 18 | 4 | 28 | 12 | 8 | 30 |
| | | GREEDY | WBGA | 21 | 6 | 23 | 21 | 6 | 23 | 20 | 7 | 23 |
| | | GREEDY | RWGA | 21 | 9 | 20 | 21 | 7 | 22 | 19 | 6 | 25 |
| | | AVM | WBGA | 14 | 31 | 5 | 14 | 28 | 8 | 17 | 27 | 6 |
| | | AVM | RWGA | 17 | 29 | 4 | 15 | 29 | 6 | 16 | 30 | 4 |
| | | WBGA | RWGA | 1 | 49 | 0 | 0 | 50 | 0 | 5 | 42 | 3 |
| TANK | RQ1 | RS | GREEDY | 5 | 4 | 41 | 7 | 2 | 41 | 0 | 7 | 43 |
| | | RS | AVM | 15 | 26 | 9 | 15 | 27 | 8 | 9 | 27 | 14 |
| | | RS | WBGA | 2 | 29 | 19 | 2 | 31 | 17 | 17 | 22 | 11 |
| | | RS | RWGA | 2 | 26 | 22 | 2 | 35 | 13 | 14 | 28 | 8 |
| | RQ2 | GREEDY | AVM | 41 | 6 | 3 | 39 | 7 | 4 | 48 | 1 | 1 |
| | | GREEDY | WBGA | 39 | 4 | 7 | 42 | 3 | 5 | 41 | 7 | 1 |
| | | GREEDY | RWGA | 39 | 6 | 5 | 40 | 5 | 5 | 44 | 5 | 1 |
| | | AVM | WBGA | 1 | 31 | 18 | 3 | 24 | 23 | 17 | 23 | 10 |
| | | AVM | RWGA | 1 | 31 | 18 | 3 | 28 | 19 | 16 | 27 | 7 |
| | | WBGA | RWGA | 0 | 49 | 1 | 1 | 47 | 2 | 0 | 50 | 0 |
| DC eng | RQ1 | RS | GREEDY | 0 | 1 | 39 | 0 | 1 | 39 | 0 | 0 | 40 |
| | | RS | AVM | 40 | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 |
| | | RS | WBGA | 34 | 6 | 0 | 37 | 1 | 2 | 37 | 2 | 1 |
| | | RS | RWGA | 35 | 5 | 0 | 36 | 3 | 1 | 36 | 3 | 1 |
| | RQ2 | GREEDY | AVM | 40 | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 |
| | | GREEDY | WBGA | 39 | 0 | 1 | 39 | 0 | 1 | 40 | 0 | 0 |
| | | GREEDY | RWGA | 39 | 0 | 1 | 39 | 0 | 1 | 40 | 0 | 0 |
| | | AVM | WBGA | 0 | 0 | 40 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | AVM | RWGA | 0 | 1 | 39 | 0 | 0 | 40 | 0 | 0 | 40 |
| | | WBGA | RWGA | 1 | 37 | 2 | 0 | 39 | 1 | 4 | 34 | 2 |

Table A.7: Results for the Spearman's rank correlation test, which measures the correlation of the FRCT metric with respect to the test suite size. Notice that a negative $\rho$ means an improve in the performance of the algorithm with a larger test suite.

| | | | Product 1 | | Product 2 | | Product 3 | | Product 4 | | Product 5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value |
| ACC | MiL | RS | 0.052 | 0.24214 | -0.020 | 0.65343 | -0.285 | <0.0000 | -0.163 | 0.00026 | -0.167 | 0.00017 |
| | | GREEDY | 0.587 | <0.0000 | -0.222 | <0.0000 | 0.128 | 0.00413 | -0.116 | 0.00952 | -0.116 | 0.00952 |
| | | AVM | 0.043 | 0.34097 | 0.191 | 0.00002 | -0.248 | <0.0000 | -0.363 | <0.0000 | -0.387 | <0.0000 |
| | | WBGA | -0.065 | 0.14734 | -0.102 | 0.02296 | -0.188 | 0.00002 | -0.181 | 0.00004 | -0.121 | 0.00675 |
| | | RWGA | -0.110 | 0.01364 | -0.116 | 0.00953 | -0.186 | 0.00003 | -0.166 | 0.00020 | -0.090 | 0.04352 |
| | SiL | RS | -0.057 | 0.20100 | -0.006 | 0.88524 | -0.221 | <0.0000 | -0.181 | 0.00005 | -0.177 | 0.00007 |
| | | GREEDY | 0.587 | <0.0000 | -0.222 | <0.0000 | 0.128 | 0.00413 | -0.116 | 0.00952 | -0.116 | 0.00952 |
| | | AVM | 0.029 | 0.52398 | 0.171 | 0.00013 | -0.323 | <0.0000 | -0.302 | <0.0000 | -0.284 | <0.0000 |
| | | WBGA | -0.098 | 0.02852 | -0.094 | 0.03530 | -0.181 | 0.00005 | -0.146 | 0.00102 | -0.133 | 0.00284 |
| | | RWGA | -0.066 | 0.14320 | -0.108 | 0.01541 | -0.284 | <0.0000 | -0.133 | 0.00279 | -0.113 | 0.01166 |
| | HiL | RS | 0.088 | 0.04974 | 0.161 | 0.00031 | -0.291 | <0.0000 | -0.132 | 0.00303 | -0.113 | 0.01132 |
| | | GREEDY | -0.245 | <0.0000 | -0.222 | <0.0000 | 0.128 | 0.00413 | -0.116 | 0.00952 | -0.116 | 0.00952 |
| | | AVM | 0.129 | 0.00385 | 0.029 | 0.51926 | -0.118 | 0.00831 | -0.139 | 0.00189 | -0.077 | 0.08540 |
| | | WBGA | 0.146 | 0.00105 | 0.047 | 0.29833 | -0.055 | 0.22332 | -0.031 | 0.48526 | 0.039 | 0.38198 |
| | | RWGA | 0.125 | 0.00521 | 0.045 | 0.31242 | -0.108 | 0.01601 | -0.009 | 0.83594 | 0.039 | 0.37895 |
| UAV | MiL | RS | 0.556 | <0.0000 | -0.060 | 0.18055 | 0.047 | 0.29204 | -0.488 | <0.0000 | -0.430 | <0.0000 |
| | | GREEDY | 0.813 | <0.0000 | 0.813 | <0.0000 | -0.287 | <0.0000 | -0.287 | <0.0000 | -0.287 | <0.0000 |
| | | AVM | 0.194 | 0.00001 | -0.086 | 0.05344 | -0.146 | 0.00103 | -0.115 | 0.01021 | -0.098 | 0.02851 |
| | | WBGA | 0.246 | <0.0000 | 0.099 | 0.02735 | 0.328 | <0.0000 | -0.001 | 0.98137 | -0.015 | 0.74082 |
| | | RWGA | 0.212 | <0.0000 | 0.017 | 0.70894 | 0.273 | <0.0000 | -0.040 | 0.36830 | -0.012 | 0.78122 |
| | SiL | RS | 0.588 | <0.0000 | -0.121 | 0.00695 | 0.076 | 0.08866 | -0.467 | <0.0000 | -0.446 | <0.0000 |
| | | GREEDY | 0.813 | <0.0000 | 0.813 | <0.0000 | -0.287 | <0.0000 | -0.287 | <0.0000 | -0.287 | <0.0000 |
| | | AVM | 0.202 | 0.00001 | -0.138 | 0.00194 | -0.123 | 0.00569 | -0.163 | 0.00025 | -0.134 | 0.00262 |
| | | WBGA | 0.241 | <0.0000 | 0.052 | 0.24453 | 0.313 | <0.0000 | 0.022 | 0.62604 | 0.021 | 0.64036 |
| | | RWGA | 0.257 | <0.0000 | 0.046 | 0.30504 | 0.347 | <0.0000 | 0.006 | 0.89444 | 0.027 | 0.54427 |
| | HiL | RS | 0.462 | <0.0000 | -0.454 | <0.0000 | 0.103 | 0.02148 | 0.005 | 0.90770 | -0.258 | <0.0000 |
| | | GREEDY | 0.798 | <0.0000 | 0.798 | <0.0000 | -0.287 | <0.0000 | -0.287 | <0.0000 | -0.287 | <0.0000 |
| | | AVM | 0.257 | <0.0000 | -0.423 | <0.0000 | -0.216 | <0.0000 | -0.057 | 0.20256 | -0.054 | 0.22988 |
| | | WBGA | 0.269 | <0.0000 | 0.103 | 0.02100 | 0.256 | <0.0000 | 0.263 | <0.0000 | 0.141 | 0.00155 |
| | | RWGA | 0.190 | 0.00002 | -0.109 | 0.01504 | 0.300 | <0.0000 | 0.265 | <0.0000 | 0.147 | 0.00101 |
| TANK | MiL | RS | 0.302 | <0.0000 | 0.257 | <0.0000 | 0.195 | 0.00001 | 0.179 | 0.00005 | 0.163 | 0.00025 |
| | | GREEDY | -0.809 | <0.0000 | -0.809 | <0.0000 | -0.809 | <0.0000 | -0.798 | <0.0000 | -0.798 | <0.0000 |
| | | AVM | 0.039 | 0.38525 | -0.134 | 0.00273 | -0.193 | 0.00001 | 0.276 | <0.0000 | -0.081 | 0.07003 |
| | | WBGA | 0.129 | 0.00396 | 0.211 | <0.0000 | 0.110 | 0.01418 | 0.073 | 0.10513 | -0.003 | 0.95221 |
| | | RWGA | 0.172 | 0.00011 | 0.205 | <0.0000 | 0.174 | 0.00009 | 0.086 | 0.05578 | 0.026 | 0.55942 |
| | SiL | RS | 0.297 | <0.0000 | 0.207 | <0.0000 | 0.263 | <0.0000 | 0.147 | 0.00101 | 0.121 | 0.00680 |
| | | GREEDY | -0.809 | <0.0000 | -0.809 | <0.0000 | -0.809 | <0.0000 | -0.798 | <0.0000 | -0.798 | <0.0000 |
| | | AVM | -0.001 | 0.97580 | -0.141 | 0.00163 | -0.185 | 0.00003 | 0.322 | <0.0000 | -0.161 | 0.00029 |
| | | WBGA | 0.117 | 0.00880 | 0.227 | <0.0000 | 0.233 | <0.0000 | 0.022 | 0.62111 | 0.031 | 0.48279 |
| | | RWGA | 0.117 | 0.00868 | 0.192 | 0.00002 | 0.151 | 0.00070 | 0.043 | 0.33262 | 0.051 | 0.25402 |
| | HiL | RS | -0.107 | 0.01708 | -0.179 | 0.00006 | -0.202 | 0.00001 | -0.004 | 0.92599 | 0.048 | 0.28582 |
| | | GREEDY | -0.809 | <0.0000 | -0.809 | <0.0000 | -0.809 | <0.0000 | -0.798 | <0.0000 | -0.798 | <0.0000 |
| | | AVM | -0.361 | <0.0000 | -0.285 | <0.0000 | -0.415 | <0.0000 | 0.050 | 0.25974 | -0.027 | 0.54164 |
| | | WBGA | 0.204 | <0.0000 | 0.315 | <0.0000 | 0.226 | <0.0000 | 0.171 | 0.00013 | 0.097 | 0.02989 |
| | | RWGA | 0.152 | 0.00064 | 0.208 | <0.0000 | 0.218 | <0.0000 | 0.132 | 0.00321 | 0.089 | 0.04714 |
| DC eng | MiL | RS | 0.087 | 0.05196 | 0.205 | <0.0000 | 0.188 | 0.00002 | 0.184 | 0.00004 | | |
| | | GREEDY | 0.158 | 0.00039 | -0.515 | <0.0000 | -0.693 | <0.0000 | -0.467 | <0.0000 | | |
| | | AVM | 0.345 | <0.0000 | 0.571 | <0.0000 | 0.667 | <0.0000 | 0.781 | <0.0000 | | |
| | | WBGA | 0.287 | <0.0000 | 0.318 | <0.0000 | 0.392 | <0.0000 | 0.510 | <0.0000 | | |
| | | RWGA | 0.214 | <0.0000 | 0.321 | <0.0000 | 0.447 | <0.0000 | 0.486 | <0.0000 | | |
| | SiL | RS | 0.102 | 0.02304 | 0.191 | 0.00002 | 0.131 | 0.00334 | 0.166 | 0.00020 | | |
| | | GREEDY | 0.158 | 0.00039 | -0.515 | <0.0000 | -0.693 | <0.0000 | -0.467 | <0.0000 | | |
| | | AVM | 0.372 | <0.0000 | 0.572 | <0.0000 | 0.698 | <0.0000 | 0.799 | <0.0000 | | |
| | | WBGA | 0.264 | <0.0000 | 0.325 | <0.0000 | 0.388 | <0.0000 | 0.510 | <0.0000 | | |
| | | RWGA | 0.245 | <0.0000 | 0.308 | <0.0000 | 0.409 | <0.0000 | 0.461 | <0.0000 | | |
| | HiL | RS | 0.206 | <0.0000 | 0.376 | <0.0000 | 0.265 | <0.0000 | 0.343 | <0.0000 | | |
| | | GREEDY | 0.158 | 0.00039 | -0.552 | <0.0000 | -0.644 | <0.0000 | -0.382 | <0.0000 | | |
| | | AVM | 0.629 | <0.0000 | 0.720 | <0.0000 | 0.764 | <0.0000 | 0.841 | <0.0000 | | |
| | | WBGA | 0.379 | <0.0000 | 0.404 | <0.0000 | 0.473 | <0.0000 | 0.596 | <0.0000 | | |
| | | RWGA | 0.369 | <0.0000 | 0.451 | <0.0000 | 0.482 | <0.0000 | 0.582 | <0.0000 | | |

Table A.8: Summary for the Mann-Whiteney U-Test Statistical Test results for the NFRCT metric. The columns contain the number of artificial problems where algorithm A is significantly superior (+), equal (=), or inferior (-) to algorithm B.

| | | | | HiL | | |
|---|---|---|---|---|---|---|
| Case Study | RQ | A | B | + | = | - |
| ACC | RQ1 | RS | GREEDY | 0 | 2 | 48 |
| | | RS | AVM | 1 | 6 | 43 |
| | | RS | WBGA | 2 | 3 | 45 |
| | | RS | RWGA | 2 | 4 | 44 |
| | RQ2 | GREEDY | AVM | 24 | 10 | 16 |
| | | GREEDY | WBGA | 27 | 7 | 16 |
| | | GREEDY | RWGA | 26 | 9 | 15 |
| | | AVM | WBGA | 5 | 40 | 5 |
| | | AVM | RWGA | 10 | 37 | 3 |
| | | WBGA | RWGA | 2 | 38 | 0 |
| UAV | RQ1 | RS | GREEDY | 27 | 1 | 22 |
| | | RS | AVM | 13 | 24 | 13 |
| | | RS | WBGA | 15 | 27 | 8 |
| | | RS | RWGA | 15 | 24 | 11 |
| | RQ2 | GREEDY | AVM | 12 | 8 | 30 |
| | | GREEDY | WBGA | 20 | 8 | 22 |
| | | GREEDY | RWGA | 20 | 5 | 25 |
| | | AVM | WBGA | 17 | 26 | 7 |
| | | AVM | RWGA | 14 | 32 | 4 |
| | | WBGA | RWGA | 5 | 42 | 3 |
| TANK | RQ1 | RS | GREEDY | 2 | 1 | 47 |
| | | RS | AVM | 1 | 2 | 47 |
| | | RS | WBGA | 1 | 4 | 45 |
| | | RS | RWGA | 1 | 3 | 46 |
| | RQ2 | GREEDY | AVM | 3 | 6 | 41 |
| | | GREEDY | WBGA | 4 | 21 | 25 |
| | | GREEDY | RWGA | 3 | 12 | 35 |
| | | AVM | WBGA | 1 | 6 | 43 |
| | | AVM | RWGA | 2 | 5 | 43 |
| | | WBGA | RWGA | 1 | 3 | 46 |
| DC eng | RQ1 | RS | GREEDY | 0 | 0 | 40 |
| | | RS | AVM | 39 | 1 | 0 |
| | | RS | WBGA | 31 | 5 | 4 |
| | | RS | RWGA | 31 | 4 | 5 |
| | RQ2 | GREEDY | AVM | 40 | 0 | 0 |
| | | GREEDY | WBGA | 40 | 0 | 0 |
| | | GREEDY | RWGA | 40 | 0 | 0 |
| | | AVM | WBGA | 0 | 0 | 40 |
| | | AVM | RWGA | 0 | 0 | 40 |
| | | WBGA | RWGA | 6 | 33 | 1 |

Table A.9: Results for the Spearman's rank correlation test, which measures the correlation of the NFRCT metric with respect to the test suite size. Notice that a negative $\rho$ means an improve in the performance of the algorithm with a larger test suite.

| | | | Product 1 | | Product 2 | | Product 3 | | Product 4 | | Product 5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value | $\rho$ | p-value |
| ACC | HiL | RS | -0.265 | <0.0000 | 0.123 | 0.00608 | 0.262 | <0.0000 | 0.226 | <0.0000 | 0.155 | 0.00052 |
| | | GREEDY | -0.685 | <0.0000 | -0.165 | 0.00022 | -0.182 | 0.00004 | -0.195 | 0.00001 | 0.024 | 0.58751 |
| | | AVM | -0.129 | 0.00395 | -0.126 | 0.00491 | -0.056 | 0.20819 | -0.149 | 0.00082 | -0.131 | 0.00327 |
| | | WBGA | 0.026 | 0.56781 | 0.085 | 0.05695 | 0.079 | 0.07932 | 0.151 | 0.00071 | 0.086 | 0.05446 |
| | | RWGA | 0.061 | 0.17604 | 0.178 | 0.00006 | 0.098 | 0.02778 | 0.003 | 0.94266 | 0.138 | 0.00204 |
| UAV | HiL | RS | 0.446 | <0.0000 | -0.454 | <0.0000 | 0.193 | 0.00001 | 0.200 | 0.00001 | -0.252 | <0.0000 |
| | | GREEDY | 0.798 | <0.0000 | 0.798 | <0.0000 | -0.287 | <0.0000 | -0.287 | <0.0000 | -0.287 | <0.0000 |
| | | AVM | 0.257 | <0.0000 | -0.423 | <0.0000 | -0.216 | <0.0000 | -0.057 | 0.20256 | -0.327 | <0.0000 |
| | | WBGA | 0.134 | 0.00267 | 0.103 | 0.02093 | 0.283 | <0.0000 | 0.289 | <0.0000 | -0.412 | <0.0000 |
| | | RWGA | 0.012 | 0.78057 | -0.109 | 0.01504 | 0.326 | <0.0000 | 0.299 | <0.0000 | -0.375 | <0.0000 |
| TANK | HiL | RS | -0.237 | <0.0000 | -0.722 | <0.0000 | -0.785 | <0.0000 | -0.437 | <0.0000 | -0.502 | <0.0000 |
| | | GREEDY | -0.798 | <0.0000 | -0.921 | <0.0000 | -0.921 | <0.0000 | -0.809 | <0.0000 | -0.931 | <0.0000 |
| | | AVM | -0.362 | <0.0000 | -0.605 | <0.0000 | -0.769 | <0.0000 | -0.288 | <0.0000 | -0.323 | <0.0000 |
| | | WBGA | -0.266 | <0.0000 | -0.629 | <0.0000 | -0.647 | <0.0000 | -0.292 | <0.0000 | -0.389 | <0.0000 |
| | | RWGA | -0.233 | <0.0000 | -0.575 | <0.0000 | -0.594 | <0.0000 | -0.259 | <0.0000 | -0.299 | <0.0000 |
| DC Engine | HiL | RS | 0.078 | 0.08260 | 0.224 | <0.0000 | 0.250 | <0.0000 | 0.284 | <0.0000 | | |
| | | GREEDY | -0.697 | <0.0000 | -0.697 | <0.0000 | -0.745 | <0.0000 | -0.733 | <0.0000 | | |
| | | AVM | 0.673 | <0.0000 | 0.758 | <0.0000 | 0.776 | <0.0000 | 0.844 | <0.0000 | | |
| | | WBGA | 0.369 | <0.0000 | 0.470 | <0.0000 | 0.513 | <0.0000 | 0.641 | <0.0000 | | |
| | | RWGA | 0.421 | <0.0000 | 0.493 | <0.0000 | 0.532 | <0.0000 | 0.598 | <0.0000 | | |